UNIVERSIDAD DE CANTABRIA
FACULTAD DE CIENCIAS
DEPARTAMENTO DE ELECTRONICA

PLANIFICACION, ANALISIS Y OPTIMIZACION DE SISTEMAS DISTRIBUIDOS DE TIEMPO REAL ESTRICTO

MEMORIA
presentada para optar al grado de
DOCTOR EN CIENCIAS FISICAS
por
José Javier Gutiérrez García
Licenciado en Ciencias, Sección Físicas
Especialidad de Electrónica

UNIVERSIDAD DE CANTABRIA
FACULTAD DE CIENCIAS
DEPARTAMENTO DE ELECTRONICA

PLANIFICACION, ANALISIS Y OPTIMIZACION DE SISTEMAS DISTRIBUIDOS DE TIEMPO REAL ESTRICTO

MEMORIA
presentada para optar al grado de Doctor en Ciencias Físicas por el Licenciado en Ciencias
José Javier Gutiérrez García

El Director,

Dr. Michael González Harbour
Profesor Titular de Universidad

DECLARO:
Que el presente trabajo ha sido realizado en el Departamento de Electrónica de la Universidad de Cantabria, bajo mi dirección y reúne las condiciones exigidas a los trabajos de Doctorado.

Santander, Septiembre de 1995

Fdo. J. J. Gutiérrez García

Fdo. Michael González Harbour

A Raquel,
a mis padres,
y a mi tía.

Deseo expresar mi agradecimiento a todas aquellas personas que de una u otra manera han hecho posible la realización de este trabajo.

En primer lugar, a Michael González por su inestimable ayuda como tutor, y como director de esta Tesis Doctoral. En todo momento ha sido una magnífica guía para la elaboración de este trabajo, con sus valiosos comentarios y sus oportunos consejos.

A Raquel por su intenso, constante e incondicional apoyo, y por su desinteresada y obligada colaboración en la corrección de estilo de esta memoria (confío en que al menos las comas estarán en su sitio).

A los compañeros del Grupo de Computadores y Tiempo Real (y también de otros grupos) con los que tantos cafés, charla y discusión he compartido, por su continuado ánimo y aliento.

A José María Drake por brindarme la oportunidad de dar mis primeros (y también los sucesivos) pasos en la faena investigadora.

A José Angel Gregorio por avalar año tras año los interminables informes anuales de mi Beca Ministerial.

A la empresa Equipos Nucleares S.A., bajo cuyos auspicios llevé a cabo mi primer trabajo de investigación, y especialmente a Fernando Linares por la revisión crítica de ese primer trabajo.

A mis padres y a mi tía por su calurosa y eterna hospitalidad, y por su paciencia para soportarme (absolutamente), sobre todo, en la última fase de la realización de esta Tesis.

Finalmente, quiero hacer extensivo este agradecimiento a mis amigos, a los que espero no haber aburrido demasiado con este Tema.

A todos, gracias.

La presente Tesis Doctoral, ha sido desarrollada en el marco de los siguientes proyectos de investigación:

"Extensión de la teoría RMA al diseño y análisis de sistemas de tiempo real gobernados por eventos".
Proyecto del Plan Nacional de Investigación en Automatización Avanzada y Robótica, 1992-1994, Ref. ROB91-0288.

"Herramientas para el análisis de tiempo real de sistemas distribuidos para automatización industrial".
Proyecto del Plan Nacional de Investigación en Tecnologías Avanzadas de la Producción, 1994-1997, Ref. TAP94-0996.

Indice de contenidos

1. Introducción

1.1. Sistemas de tiempo real 1-1
1.2. Sistemas distribuidos de tiempo real 1-4
1.3. Teoría *RMA* 1-6
1.3.1. Análisis de tareas periódicas e independientes 1-6
1.3.2. Primeras extensiones de la teoría *RMA* básica 1-8
1.3.3. Inversión de Prioridad 1-10
1.3.4. Tratamiento de eventos aperiódicos 1-13
1.3.5. Plazos superiores al período 1-15
1.3.6. Tareas compuestas por secuencias de acciones 1-17
1.3.7. Tratamiento del *jitter* 1-18
1.3.8. Análisis en sistemas distribuidos 1-19
1.4. Objetivos de este trabajo 1-19
1.4.1. Metodología de análisis y diseño 1-21
1.4.2. Implementación práctica en Ada 1-23
1.5. Organización de esta memoria de Tesis Doctoral 1-24

2. Análisis de tiempo real para sistemas distribuidos

2.1. Introducción 2-1
2.2. Modelo lineal de sistema distribuido 2-2
2.3. Análisis de tiempo real 2-7
2.4. Problemática derivada del análisis 2-11

3. Optimización de la asignación de prioridades en sistemas distribuidos

3.1. Introducción 3-1
3.2. Planificación mediante *templado simulado* 3-3
3.3. Algoritmo heurístico de planificación 3-8
3.4. Implementación de las técnicas de planificación 3-14
3.4.1. Herramienta de planificación para sistemas distribuidos 3-14

3.4.2. Herramienta de generación automática de ejemplos 3-25
3.5. Comparación de las técnicas de planificación 3-28

4. Extensión del análisis de tiempo real y de la planificación de sistemas distribuidos
4.1. Introducción 4-1
4.2. Modelo general de sistema distribuido 4-2
4.3. Extensión del análisis de tiempo real 4-9
4.3.1. Modelo equivalente para el análisis 4-10
4.3.2. Análisis de tiempo real para el modelo general 4-19
4.4. Algoritmos de planificación para el modelo general 4-23
4.4.1. Algoritmo heurístico 4-27
4.4.2. *Templado simulado* 4-29
4.5. Aplicación de la planificación al modelo general 4-31
4.5.1. Extensión de las herramientas de planificación y generación de ejemplos 4-32
4.5.2. Planificación de un sistema distribuido de tiempo real estricto . 4-35

5. Implementación de redes de comunicación de tiempo real estricto
5.1. Introducción 5-1
5.2. Colas de mensajes 5-3
5.3. Implementación de las colas de mensajes 5-10
5.3.1. Estructura general 5-12
5.3.2. Estructura de datos 5-17
5.3.3. Implementación de las operaciones 5-25
5.3.4. Aspectos dependientes de la implementación 5-30
5.4. Implementación de las comunicaciones de tiempo real 5-31
5.4.1. Protocolo de comunicaciones 5-32
5.4.2. Gestión de redes 5-38
5.4.3. Comunicaciones mediante líneas punto a punto 5-39
5.4.4. Comunicaciones a través del bus VME 5-43

6. Servidor esporádico para comunicaciones

6.1. Introducción 6-1
6.2. Estudio de los efectos del *jitter* sobre la planificabilidad 6-3
6.3. Modelo del servidor esporádico para comunicaciones 6-5
6.4. Implementación del servidor esporádico para comunicaciones 6-11
6.4.1. Implementación en redes punto a punto 6-14
6.4.2. Implementación en redes multipunto 6-24
6.5. Aplicación a las redes implementadas del servidor esporádico para comunicaciones 6-31

7. Modelado de las redes de comunicación
7.1. Introducción 7-1
7.2. Modelado de las colas de mensajes 7-2
7.2.1. Modelado de las operaciones sobre las colas de mensajes 7-5
7.2.2. Modelado de las comunicaciones 7-14
7.3. Evaluación de las colas de mensajes 7-20
7.3.1. Operaciones locales 7-22
7.3.2. Operaciones remotas en el nudo actual 7-26
7.3.3. Operaciones remotas en el nudo remoto 7-30
7.3.4. Comunicaciones 7-35
7.4. Modelado y evaluación del servidor esporádico para comunicaciones . 7-41
7.4.1. Evaluación del servidor esporádico para comunicaciones 7-42
7.4.2. Evaluación de las comunicaciones que utilizan servidor esporádico 7-45
7.4.3. Estudio comparativo de las comunicaciones que utilizan servidor esporádico 7-48
7.5. Ejemplos 7-55
7.5.1. Sistema monoprocesador de tiempo real estricto 7-58
7.5.2. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones punto a punto 7-63
7.5.3. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones multipunto 7-71
7.5.4. Aplicación de un sistema multiprocesador distribuido de tiempo real estricto 7-78

8. Conclusiones

8.1. Revisión de los objetivos 8-1

8.2. Contribuciones de este trabajo 8-5

8.3. Trabajo futuro 8-7

A. Implementación en Ada de las redes de comunicación

A.1. Especificaciones de los paquetes Ada A-1

A.2. Estructura de datos de las colas de mensajes A-14

A.3. Operaciones sobre las colas de mensajes A-20

A.4. Implementación de las comunicaciones A-32

B. Terminología del modelado de las redes de comunicación

B.1. Lista de términos y significados B-1

B.2. Correspondencias de las acciones B-7

Bibliografía

1. *Introducción*

1.1. Sistemas de tiempo real

Los sistemas de tiempo real son sistemas computacionales en los que el computador está fuertemente ligado a un entorno que cambia con el tiempo. Por esta causa, en los sistemas de tiempo real el funcionamiento correcto no sólo depende de los resultados del cálculo, sino también del instante de tiempo en el que estos cálculos son finalizados [STA88]. A este tipo de sistemas pertenecen muchos de los computadores y del software destinados a aplicaciones como los sistemas de control empotrados —por ejemplo controladores de robots, máquinas industriales, etc.—, la transmisión y el procesado de imagen —visión artificial en tiempo real—, sistemas de control y navegación de vehículos —como automóviles, aviones, o naves espaciales—, etc. Gran parte de estas aplicaciones se implementan hoy en día como sistemas multiprocesadores y distribuidos con ciertos requisitos temporales que deben satisfacer.

En la mayoría de los sistemas de tiempo real (tanto monoprocesadores como multiprocesadores) las actividades que el sistema debe ejecutar se conocen por adelantado, ya que tenemos un determinado código activo en la memoria del computador que está preparado para ejecutar. La ejecución de cada una de las actividades que se realizan en el sistema es disparada por la llegada de un estímulo llamado *evento*, y la actividad propiamente dicha que se realiza debido a la llegada de un evento se conoce como la *respuesta* a ese evento. Estas respuestas están formadas por diferentes actividades, cada una de las cuales constituye una *acción*. La ejecución de la respuesta (y algunas veces de las acciones individuales) puede tener impuestos ciertos requerimientos temporales que debe cumplir. En muchos casos estos requerimientos se refieren al máximo tiempo del que dispone la respuesta para su finalización desde la llegada del evento de disparo. Este tipo de requerimiento se conoce con el nombre de *plazo* (*deadline*).

En general, los sistemas de tiempo real constan de un conjunto de actividades que son planificables de manera independiente. Los eventos que llegan al sistema pueden generar la realización de actividades periódicas (se ejecutan a intervalos regulares) o aperiódicas (se ejecutan a intervalos no regulares), y pueden tener impuestos plazos *estrictos* (el sistema se considera que falla si no cumple con alguno de sus plazos) o *no estrictos* (el sistema puede incumplir alguno de sus plazos de vez en cuando, produciendo así una disminución de las prestaciones o de la calidad del sistema, pero no un fallo total). En muchos casos de sistemas de tiempo real estricto, el incumplimiento de los plazos puede conllevar pérdidas económicas importantes, e incluso la pérdida de vidas humanas en el caso de los sistemas de seguridad crítica.

La programación de los sistemas de tiempo real estricto es muy diferente de la de los sistemas de cálculo convencionales. Mientras que en éstos las medidas más habituales de las cualidades de un programa o de un algoritmo son el *throughput*, o capacidad media de procesado de información y el tiempo de respuesta *promedio*, en un sistema de tiempo real estricto las prestaciones medias son un factor secundario, y los factores realmente importantes son la *planificabilidad* (habilidad para cumplir todos los plazos) y el tiempo de respuesta en el *peor caso*.

Tradicionalmente, una de las técnicas más habituales para el diseño de sistemas de tiempo real estricto ha venido siendo el empleo de ejecutivos cíclicos [BAK88]. En esta aproximación, a cada una de las tareas o actividades concurrentes que se desean ejecutar se le asigna una *rodaja* temporal durante la cual se puede ejecutar. Estas rodajas temporales se ejecutan de forma periódica o cíclica en un orden preestablecido, e intercaladas adecuadamente para garantizar el cumplimiento de todos los requerimientos temporales.

Aunque esta aproximación es fácilmente abordable para sistemas pequeños, se hace rápidamente intratable cuando los sistemas van aumentando su complejidad. En un sistema con ejecutivos cíclicos los programadores disponen de una rodaja temporal que no pueden sobrepasar, y deben adaptar su programa y su diseño a estas restricciones temporales. En muchas ocasiones, una actividad que no "cabe" dentro de su rodaja temporal debe ser fragmentada en dos o más partes que se asignan a distintas rodajas. Esta dificultad en el diseño hace a los programadores sacrificar la estructura del programa para adaptar el código

a la rodaja temporal correspondiente, lo que conduce a sistemas difíciles de entender y de mantener, y por consiguiente, que están en contra de los principios generales de la ingeniería software.

La técnica de ejecutivos cíclicos se corresponde con una política de planificación que se lleva a cabo en tiempo de compilación. Otro tipo de políticas de planificación utilizadas en los sistemas de tiempo real son las que se realizan en tiempo de ejecución. Estas políticas resultan más adecuadas que la anterior en sistemas de tiempo real complejos, y se basan en mecanismos expulsores y no expulsores por prioridades (estáticas o dinámicas). En la planificación no expulsora, si una tarea comienza su ejecución no se detiene hasta que finaliza, aunque haya otras tareas de mayor prioridad esperando. Esta situación puede producir un retraso significativo para las tareas de alta prioridad, que es posible evitar con una planificación expulsora en la que una tarea de alta prioridad expulsa de la CPU a una tarea de baja prioridad (aunque ésta última no haya finalizado). En la política de planificación con mecanismos expulsores por prioridades fijas, la prioridad de la tarea la fija el usuario y no cambia hasta que éste vuelve a fijar una nueva prioridad. En cambio, en las políticas basadas en prioridades dinámicas, éstas pueden variar dinámicamente en función de lo cerca que se encuentre una tarea de su plazo, o en función de alguna relación entre el tiempo que necesita para terminar su trabajo y el plazo que tiene asignado.

Las políticas de planificación para tiempo real deben cumplir principalmente los siguientes requisitos:

- Predecibilidad. En el sentido de la facilidad de analizar y de comprender el comportamiento temporal del sistema, sobre todo, en lo que respecta a la respuesta temporal de peor caso.

- Poder alcanzar utilizaciones altas de los recursos (y en especial de la CPU), a la vez que se garantiza el cumplimiento de los plazos.

- Asegurar una respuesta rápida a los eventos aperiódicos garantizando el cumplimiento de los plazos de todas las tareas del sistema.

- Producir un bajo aumento de la sobrecarga (*overhead*), es decir, del trabajo extra que la CPU dedica a realizar la planificación. Las políticas dinámicas producen en el sistema una mayor sobrecarga que las estáticas, ya que deben reordenar la cola de ejecución en cada decisión que toma el planificador.

- Mantener la estabilidad bajo situaciones de sobrecarga transitoria, es decir, asegurar que se verifican los plazos de las tareas críticas bajo estas condiciones.

- Estar disponible en una amplia gama de productos comerciales.

Las políticas de planificación expulsoras basadas en prioridades fijas cumplen con todos estos requisitos y constituyen una excelente elección para la construcción de aplicaciones prácticas.

1.2. Sistemas distribuidos de tiempo real

Los sistemas multiprocesadores y distribuidos de tiempo real tienen una importancia creciente en los sistemas de control de hoy en día, sobre todo, debido a que el bajo coste de las redes de comunicación permite la interconexión de múltiples dispositivos y de sus controladores en un gran sistema.

La arquitectura típica de un sistema de control distribuido de tiempo real consta de varios nudos procesadores interconectados a través de una o más redes de comunicación. El software del sistema se compone de varias tareas concurrentes que normalmente están localizadas de forma estática en los nudos procesadores, ya que los efectos de una localización dinámica son muy difíciles de predecir para sistemas de tiempo real estricto. Normalmente, la comunicación entre las tareas del sistema (dentro del propio nudo o entre diferentes nudos) se realiza a través del intercambio de mensajes. También es normal que los mensajes intercambiados entre tareas de diferentes nudos se asignen de forma estática a las redes de comunicación. Además, existe la posibilidad de que las tareas de un mismo nudo compartan datos o recursos mediante los mecanismos de sincronización habitualmente utilizados en sistemas de memoria compartida.

Una vía común para la activación de las tareas del sistema distribuido puede ser bien la llegada de un evento de disparo, que puede ser generado por un dispositivo externo del entorno de la implementación (como un temporizador), o bien la llegada de un mensaje enviado por otra tarea del sistema. Muchos de estos eventos tienen asociados ciertos requisitos temporales como el *plazo de principio-a-fin*, que es el máximo tiempo de respuesta permitido desde la llegada del evento hasta el final de la ejecución de su respuesta. En la ejecución de la respuesta a un evento se pueden ver implicadas varias tareas localizadas en diferentes nudos que se intercambian mensajes a través de las redes del sistema distribuido. También es posible imponer requisitos temporales a las acciones individuales que componen la respuesta a un evento.

Todas las cuestiones que se plantean en los sistemas monoprocesadores cuando se les imponen requisitos temporales se ven aumentadas tanto cualitativa como cuantitativamente cuando trabajamos con sistemas distribuidos de tiempo real. Para este tipo de sistemas debemos considerar principalmente los siguientes puntos:

- Análisis para determinar la planificabilidad de los recursos. Aparecen nuevos problemas en el análisis para el cálculo de las respuestas de peor caso y su comparación con los plazos de principio-a-fin; estos problemas se derivan del hecho de que la respuesta temporal en unos recursos influye en la respuesta en los demás recursos.

- Suspensión de una respuesta. Es necesario considerar el caso en el que las tareas se autosuspenden en espera de algún mensaje de un recurso remoto.

- Inversión de prioridad. Aparecen nuevas fuentes de inversión de prioridad, como la debida a la sincronización en multiprocesadores para el uso de memoria compartida (bloqueo remoto).

- Asignación de prioridades a las acciones y de acciones a los recursos. En un sistema distribuido se debe decidir por un lado cómo asignar las tareas a los nudos procesadores y los mensajes a las redes de comunicación, y por otro cómo asignar prioridades a las tareas y a los mensajes para que el sistema cumpla los requisitos de tiempo real.

Para terminar, hay que poner de manifiesto la importancia que tienen las redes de comunicación en el conjunto del sistema distribuido del que forman parte. No todas las redes de comunicación se pueden utilizar en sistemas de tiempo real, por lo que hay que buscar aquellas redes en las que sea posible garantizar el cumplimiento de los requisitos temporales impuestos.

1.3. Teoría *RMA*

La teoría *RMA* (Rate Monotonic Analysis) es una técnica matemática para el análisis de tiempo real de las tareas que integran un sistema planificado por prioridades fijas, cuyo objetivo es determinar su planificabilidad. Esta teoría tiene sus orígenes en un trabajo llevado a cabo por Liu y Layland en 1973 [LIU73] en el que introdujeron el algoritmo *RMS* (Rate Monotonic Scheduling), que es un algoritmo de planificación con asignación óptima de prioridades estáticas. Este algoritmo da mayor prioridad a las tareas con períodos más pequeños (mayores frecuencias). El trabajo de Liu y Layland sólo era aplicable a sistemas extremadamente restrictivos y tenía por tanto una reducida aplicación práctica. Sin embargo, sirvió como base para un gran número de trabajos posteriores en los que se fueron eliminando las restricciones impuestas, hasta conseguir un cuerpo teórico con el que hoy somos capaces de analizar sistemas de tiempo real estricto basados en prioridades fijas [KLE93].

1.3.1. Análisis de tareas periódicas e independientes

Los resultados principales de la teoría *RMA* están basados en dos conceptos sencillos que ayudan a construir la situación de peor caso para un conjunto de tareas:

Instante crítico. El tiempo de respuesta de peor caso para cualquier tarea de un conjunto dado se obtiene cuando todas las tareas comienzan a ejecutar a la vez.

Chequeo del primer plazo. Cuando todas las tareas se activan a la vez, si una tarea cumple su primer plazo, cumplirá siempre todos sus plazos. En otras palabras, el peor caso para el tiempo de respuesta de una tarea corresponde a la primera activación después de un instante crítico.

Los resultados que obtuvieron Liu y Layland [LIU73] solamente se podían aplicar si todas las tareas eran periódicas con plazos al final de cada período, y además eran perfectamente expulsables e independientes (no interaccionaban unas con otras). En estas condiciones, probaron que la asignación óptima de prioridades es la *rate monotonic*, es decir, aquella en la que las prioridades se asignan de acuerdo con los períodos de las tareas (mayor prioridad a los períodos más pequeños).

Otro de los resultados importantes es el postulado por el que un sistema constituido por un conjunto de n tareas periódicas e independientes, y planificado de acuerdo con el algoritmo *RMS* cumplirá siempre sus plazos, para fases cualesquiera de todas las tareas, si se verifica la siguiente inecuación:

$$\frac{C_1}{T_1} + \ldots + \frac{C_n}{T_n} \leq U(n) = n\,(2^{\frac{1}{n}} - 1)$$

donde,

C_i Es el tiempo de ejecución de peor caso de la tarea τ_i.

T_i Es el período de la tarea τ_i (tiempo transcurrido entre dos activaciones consecutivas de la tarea).

$U(n)$ Es el límite de utilización del procesador para n tareas.

La utilización U_i de la CPU para la tarea τ_i viene dada por C_i / T_i. La fase inicial ϕ_i de la tarea τ_i se define como el instante en el que la tarea se activa por primera vez. Uno de los aspectos importantes de la ecuación es la independencia del resultado respecto de las fases ϕ_i. Esto es extremadamente importante, ya que en la mayoría de los sistemas con planificación en tiempo de ejecución, las fases no se conocen y son arbitrarias. La ecuación descrita se conoce como *test de límite de utilización*, y representa una condición suficiente (pero no necesaria) que garantiza la planificabilidad de un conjunto de tareas bajo las condiciones del algoritmo *RMS*.

Los principios de instante crítico y chequeo del primer plazo nos permiten también hacer un cálculo exacto del tiempo de ejecución de peor caso para cualquier tarea de un conjunto de ellas independientes y periódicas, independientemente de la asignación de prioridades que se utilice. Si todas las tareas (periódicas e independientes) verifican sus plazos cuando comienzan a ejecutar al mismo tiempo, se puede decir que los verificarán siempre,

es decir, para cualquier combinación de instantes de comienzo de ejecución. Esto da lugar al desarrollo del *test de tiempos de ejecución* o *test exacto* que permite obtener el tiempo exacto de ejecución de peor caso para una tarea. Este tiempo se obtiene mediante una ecuación iterativa que va calculando el tiempo de ejecución necesario para finalizar todo el trabajo activado hasta el instante de la iteración anterior. La iteración termina cuando dos pasos consecutivos alcanzan el mismo resultado, lo cual indica que no queda más trabajo por realizar a ese nivel de prioridad. La ecuación se puede expresar de la forma siguiente [JOS86] [AUD93] [KLE93]:

$$r_i^{n+1} = C_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{r_i^{\,n}}{T_j} \right\rceil C_j$$

donde,

r_i Es el tiempo de respuesta de peor caso de la tarea τ_i al que converge la ecuación cuando $r_i^{n+1} = r_i^{\,n}$. El valor de arranque es $r_i^{\,0} = C_i$.

hp(i) Es el conjunto de tareas que pueden expulsar a la tarea τ_i, o sea, que tienen mayor prioridad.

C_i Es el tiempo de ejecución de peor caso de la tarea τ_i.

T_j Es el período de la tarea τ_j.

$\lceil x \rceil$ Es la función techo sobre x, es decir, redondeo de x al entero superior.

1.3.2. Primeras extensiones de la teoría *RMA* básica

Hasta este punto, los principios de la teoría *RMS* dan lugar a dos test diferentes de planificabilidad con las siguientes limitaciones:

- Todas las tareas corren en un solo procesador.
- Todas las tareas son periódicas y no interaccionan entre sí.
- Los plazos de todas las tareas se encuentran al final de cada período.
- No se usan interrupciones.
- No se tienen en cuenta los cambios de contexto.
- Las tareas no se autosuspenden.
- Para el test de límite de utilización se deben asignar prioridades *RMS*.

Una primera extensión de la teoría va encaminada a incorporar los cambios de contexto como una cantidad de tiempo que se añade tanto al principio, como al final de la ejecución de cada tarea. El tiempo de cambio de contexto es la suma de estos dos tiempos, y el modelado de una tarea que tenga en cuenta el cambio de contexto consiste en sumar a su tiempo de ejecución el correspondiente a dos cambios de contexto [KLE93].

Leung y Whitehead [LEU82] probaron que para un conjunto de tareas periódicas e independientes con plazos menores o iguales a sus respectivos períodos, la asignación óptima de prioridades es la *deadline monotonic*, es decir, aquella en la que las prioridades se asignan de acuerdo con los plazos de las tareas (mayor prioridad a los plazos más pequeños). La asignación de prioridades *RMS* es un caso particular de la asignación *deadline monotonic*; sin embargo por razones históricas a la teoría se la suele llamar *RMA* (Rate Monotonic Analysis).

En los sistemas de tiempo real es muy común la existencia de tareas con plazos anteriores al período, por ejemplo cuando se necesita una respuesta rápida a la llegada de un evento. Aunque alguno o todos los plazos asignados a las tareas sean anteriores a los períodos, aún siguen vigentes los criterios del instante crítico y chequeo del primer plazo. El test de límite de utilización no es aplicable, pero existe un nuevo test de límite de utilización para plazos anteriores al período [KLE93]. Sin embargo, el test de tiempos de ejecución es aplicable porque permite obtener los tiempos de ejecución para cualquier asignación de prioridades, y después éstos se pueden comparar con los plazos.

En las aplicaciones de tiempo real también es muy común la existencia de tareas dedicadas al procesado de interrupciones. Estas tareas a menudo violan la asignación óptima de prioridades. Normalmente, el procesado de interrupciones se realiza a un nivel de prioridad superior al de las tareas de aplicación. Así pues, aunque se asigne a la tarea de interrupción una prioridad más baja que a una tarea de aplicación, su prioridad real en la ejecución será mayor, y por lo tanto, podrá expulsar a las tareas del nivel de aplicación. El test de tiempo de ejecución es capaz de soportar este efecto al ser válido para cualquier asignación de prioridades. Sin embargo, el test de límite de utilización debe modificarse para incorporar el manejo de interrupciones u otras desviaciones de la asignación óptima de prioridades [KLE93].

1.3.3. Inversión de Prioridad

Un aspecto importante que hay que considerar en los sistemas de tiempo real es la *inversión de prioridad*, es decir, el efecto producido cuando una tarea es retrasada por otra de menor prioridad. La inversión de prioridad es una de las causas más frecuentes por las que una tarea puede incumplir su plazo. Hay muchas fuentes potenciales de inversión de prioridad, entre las que podemos considerar las siguientes:

- Regiones de código no expulsables por cuya ejecución debe esperar una tarea de prioridad superior.

- Colas FIFO en las que tareas de prioridad alta esperan la atención de sus peticiones (como por ejemplo llamadas de I/O) y deben esperar a que sean atendidas peticiones de menor prioridad que han llegado primero.

- Sincronización por recursos compartidos. Si una tarea reserva un recurso que comparte con otra tarea de mayor prioridad, esta última se ve forzada a esperar. Este es probablemente el más común de los efectos que causan inversión de prioridad, ya que la sincronización por recursos compartidos está presente en la práctica totalidad de los sistemas concurrentes de tiempo real.

El modelado de la inversión de prioridad se lleva a cabo añadiendo *tiempos de bloqueo* al test de planificabilidad. Estos tiempos de bloqueo tienen en cuenta los retrasos de peor caso que se pueden producir en la ejecución de las tareas debidos a la inversión de prioridad.

Uno de los casos más estudiados es la inversión de prioridad producida por la sincronización a través de recursos compartidos. Para modelar la sincronización se introduce el concepto de *sección crítica*, definida como una sección de código que utiliza un recurso compartido que una vez tomado no puede ser utilizado por otras tareas. De este modo, el modelo de sincronización (o más exactamente, la exclusión mutua) se basa en el concepto de sección crítica protegida por semáforos. Uno de los resultados más importantes de la teoría *RMA* consiste precisamente en poner de manifiesto el hecho de que el uso de semáforos normales puede producir retrasos extremadamente grandes a tareas de altas prioridad

[SHA90A] (efecto denominado *inversión de prioridad no acotada*). Este efecto se produce cuando una tarea de prioridad baja ha tomado un semáforo que protege a un recurso compartido con otra tarea de alta prioridad, y es expulsada por una de prioridad intermedia mientras la tarea de alta prioridad intenta tomar el semáforo.

A raíz del fenómeno de la inversión de prioridad no acotada producido en la sincronización, aparecen soluciones que tratan de evitarla. Estas soluciones son diversos *protocolos de sincronización* que limitan la cantidad de inversión de prioridad haciéndola igual a la duración de secciones críticas (que normalmente son cortas), y determinan el tiempo de bloqueo y su efecto en la planificación de las tareas. Para eliminar la inversión de prioridad no acotada podemos considerar los siguientes protocolos de sincronización en sistemas monoprocesadores [SHA90A] [RAJ89]:

Protocolo de no expulsión. No se permite la expulsión durante la ejecución de secciones críticas, es decir, se ejecutan a prioridad infinita. Este protocolo es capaz de prevenir el bloqueo mutuo salvo que las tareas se autosuspendan en el interior de una sección crítica, y además cumple la propiedad de bloquear una sola vez como máximo a cualquier segmento de código contiguo (o sea, sin suspensiones en el medio). Este protocolo presenta el problema de que no es capaz de evitar bloqueos innecesarios.

Protocolo de prioridad maximal o de protección de prioridad. La sección crítica se ejecuta a la prioridad de la tarea de mayor prioridad que pueda bloquear el semáforo. Las tareas de mayor prioridad pueden expulsar a la sección crítica (por definición no pueden intentar bloquear el semáforo). De la misma manera que el protocolo anterior también es capaz de prevenir el bloqueo mutuo (salvo que las tareas se autosuspendan en el interior de una sección crítica), y cumple la propiedad de bloquear como máximo una vez a cualquier segmento de código contiguo.

Protocolo de herencia básica de prioridad. La sección crítica se ejecuta a la prioridad de la tarea con mayor prioridad que está siendo bloqueada. Las tareas de mayor prioridad pueden expulsar a la sección crítica siempre que no haya

otras tareas con mayor prioridad aún intentando bloquear el semáforo. Este protocolo presenta como inconveniente el bloqueo encadenado (posibilidad de múltiples bloqueos a segmentos de código contiguo), y no es capaz de prevenir el bloqueo mutuo.

Protocolo de techo de prioridad. Es el protocolo de herencia básica de prioridad más una regla acerca de la atención a la petición de bloqueo sobre un semáforo libre: una tarea no puede bloquear un semáforo libre a menos que su prioridad sea estrictamente mayor que la del techo del sistema (máximo techo de todos los semáforos actualmente bloqueados por otras tareas). Este protocolo previene el bloqueo mutuo siempre y cumple la propiedad de bloquear como máximo una vez a segmentos de código contiguo. La implementación de este protocolo da lugar a una pequeña sobrecarga debido a que hay que guardar siempre el techo de prioridad del sistema.

El cálculo de los términos de bloqueo para los protocolos en los que se cumple la propiedad de bloquear como máximo una vez a segmentos de código contiguo consiste en obtener la duración máxima de todas las secciones críticas de tareas con prioridad inferior, cuyo techo de prioridad es mayor o igual que la prioridad de la tarea bajo análisis. Para el caso del protocolo de herencia básica de prioridad el cálculo de forma exacta de los tiempos de bloqueo es muy complejo, y se puede encontrar en [TOR92].

Si se consideran los términos de bloqueo se puede generalizar el test de límite de utilización dado en la página 1-7. Un sistema constituido por un conjunto de *n* tareas periódicas que utilizan uno de los protocolos anteriores puede ser planificado de acuerdo con el algoritmo *RMS* para cualesquiera fases de todas las tareas, si para cada tarea τ_i se verifica la siguiente inecuación [SHA90A]:

$$\frac{C_1}{T_1} + \ldots + \frac{C_i}{T_i} + \frac{B_i}{T_i} \leq U(i) = i\ (2^{\frac{1}{i}} - 1)$$

donde, B_i es el mayor bloqueo que puede experimentar la tarea τ_i (B_n es siempre cero, ya que la tarea de prioridad inferior no puede ser bloqueada por tareas de prioridad inferior, ya que no existen). Los demás términos son los de la fórmula anterior (ver página 1-7).

1.3.4. Tratamiento de eventos aperiódicos

Otra de las extensiones realizadas sobre la teoría *RMA* para poder abordar el análisis de sistemas reales, consiste en la incorporación al análisis de las tareas que no son periódicas. La aproximación que se sigue consiste en tratar a estas tareas como si fueran periódicas (pseudo-periódicas) o, si ello no es posible, planificarlas mediante algún método que permita conseguir que los efectos de su ejecución sobre tareas de prioridad inferior sean equivalentes a los de una tarea periódica. De esta forma, se pueden aplicar los test de planificación sin modificarlos en nada. Las tareas aperiódicas se activan a intervalos irregulares y se suelen usar para procesar eventos que ocurren aleatoriamente (eventos aperiódicos), como pueden ser los que se producen en las peticiones a un operador o los que generan los dispositivos de interrupción. En los eventos aperiódicos podemos considerar los patrones de llegada siguientes:

Llegadas limitadas. La cantidad máxima de eventos que pueden llegar en un intervalo de tiempo dado está limitada. En muchos casos hay restricciones físicas o dadas por el software que limitan el número de eventos que se pueden producir en un intervalo de tiempo dado. Por ejemplo, la acción de pulsar una tecla en una consola dispone de un circuito que garantiza un tiempo mínimo entre la llegada de dos eventos sucesivos. Los requerimientos temporales sobre las respuestas a este tipo de eventos pueden ser tanto estrictos como no estrictos.

Llegadas ilimitadas. La llegada de eventos se caracteriza por una función de probabilidad, pero puede suceder que lleguen un número ilimitado de eventos en un intervalo de tiempo arbitrario y muy pequeño. Este tipo de eventos únicamente pueden soportar requerimientos de tiempo real no estricto.

El objetivo en la planificación de tareas aperiódicas ilimitadas es conseguir que se verifiquen los plazos con requerimientos no estrictos (generalmente un tiempo de respuesta promedio) mientras se cumplen los plazos del resto de las tareas del sistema. Normalmente se usa una teoría de colas estocástica para estimar el tiempo de respuesta promedio a partir de un tiempo promedio de llegada de eventos. Además, se tiene que poder predecir el tiempo

de respuesta para el resto de las tareas aún cuando los eventos lleguen a un ritmo no predecible.

Tradicionalmente se han usado tres aproximaciones para la planificación de actividades aperiódicas:

Muestreo periódico (polling). Consiste en el chequeo periódico de los eventos que pueden haber llegado al sistema. Los tiempos de peor caso y de caso promedio son una función directa del período de chequeo, al igual que la sobrecarga que produce en el sistema. El mecanismo de muestreo periódico es predecible pero produce unos tiempos de respuesta lentos y/o una sobrecarga alta.

Procesado directo. Si se utiliza un manejador de interrupción se puede obtener una respuesta inmediata a un evento aperiódico. La respuesta promedio que se obtiene es muy buena pero con este método desaparece la predecibilidad para las tareas de menor prioridad si el ritmo de llegadas de eventos es ilimitado.

Procesado background. Los eventos aperiódicos son procesados por una tarea de muy baja prioridad. En este caso se mantiene la planificabilidad de las tareas de mayor prioridad, pero a cambio se hace muy difícil o imposible cumplir los requisitos temporales impuestos a los eventos aperiódicos, ya que deben esperar a que ejecuten todas las demás tareas.

El *servidor esporádico* [SPR89A] es una técnica para la planificación de eventos aperiódicos que combina la predecibilidad del *polling* con los cortos tiempos de respuesta del procesado directo de eventos. Es una solución extremadamente flexible para la planificación de una gran cantidad de actividades aperiódicas. El servidor esporádico reserva una cierta capacidad limitada de tiempo de ejecución para el procesado de eventos aperiódicos a un nivel de prioridad dado, de manera que garantiza el cumplimiento de los plazos del resto de las tareas del sistema. Un servidor esporádico controla la ejecución de una tarea aperiódica, y se caracteriza por dos parámetros que son: la capacidad de ejecución y el período de relleno. La capacidad de ejecución representa un espacio de tiempo durante el cual la tarea

aperiódica tiene derecho a ejecutar a la prioridad que se le ha asignado (el valor inicial representa la máxima capacidad del servidor). El período de relleno es el tiempo que debe transcurrir para que el servidor recupere la capacidad utilizada. Las reglas por las que se rige la actividad del servidor esporádico son las siguientes:

- Cuando ocurre un evento aperiódico, si hay capacidad de ejecución disponible, la tarea aperiódica ejecuta a la prioridad asignada consumiendo la correspondiente capacidad de ejecución.

- Cuando la capacidad de ejecución se termina, la tarea puede continuar ejecutando a un nivel de prioridad bajo (*background*).

- Cada porción de capacidad de ejecución consumida se rellena, es decir, se añade a la capacidad de ejecución actual después de que haya transcurrido un tiempo igual al período de relleno, contado a partir del instante en el que la porción de capacidad de ejecución consumida estaba lista para ejecutar.

- Cuando se produce un relleno y el servidor esporádico estaba ejecutando a prioridad de background, su prioridad se eleva al nivel normal.

1.3.5. Plazos superiores al período

Uno de los puntos que aún quedan por revisar en esta teoría es el caso en el que las tareas tienen los plazos después del final de su período. Este caso es especialmente importante en sistemas distribuidos en los que es normal que una actividad periódica tenga que recorrer varios recursos hasta su finalización. Aquí, el concepto de instante crítico todavía tiene validez, pero no es suficiente con chequear el primer plazo porque un trabajo anterior de la tarea τ_i podría retrasar una ejecución posterior. Lo que se debe determinar ahora es la longitud del *período de ocupación* de peor caso. Un período de ocupación para una tarea τ_i es el intervalo de tiempo durante el que el procesador está ocupado ejecutando tareas de prioridad igual o superior a la de la tarea τ_i. Si el plazo de la tarea τ_i es superior al período, los períodos de ocupación también pueden ser mayores que el período, y es preciso comprobar

entonces que se cumplen todos los plazos comprendidos en el período de ocupación de peor caso, que se produce después de un instante crítico.

La longitud del período de ocupación de peor caso se calcula mediante la ecuación iterativa del test de tiempos de ejecución (ver página 1-8), teniendo en cuenta que el tiempo de ejecución de peor caso (C_i) de la tarea τ_i bajo análisis puede influir varias veces. Así, el período de ocupación para la tarea τ_i se calcula mediante la siguiente ecuación utilizada de manera iterativa [LEH90] [TIN93D]:

$$w_i^{n+1}(q) = (q+1)\ C_i + \sum_{\forall\, j \in hp(i)} \left\lceil \frac{w_i^{\,n}(q)}{T_j} \right\rceil C_j$$

donde $w_i(q)$ es el tiempo de finalización de peor caso de la tarea τ_i para la *q-ésima* activación (después de un instante crítico) al que converge la ecuación cuando $w_i^{n+1}(q) = w_i^{\,n}(q)$. El valor con el que comienza la iteración es $w_i^{\,0}(q) = (q+1)\ C_i$. El resto de los términos que aparecen en la ecuación son los descritos en la página 1-8. Esta ecuación se aplica de forma sucesiva para valores de q iguales a 0,1,2,…, y termina cuando se verifica la siguiente desigualdad:

$$w_i(q) \leq (q+1)\ T_i$$

En ese momento se ha finalizado todo el trabajo de prioridad P_i o superior, y por consiguiente, se ha obtenido la longitud del período de ocupación de peor caso, igual a $w_i(q)$.

A partir de los resultados obtenidos para los distintos valores de q, se puede calcular la respuesta de peor caso para la tarea τ_i restando del tiempo de finalización de la activación *q-ésima*, el instante de activación correspondiente de la manera en que se muestra a continuación:

$$r_i = \max_{q=0,1,2,\ldots} (w_i(q) - qT_i)$$

Estos resultados no solamente son válidos para el cálculo de las respuestas de peor caso cuando los plazos son mayores que el período, sino que son aplicables para plazos

cualesquiera. Además se puede añadir el efecto de la sincronización por recursos compartidos modificando el cálculo de $w_i(q)$ de la forma en que se muestra a continuación:

$$w_i^{n+1}(q) = (q+1)\ C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{w_i^{\,n}(q)}{T_j} \right\rceil C_j$$

donde B_i es el término de bloqueo correspondiente a la tarea τ_i (mayor sección crítica de un semáforo cuyo techo de prioridad es mayor o igual que la prioridad de la tarea τ_i).

En el caso de que los plazos sean mayores que el período, ninguna de las dos asignaciones de prioridades (acorde al período o acorde a los plazos) es la óptima. Audsley propuso un *algoritmo de ordenación* [AUD91B] que consigue la asignación óptima de prioridades para el caso en el que los plazos sean arbitrarios. El algoritmo es óptimo en el sentido de que si existe una ordenación en la que todas las tareas son planificables, es capaz de encontrarla.

1.3.6. Tareas compuestas por secuencias de acciones

En un sistema de tiempo real complejo las tareas se pueden componer de acciones individuales (cada acción puede ser una tarea) que ejecutan secuencialmente, y que pueden tener asignadas prioridades diferentes. Algunas de las razones para considerar esta posibilidad son las siguientes:

- Algunas acciones pueden tener plazos individuales que deben cumplir (si una acción es muy urgente puede tener mayor prioridad que es resto de la secuencia).

- Puede ocurrir que para la sincronización se utilice el protocolo de protección de prioridad, que es un ejemplo de uso de segmentos de código con diferentes prioridades.

- Las secciones no expulsables son equivalentes a una acción que ejecuta a la mayor prioridad posible.

- Aumentar la planificabilidad. El análisis muestra que se puede incrementar la planificabilidad si se aumenta la prioridad de la parte final de una o más tareas de las que están ejecutando en el sistema.

- Un sistema puede estar formado por tareas que se intercambian eventos mediante un protocolo de sincronización de *para y espera* en el que las prioridades se han asignado en función de la importancia semántica de las tareas, y no de acuerdo con la teoría *RMA*. Un caso similar se da también en arquitecturas de paso de mensajes, en las que las prioridades de las tareas se asignan con objeto de controlar el tamaño máximo de las colas de mensajes.

González, Klein y Lehoczky obtuvieron algunos resultados para el análisis de tareas con estas características [GON91A]. Un concepto importante para analizar este tipo de tareas es lo que denominaron *forma canónica de una tarea*. Una tarea está en forma canónica cuando se compone de acciones consecutivas en las que no se decrementa la prioridad. Además, cualquier tarea se puede transformar a su forma canónica haciendo uso de un algoritmo que además no cambia su tiempo de respuesta. La transformación de una tarea a su forma canónica simplifica en gran medida el análisis (con respecto al de la tarea original). El análisis se basa en la obtención de la forma canónica para la tarea que se va a analizar, y en la clasificación del resto de las tareas (en su forma original) en diferentes categorías, según sean las prioridades de sus acciones comparadas con la prioridad de la primera acción de la forma canónica de la tarea que se analiza.

1.3.7. Tratamiento del *jitter*

Hasta este momento hemos supuesto que una tarea comienza su ejecución en el instante en el que se dispara el evento que la activa. En general, y sobre todo en sistemas distribuidos (como veremos más adelante), este no es el caso y las tareas pueden sufrir un retraso variable en su activación que recibe el nombre de *jitter*. Este efecto es necesario tenerlo en cuenta a la hora de calcular los tiempos de respuesta de peor caso. De este modo, se modifica la ecuación mediante la que se calcula el período de ocupación, para incorporar el *jitter* al efecto que las tareas τ_j de mayor prioridad tienen sobre la tarea τ_i, obteniéndose la siguiente expresión [TIN93D]:

$$w_i^{n+1}(q) = (q+1)\ C_i + B_i + \sum_{\forall j \in hp(i)} \left\lceil \frac{J_j + w_i^n(q)}{T_j} \right\rceil C_j$$

donde, J_j es el *jitter* correspondiente a la tarea τ_j, y el resto de los términos han sido descritos con anterioridad (ver páginas 1-8, 1-16 y 1-17).

1.3.8. Análisis en sistemas distribuidos

Por último, vamos a dar una breve reseña de los aspectos más importantes de la aplicación de la teoría *RMA* a sistemas distribuidos, ya que este tema será tratado con más profundidad en el capítulo 2. Para la aplicación de las técnicas derivadas de la teoría *RMA* a sistemas distribuidos [KLE93] [SHA93] [TIN94F] se modela cada red como si fuera un procesador y cada mensaje como si fuera una tarea. De esta forma se pueden asignar prioridades a las tareas y a los mensajes en estos sistemas utilizando algoritmos de optimización que tratan de buscar la asignación óptima de manera que las diferentes tareas y mensajes cumplan los plazos que se les ha asignado [TIN92] [GUT95A]. En esta misma línea se trata de resolver el problema de la asignación de acciones a recursos [TIN92]. Las técnicas de análisis deben tener en cuenta el efecto del *jitter*, que cobra una especial relevancia en este tipo de sistemas. Esta importancia se debe a que la respuesta a un evento periódico en un sistema distribuido puede estar formada por una secuencia de acciones localizadas en distintos recursos del sistema. La primera acción de la respuesta es posible que se inicie siempre de forma periódica; sin embargo, el resto de las acciones pueden tener activaciones no perfectamente periódicas, o lo que es lo mismo activaciones retrasadas (*jitter*).

1.4. Objetivos de este trabajo

Como vimos en los apartados precedentes de este mismo capítulo, el análisis y planificación de sistemas de tiempo real estricto está bastante bien resuelto para sistemas monoprocesadores en los que el software se puede modelar mediante conjuntos de tareas periódicas o aperiódicas simples [KLE93]. Sin embargo, tanto el análisis como la planificación en sistemas multiprocesadores y distribuidos de tiempo real estricto son aún campos abiertos a la investigación en lo que se refiere a la búsqueda de mejores soluciones, y también de soluciones más generales, es decir, que se puedan aplicar a un número mayor de sistemas

de este tipo. Prueba de ello es que buena parte de los trabajos más relevantes que se han publicado en este área lo han sido durante el período de realización de esta Tesis Doctoral [BUR93B] [SHA92] [SHA93] [TIN92] [TIN94A] [TIN94E] [TIN94F] [WAN93] [WED94] [SUN95].

Si nos centramos en las políticas de planificación basadas en prioridades fijas —que son una buena elección para el tipo de sistemas que estamos considerando—, encontramos algunas soluciones a los problemas del análisis y planificación de sistemas multiprocesadores y distribuidos de tiempo real que toman como base la teoría *RMA*. Estas soluciones son un tanto restrictivas debido principalmente a que los modelos utilizados para la representación del sistema son siempre modelos de conjuntos de tareas periódicas o aperiódicas simples y que no se sincronizan entre sí para intercambiarse eventos; normalmente sólo se tiene en cuenta la sincronización para el acceso mutuamente exclusivo.

Los objetivos que nos planteamos a la hora de realizar este trabajo van enfocados a la extensión de las técnicas existentes para la planificación y análisis de este tipo de sistemas de tiempo real estricto. Estos objetivos se pueden agrupar en dos grandes bloques, que son los siguientes:

Metodología de análisis y diseño. Se pretende optimizar los métodos existentes para la planificación y análisis de sistemas distribuidos de tiempo real, así como ampliar el campo de aplicación del análisis a sistemas más complejos en los que existe sincronización por intercambio de eventos o paso de mensajes. Con ello, se intentará desarrollar una metodología global de análisis y diseño de sistemas distribuidos de tiempo real estricto, así como de un conjunto de herramientas que faciliten su aplicación.

Implementación práctica en Ada. Se pretende demostrar que la metodología que se desarrolle se puede implementar en sistemas de tiempo real prácticos, para lo cual se abordará la implementación de los elementos necesarios que permitan la aplicación de esta metodología a sistemas distribuidos programados en lenguaje Ada. Dicha aplicación se apoya en el desarrollo e implementación

de una plataforma de trabajo capaz de responder a las características de tiempo real requeridas para este tipo de sistemas.

A continuación, describiremos de una forma más exhaustiva estos objetivos, y también las premisas y planteamientos en los que se fundamentan.

1.4.1. Metodología de análisis y diseño

El problema que se pretende abordar en el primer objetivo de este trabajo toma como punto de partida un sistema que responde a las siguientes especificaciones:

- Dispone de una arquitectura hardware correspondiente a un sistema distribuido compuesto por un conjunto de nudos procesadores conectados a través de un conjunto de redes de comunicación que son capaces de responder a requerimientos de tiempo real.

- Se compone de un conjunto de tareas periódicas o aperiódicas distribuidas por los nudos procesadores. Estas tareas se sincronizan entre sí y se intercambian mensajes o eventos a través de mecanismos locales como colas de mensajes, si están en el mismo nudo procesador, o a través de las redes de comunicación de tiempo real, si están en diferentes nudos procesadores.

- Tiene impuestos un conjunto de requerimientos estrictos y no estrictos de tiempo real que pueden ser:

  Simples. Son requerimientos relativos a los instantes de activación de las propias tareas o a los instantes de inicio de la transmisión de los propios mensajes considerados individualmente.

  Compuestos. Son requerimientos relativos a la ejecución secuencial de varias tareas junto con la transmisión de los posibles mensajes que se pueden enviar entre ellas.

La metodología de análisis y diseño pretende ofrecer soluciones basadas en la teoría *RMA* al problema del análisis y planificación de sistemas distribuidos de tiempo real estricto (con las especificaciones comentadas anteriormente). La resolución del problema que se plantea pasa necesariamente por la determinación de los siguientes aspectos del sistema:

Modelo de representación del sistema. Se necesita obtener un modelo que sea representativo de la mayor cantidad posible de este tipo de sistemas, y sobre el que se pueda definir una técnica de análisis que de una manera sencilla permita calcular la respuesta del sistema en el peor caso. Como hemos mencionado anteriormente, el modelo de representación del sistema que se ha utilizado hasta ahora es muy restringido y no tiene en cuenta los efectos de sincronización entre diferentes tareas (excepto la sincronización para el acceso mutuamente exclusivo). Para obtener este modelo, será preciso determinar los parámetros más importantes y la forma de representar la sincronización.

Método de análisis. También es necesario buscar una técnica que permita obtener la respuesta del sistema ante el peor caso para tener garantía analítica del cumplimiento o no de los requerimientos de tiempo real estricto. Un efecto muy importante que pretendemos estudiar es el causado por aquellas tareas que sufren retrasos irregulares en su activación (*jitter*). Una posible solución al problema que plantea el *jitter* en el análisis puede ser el uso de planificadores que lo eliminen, tanto en los procesadores como en las redes de comunicación (de esta manera la influencia en el análisis sería nula). Es preciso por tanto estudiar en qué casos puede compensar permitir el *jitter*, y en qué casos es necesario eliminarlo.

Asignación de prioridades. Una cuestión realmente importante en este tipo de sistemas consiste en determinar la forma de realizar la comunicación y sincronización entre estas tareas del modo más eficiente desde el punto de vista de los requerimientos de tiempo real estricto. Esto implica realizar la asignación de prioridades en el sistema tanto a las tareas que se ejecutan en cada nudo procesador, como a los mensajes que se transmiten por las redes de comunicación de tiempo real. Una posible aproximación a este problema

consistirá en desarrollar una técnica que convierta los requerimientos temporales (plazos) compuestos en simples para conseguir realizar una partición del sistema, y poder aplicar después las técnicas de asignación de prioridades que se utilizan en este momento en sistemas monoprocesadores.

Como complemento de esta metodología, podemos considerar además la posibilidad de conseguir una *optimización de la planificación* en los sistemas distribuidos de tiempo real estricto, si tenemos en cuenta los posibles beneficios que se obtendrían con la eliminación del *jitter*. Una técnica que se puede utilizar en los planificadores de este tipo de sistemas es el *servidor esporádico* (tanto para las tareas como para los mensajes).

Por último, una vez concluido el desarrollo de las técnicas de análisis y planificación para estos sistemas de tiempo real se pretende elaborar un conjunto integrado de herramientas que permitan automatizar su aplicación.

1.4.2. Implementación práctica en Ada

El segundo objetivo importante de este trabajo es mostrar cómo aplicar en la práctica la metodología desarrollada. Se pretenden marcar directrices de diseño para la programación en lenguaje Ada de sistemas distribuidos de tiempo real. Esta implementación práctica en Ada se centrará en los siguientes aspectos:

Comunicaciones de tiempo real. Se desarrollará un software de interfase que implemente la comunicación entre procesos con características de tiempo real estricto. Esta comunicación se deberá realizar de forma transparente al hecho de que los procesos estén localizados en el mismo o en diferentes nudos del sistema. En este punto se pretende aplicar un planificador basado en el *servidor esporádico* para la transmisión de mensajes con motivo de conseguir eliminar el *jitter*.

Banco de trabajo. Se desarrollará un banco de trabajo para la realización de pruebas. Para ello, se implementará la comunicación entre procesadores mediante técnicas de intercambio de mensajes con requerimientos de tiempo

real estricto. Se utilizará el lenguaje Ada y dos plataformas hardware: un sistema multiprocesador basado en el bus VME, y un conjunto de procesadores conectados por medio de múltiples líneas de comunicación punto a punto. Se pretende también que todos los tipos de comunicación que se implementen (en nuestro caso bus VME y líneas punto a punto) puedan utilizarse simultáneamente.

Aplicación de una metodología de diseño. Se desarrollará una metodología de diseño de este tipo de sistemas en lenguaje Ada, cuya demostración y evaluación se realizará sobre el banco de trabajo. La metodología se comprobará mediante alguna aplicación concreta de tiempo real estricto.

Así pues, los objetivos principales de esta implementación práctica en Ada son los siguientes:

- Demostrar la metodología de análisis y diseño de sistemas distribuidos de tiempo real estricto aplicándola a un lenguaje de programación y a un entorno concreto.

- Evaluar la metodología estudiando su adaptación a aplicaciones típicas.

- Demostrar la posibilidad de utilizar un bus VME o líneas punto a punto para realizar la comunicación en tiempo real estricto.

- Proponer una forma de aplicar la metodología para un entorno universalizado como es el de la programación en Ada de sistemas de tiempo real.

1.5. Organización de esta memoria de Tesis Doctoral

Este apartado lo dedicaremos a revisar los objetivos marcados en esta Tesis Doctoral en relación con los capítulos de la memoria en los que se abordan. A grandes rasgos, el capítulo 2 se centra en las actuales técnicas de análisis que se encuentran en la bibliografía y que se utilizan en esta Tesis. Los capítulos 3, 4 y 6 cubren el primero de los objetivos principales planteados en este trabajo, es decir, el desarrollo de una metodología de análisis

y diseño de sistemas distribuidos de tiempo real estricto. En los capítulos 5 y 7 se trata fundamentalmente el segundo de los objetivos principales que es la implementación práctica en Ada de la metodología desarrollada. A continuación vemos con más detalle la organización de esta memoria a través de cada uno de los capítulos que la componen.

En el capítulo 2 revisamos el modelo encontrado en la bibliografía para la descripción de sistemas distribuidos de tiempo real estricto gobernados por eventos sobre el que se van a poder aplicar las técnicas de análisis existentes. Lo denominamos *modelo lineal*, en contraposición al *modelo general* menos restrictivo que se desarrolla en el capítulo 4. También describimos la aplicación de las técnicas de análisis basadas en *RMA* a este *modelo lineal* definido, y planteamos la problemática que se deriva del análisis para estos sistemas de tiempo real.

En el capítulo 3 nos centramos en la resolución de los problemas de la planificación, tanto de los procesadores como de las redes de comunicación que los interconectan para sistemas distribuidos que respondan al *modelo lineal* definido en el capítulo 2. En este capítulo resolvemos el problema de la asignación de prioridades a tareas y mensajes mediante el desarrollo de un *nuevo algoritmo de planificación heurístico* que permite la optimización de las soluciones. También aplicaremos una técnica de propósito general (*templado simulado*) para resolver este problema mejorando las soluciones propuestas en la bibliografía. Con objeto de poder evaluar y comparar los algoritmos desarrollados, realizamos una herramienta de planificación en la que se implementan estos algoritmos desarrollados, y una herramienta de generación automática de ejemplos que nos permite su aplicación a un elevado número de casos. Finalmente, llevamos a cabo la comparación de los algoritmos a través de la aplicación práctica de los mismos.

En el capítulo 4 extendemos el *modelo lineal* de sistema distribuido a un *modelo general* que permite representar la sincronización por combinaciones de eventos. Esta contribución es de gran importancia para poder ampliar el tipo de sistemas de tiempo real que somos capaces de analizar actualmente. Además, desarrollamos una nueva técnica de análisis que nos permite calcular los tiempos de respuesta en el peor caso para este modelo. También extendemos los algoritmos de asignación de prioridades para que sean aplicables sobre este modelo, y de la misma forma, extendemos las herramientas de asignación de prioridades y

de generación automática de ejemplos. Por último, realizamos un estudio del comportamiento de los algoritmos mediante su aplicación práctica.

En el capítulo 5 llevamos a cabo una *implementación en Ada de redes de comunicación de tiempo real estricto*, y desarrollamos una plataforma de trabajo para la realización de pruebas. En la elaboración de las redes de comunicación implementadas se utiliza un mecanismo de paso de mensajes que permite la comunicación entre tareas con independencia del nudo en el que estén localizadas. Implementamos las redes de comunicación de tiempo real para líneas punto a punto (líneas serie RS-232 y RS-485) y para líneas multipunto (bus VME), construyendo el banco de trabajo sobre una plataforma hardware basada en tarjetas procesadoras sobre un bus VME por un lado, y en PC's por el otro.

En el capítulo 6 aplicamos las herramientas de planificación y generación de ejemplos al estudio del impacto que tiene el *jitter* en los sistemas distribuidos de tiempo real. Comprobaremos la enorme influencia que tiene en este tipo de sistemas, y la imperiosa necesidad de eliminarlo para conseguir mayores utilizaciones de los recursos. Para solucionar este problema, proponemos la utilización del *servidor esporádico* tanto en los procesadores como en las redes de comunicación, y además, definimos el *servidor esporádico para comunicaciones* como mecanismo de control en la transmisión de mensajes por las redes de comunicación de tiempo real. Desarrollamos el servidor esporádico para redes punto a punto y para redes multipunto, y lo incorporamos a las redes de tiempo real implementadas en el capítulo 5.

En el capítulo 7 modelamos las redes de comunicación implementadas para hacer posible el análisis y la planificación de los sistemas que las utilizan. En este modelado se definen los modelos de representación de las operaciones que se realizan en las redes, y se evalúan los tiempos de peor caso para esas operaciones. También se modela el servidor esporádico para comunicaciones en los mismos términos que las redes de comunicación y se estima la sobrecarga que supone su utilización para el sistema. Para terminar, elaboramos algunos ejemplos en los que se pueden ver algunas aplicaciones del uso de las redes de comunicación implementadas.

El capítulo 8 corresponde a las conclusiones extraídas del trabajo que se presenta en esta Tesis Doctoral. Además, añadimos dos apéndices en los que se presentamos las características específicas de la implementación en Ada de las redes de comunicación de tiempo real (apéndice A), y una relación de la terminología utilizada en el modelado de estas redes (apéndice B). En último término detallamos la bibliografía.

2. *Análisis de tiempo real para sistemas distribuidos*

2.1. Introducción

El software de un sistema distribuido consta de un conjunto de tareas concurrentes que a menudo están localizadas estáticamente en los nudos procesadores, ya que las consecuencias de una localización dinámica son difíciles de predecir para los sistemas de tiempo real estricto, y además muchas tareas están directamente ligadas a determinados elementos hardware. Cada tarea puede intercambiar mensajes con otras tareas ya sea en el mismo nudo, o ya entre nudos diferentes. Los mensajes intercambiados entre tareas de nudos diferentes también se localizan estáticamente en una red de comunicaciones particular. Las tareas localizadas en el mismo nudo pueden compartir datos u otros recursos mediante los mecanismos usuales de sincronización utilizados en los sistemas de memoria compartida.

Cada tarea del sistema se activa por la llegada de un evento de disparo que puede ser generado por el entorno, por un temporizador, o por la llegada de un mensaje de otra tarea. Muchos de estos eventos llevan asociados ciertos requisitos temporales, como el *plazo de principio-a-fin*, que es el máximo tiempo de respuesta permitido desde la llegada de un evento hasta que se ha completado. En la ejecución de la respuesta a un evento pueden estar implicadas varias tareas en uno o más nudos, y se pueden intercambiar además varios mensajes a través de las redes de comunicaciones. También se pueden poner requisitos temporales a las acciones individuales.

Para hacer posible la predicción de los tiempos de respuesta de peor caso, se pueden planificar las actividades en cada nudo procesador mediante una aproximación basada en prioridades que permita que las actividades con plazos más pequeños se procesen antes (a mayor prioridad) que las actividades con plazos mayores. Como vimos en el capítulo 1, una técnica de análisis que asegura la planificabilidad de los sistemas de tiempo real es la teoría

RMA, y está bien definida para sistemas monoprocesadores [KLE93]. Las técnicas *RMA* también se pueden aplicar a sistemas distribuidos [KLE93] [SHA93] [TIN94F] modelando cada red como si fuera un procesador, y cada mensaje como si fuera una tarea.

Para realizar el análisis sobre este tipo de sistemas de tiempo real es necesario definir un modelo con el que se pueda representar a cada sistema de una manera exacta a la vez que sencilla. El modelo debe reflejar tanto las características propias de la arquitectura (hardware y software) del sistema distribuido, como los requerimientos de tiempo real estricto que se le hayan impuesto.

En este capítulo vamos a definir en primer lugar un *modelo lineal* para la descripción de sistemas distribuidos de tiempo real estricto gobernados por eventos sobre el que se puedan aplicar las técnicas de análisis existentes. Este *modelo lineal* es el único que se aborda en la bibliografía, y aunque sea representativo de un gran número de este tipo de sistemas de tiempo real, es ciertamente limitado y será extendido en el capítulo 4 a un *modelo general* con mayor representatividad en los sistemas distribuidos de tiempo real estricto (esta extensión constituye una de las contribuciones importantes de este trabajo). Después, describiremos la aplicación de las actuales técnicas de análisis basadas en *RMA* al *modelo lineal* definido, y finalmente, trataremos la problemática que se plantea a partir del análisis para estos sistemas de tiempo real y cuya solución será tratada en capítulos posteriores.

2.2. Modelo lineal de sistema distribuido

En un sistema distribuido hay múltiples recursos procesadores (CPU's) y uno o más recursos de comunicación (buses, redes LAN, líneas serie, etc). El problema de la planificación y del análisis de la respuesta temporal de programas concurrentes ejecutando en los recursos procesadores está bastante bien definido. Vamos a asumir una planificación expulsora basada en prioridades fijas, para poder utilizar la teoría *RMA* en el análisis del comportamiento temporal de peor caso.

La planificación y el análisis de la respuesta temporal de peor caso en los recursos de comunicación también está bien establecida. La mayoría de las redes de comunicación estándares no se adaptan bien a las comunicaciones de tiempo real, y no soportan una

planificación de los mensajes basada en prioridades. Sin embargo, existen algunas redes de comunicación estándares que se han utilizado para llevar a cabo comunicaciones de tiempo real estricto, tales como el token-bus [STR88], FDDI [AGR91], el bus CAN [TIN94E], etc. También se han desarrollado algunas redes no estándares para hacer una planificación expulsora basada en prioridades. En esta misma Tesis (ver capítulo 5) presentamos una herramienta software basada en la interfase de colas de mensajes definida en el POSIX.1b [POS93] que permite el desarrollo de una comunicación con planificación mediante prioridades sobre subsistemas de comunicaciones estándares como el bus VME, y las líneas serie punto a punto (RS-232 y RS-485). Esta herramienta permite el intercambio de mensajes entre tareas de manera transparente a los procesadores en los que se encuentren éstas (las tareas se comunican a través de colas de mensajes sin importar dónde estén localizadas). El análisis del tráfico de mensajes en estos recursos de comunicación se realiza utilizando las mismas técnicas *RMA* que para los recursos procesadores. Por lo tanto, trataremos a los mensajes en los recursos de comunicaciones de la misma manera que a las tareas en los recursos procesadores, excepto por un pequeño término de bloqueo que se debe tener en cuenta porque los mensajes están compuestos por secuencias de paquetes, cada uno de los cuales es indivisible y no expulsable.

Definimos un modelo de sistema distribuido gobernado por eventos en el que tenemos una serie de *eventos externos* (generados por dispositivos externos, temporizadores, etc), a los que responden un conjunto de *tareas* (distribuidas por los distintos procesadores) y de *mensajes* (distribuidos por las redes de comunicación), que se activan entre sí mediante la generación de *eventos internos*. Estos eventos internos los pueden producir tanto las tareas (para activar a otras tareas del propio procesador o a algunos mensajes de las redes de comunicación), como los mensajes (para activar a las tareas a las que van dirigidos). Suponemos que todas las secuencias de eventos externos que llegan al sistema se conocen por adelantado, y que también se conocen los ritmos con los que llegan estos eventos con requerimientos de tiempo real estricto. En otro caso, sería imposible garantizar la planificabilidad del sistema a priori, es decir, en tiempo de compilación. Los eventos internos se generan en las respuestas a los eventos externos, y por lo tanto, sus características se derivan de las de éstos. Suponemos además que los eventos son instantáneos por lo que no tienen influencia alguna en el análisis del sistema (cálculo de las respuestas de peor caso de las acciones a las que activan). Esto no supone pérdida de generalidad porque los efectos de la

sobrecarga (*overhead*) se pueden incluir fácilmente en el análisis. También suponemos que las tareas se asignan estáticamente a los procesadores, y de igual manera los mensajes a las redes de comunicación. En muchos sistemas de tiempo real estricto las tareas están ligadas a procesadores específicos por la presencia de dispositivos hardware especiales que son necesarios para éstas.

Cada evento externo que llega al sistema genera una *respuesta* en forma de una *secuencia de acciones* que se activan entre sí mediante eventos internos. Cada una de estas acciones puede ser una tarea (o una parte de una tarea) ejecutada en un procesador, o un mensaje enviado a través de una red de comunicaciones. Normalmente, la primera acción de una respuesta es una tarea ejecutando en algún procesador determinado. Llamamos e_i al evento externo que activa a la acción a_i, y e_{jk} al evento interno generado por la acción a_j y que activa a la acción a_k. En el *modelo lineal* de sistema distribuido que estamos definiendo, las acciones solamente pueden ser activadas por un único evento (interno o externo), y solamente puede generar un evento interno activando por tanto a una única acción en el mismo o en diferente *recurso* (procesador o red).

Consideramos los siguientes requisitos temporales impuestos a las acciones en su respuesta a los eventos externos:

Plazos globales. Son plazos que deben cumplir determinadas acciones de la secuencia de respuesta relativos a la llegada del evento externo.

Plazos locales. Son plazos relativos a la activación de cada acción en la respuesta.

Plazos de principio-a-fin. Son los plazos globales correspondientes a la última acción en la secuencia de respuesta a un evento externo.

Llamaremos D_{ij} al plazo global de la acción a_j relativo a la llegada del evento e_i, d_{ij} al plazo local de la acción a_j para el evento e_i (relativo a la activación de la acción a_j), y ED_{ij} al plazo de principio-a-fin para la acción a_j (última en la secuencia de respuesta) relativo al evento e_i. La información correspondiente al evento que aparece en la nomenclatura de los

plazos (*i*) no es realmente necesaria, porque en este modelo cada acción sólo responde a un evento externo, pero la mantenemos para permitir que la nomenclatura sea transparente a la extensión del modelo que se realizará en el capítulo 4.

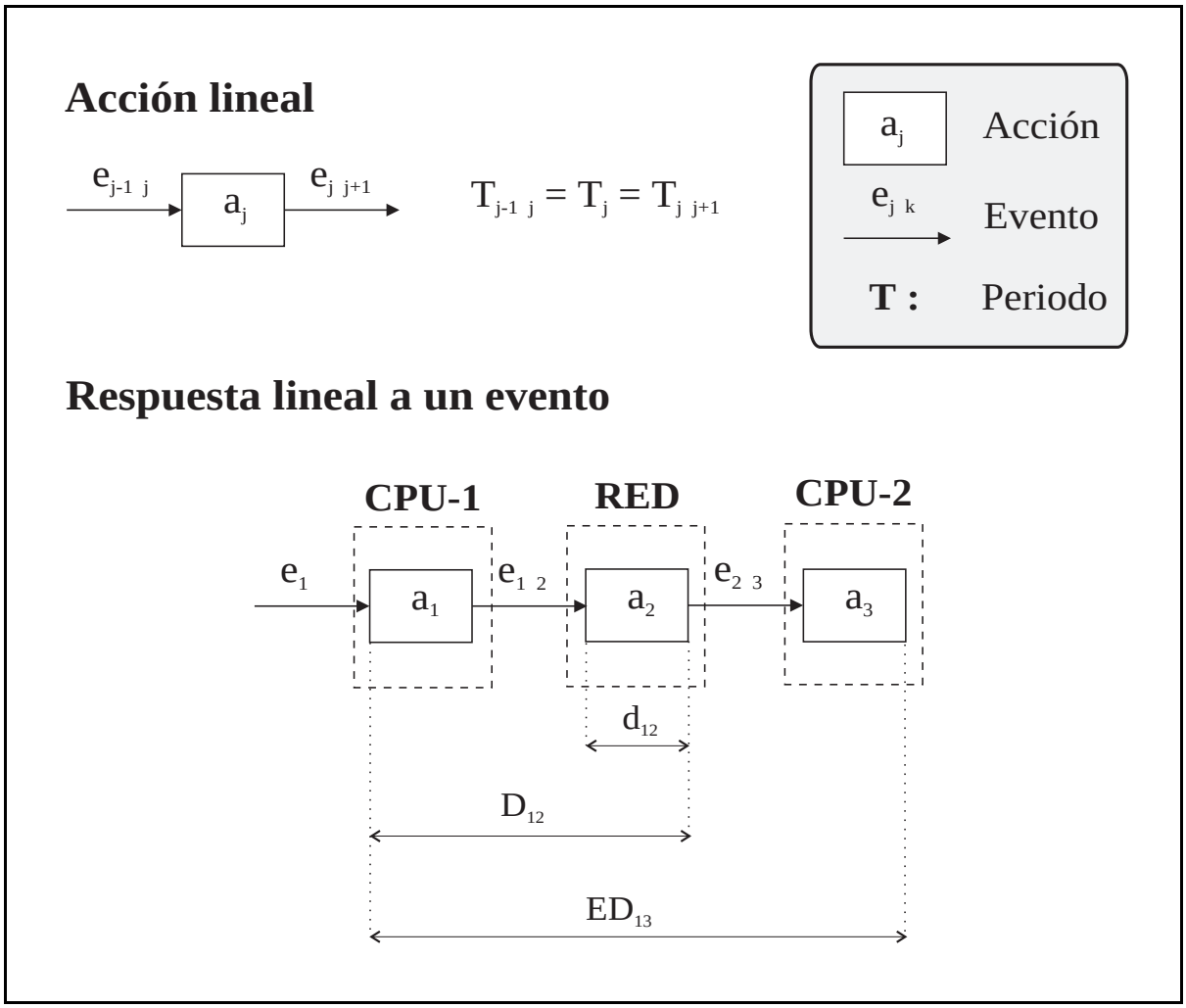

Figura 2-1. Modelo lineal de sistema distribuido gobernado por eventos.

De acuerdo con esta definición del *modelo lineal* de sistema distribuido, la Figura 2-1 muestra el patrón propio de la *acción lineal* característica de las secuencias de respuesta a los eventos en este modelo. El caso de acción lineal se corresponde con el modelo utilizado en la referencia [GUT95A], en el que una acción es disparada por un determinado evento de entrada. El período del evento es heredado por la acción a la que dispara y también por el evento de salida que puede generar ésta. En la Figura 2-1 también se muestra un ejemplo de respuesta a un evento externo en cuyas acciones se pueden apreciar los diferentes tipos de

plazos que hemos definido. La última acción en la respuesta a un evento (generalmente una tarea) es un caso particular de la acción lineal en el que no se genera el evento de salida.

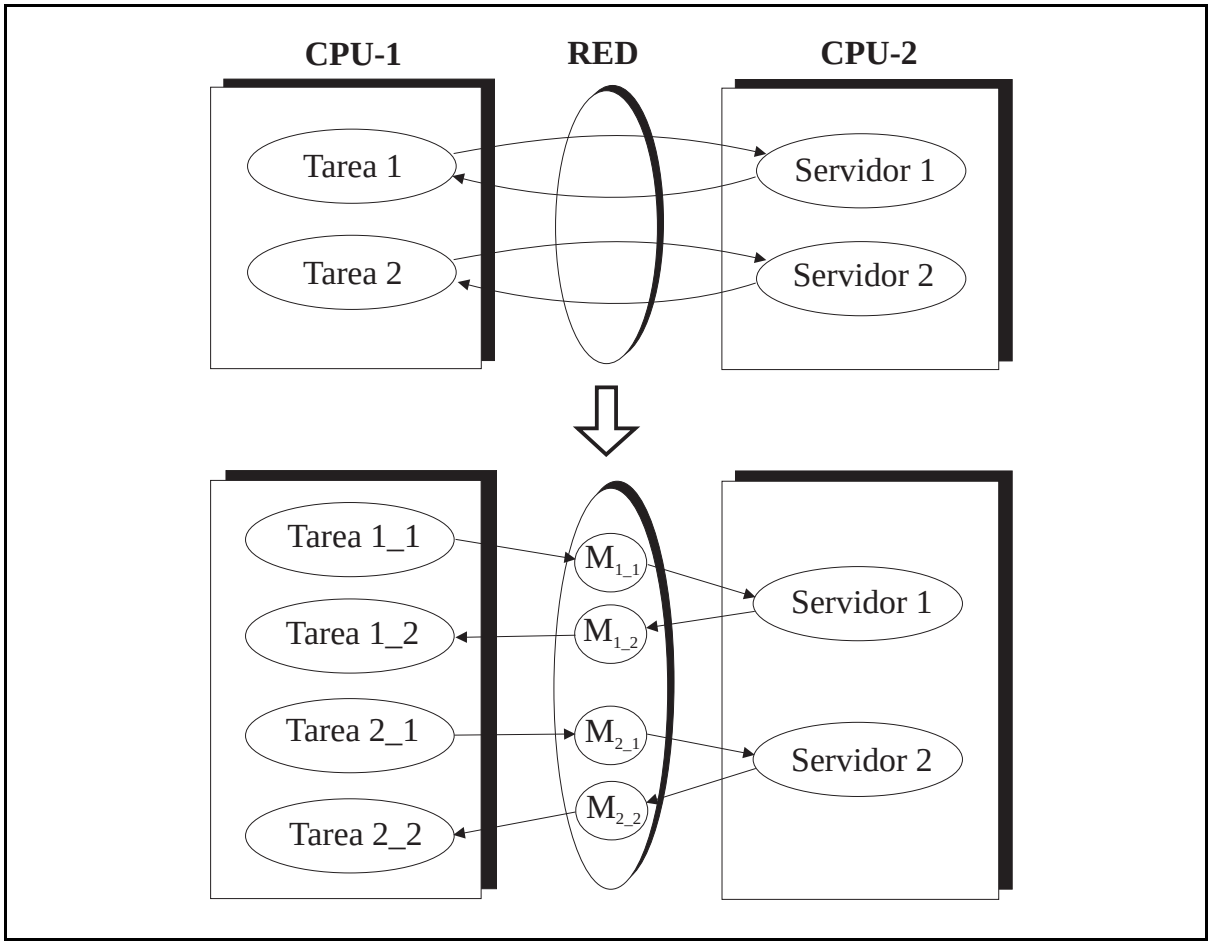

Figura 2-2. Modelado de tareas que invocan servidores remotos.

Con este *modelo lineal* es posible representar un buen número de arquitecturas de tiempo real aunque con algunas limitaciones, ya que no permite la sincronización entre tareas ni la activación por combinaciones de eventos. Un ejemplo que responde al *modelo lineal* lo podemos encontrar en un sistema que utiliza una aproximación cliente-servidor con servidores remotos. En este sistema, cada parte de una tarea que solicita un servicio de un servidor remoto se puede descomponer en la siguiente secuencia de acciones: la acción anterior a la solicitud del servicio, el mensaje enviado al servidor, la acción del servidor, la respuesta del servidor, y la acción posterior a la solicitud del servicio. La Figura 2-2 muestra esta descomposición para un sistema distribuido con dos nudos y una red de comunicaciones.

Sin embargo, podemos observar que no es posible representar con el modelo lineal la situación que se produce cuando dos o más tareas solicitan servicios al mismo servidor. Esta situación se podrá representar en cambio, con el *modelo general* que desarrollaremos en el capítulo 4.

2.3. Análisis de tiempo real

El análisis general de peor caso para un sistema distribuido continúa siendo un camino abierto, pero una buena aproximación es aquella que considera cada recurso procesador y cada recurso de comunicaciones del sistema como si fueran independientes. Esta aproximación es pesimista en el sentido de que el peor caso de verdad debe tener en cuenta el hecho de que las fases (o relaciones temporales entre acciones de diferentes respuestas) son interdependientes. Si consideramos las fases de cada acción como si fueran independientes, podemos construir una situación que es peor que el peor caso. Sin embargo, aunque ligeramente pesimista, este método nos permite garantizar la planificabilidad de los sistemas de tiempo real.

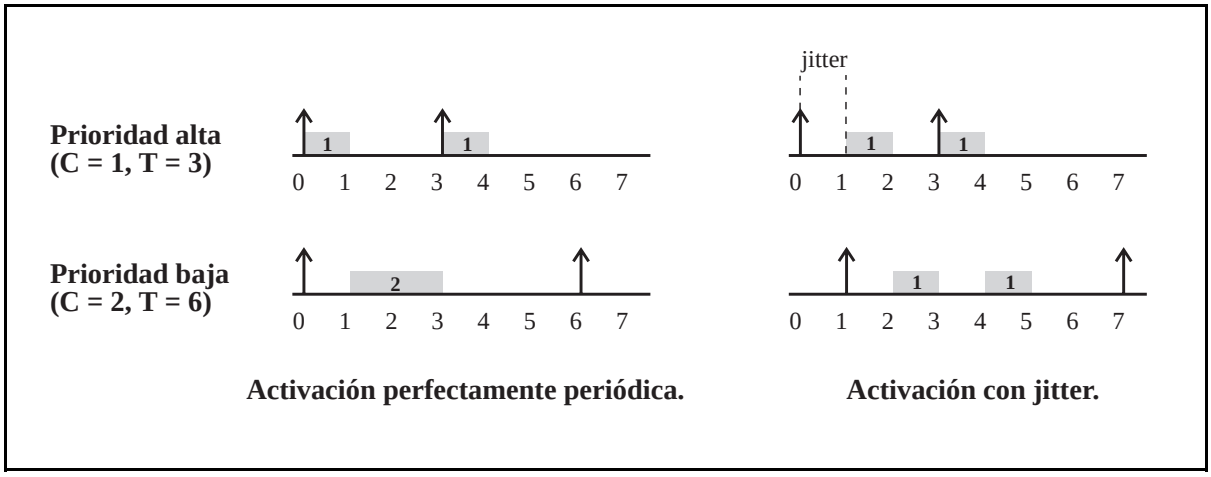

Figura 2-3. Efectos del jitter sobre una acción de baja prioridad.

El análisis de un sistema distribuido como el descrito en el modelo del apartado anterior se puede llevar a cabo mediante las técnicas *RMA* introducidas en el capítulo 1, pero teniendo en cuenta el efecto del *jitter* [TIN94F]. La Figura 2-3 muestra un ejemplo sencillo en el que una acción de prioridad baja únicamente puede ser expulsada una vez por otra acción con mayor prioridad, para el caso en el que ésta se active de manera perfectamente periódica. Sin embargo, esta acción de prioridad baja puede llegar a ser expulsada dos veces

si la activación de la de prioridad alta se retrasa, es decir, sufre el efecto del *jitter* (el instante de activación no coincide con el comienzo de la ejecución).

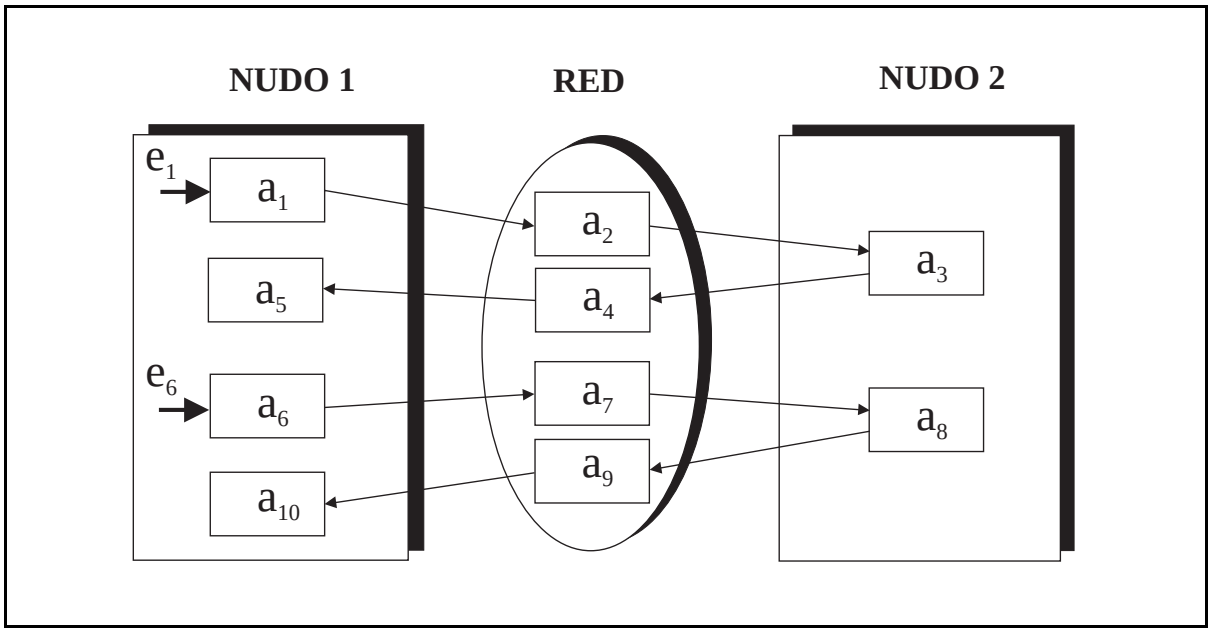

Figura 2-4. Ejemplo de sistema distribuido.

En los sistemas multiprocesadores y distribuidos los mensajes se activan a partir de la ejecución de las tareas, y sus instantes de activación no son perfectamente periódicos porque dependen de los instantes en los que finalizan las tareas que los disparan (que son variables). Exactamente lo mismo ocurre con las tareas al ser activadas por la llegada de un mensaje. Así por ejemplo, en la Figura 2-4 se muestra un sistema distribuido sencillo (que puede responder a la estructura del modelo de tareas que invocan a servidores remotos de la Figura 2-2 en la página 2-6) en el que se puede apreciar que solamente las acciones a_1 y a_6, que responden a los eventos periódicos e_1 y e_6 respectivamente, pueden activarse periódicamente. Las demás acciones dependen de los tiempos de finalización de ejecución de las acciones anteriores en las cadenas de respuesta a los eventos externos. El problema que surge al abordar el análisis de este tipo de sistemas es que el cálculo del *jitter* para las acciones de un recurso depende de los tiempos de respuesta en los demás recursos, y el cálculo de los tiempos de respuesta depende a su vez del *jitter*.

La solución a este problema la suministran Tindell y Clark [TIN94F] y consiste en repetir el análisis de peor caso de forma iterativa (cálculo de respuestas de peor caso y *jitter* para todos los recursos del sistema). Las características de *monotonicidad* del tiempo de

respuesta con el *jitter* hacen que el análisis converja a una solución correcta. Como resultado del análisis se obtienen las respuestas globales de peor caso y a partir de ellas, se puede comprobar que se verifican las condiciones de planificabilidad impuestas al sistema distribuido. El sistema es planificable cuando se verifican los plazos globales asignados a las acciones, o sea, cuando las respuestas globales de peor caso de las acciones son menores o iguales que sus plazos globales.

De igual modo que en la definición de los plazos (ver página 2-4), vamos a utilizar la información correspondiente al evento en la definición de algunos términos (aunque no sea realmente necesaria), ya que de esta manera se consigue que la nomenclatura sea transparente a la extensión del modelo que se realizará en el capítulo 4. Definimos el tiempo de respuesta local de peor caso de una acción como el tiempo transcurrido desde su activación hasta su finalización en el peor caso (la manera de calcularlo se detalla más adelante). Además, definimos el tiempo de respuesta global de peor caso R_{ij} (de la acción a_j relativo a la llegada del evento e_i) como la suma de los tiempos de respuesta locales de la propia acción y de las acciones previas en la secuencia de respuesta al evento externo e_i:

$$R_{ij} = \sum_{k \in A_{ij}} r_{ik}$$

donde,

r_{ik} Es el tiempo de respuesta local de peor caso de la acción a_k (relativo a la activación de a_k) para el evento e_i.

A_{ij} Es el conjunto de acciones de la secuencia de respuesta al evento externo e_i previas a la acción a_j (incluida ella misma).

Para el cálculo de las respuestas locales de peor caso aplicamos la teoría *RMA* a cada recurso por separado de la forma vista en el capítulo 1 (ver apartado 1.3.5). Por lo tanto, el tiempo de respuesta local de peor caso para la acción a_j en la respuesta al evento e_i se calcula a partir de la siguiente ecuación utilizada de manera iterativa [LEH90] [TIN93D]:

$$w_{ij}^{n+1}(q) = (q+1)\ C_j + B_j + \sum_{\forall\, k \in hp(j)} \left\lceil \frac{J_{ik} + w_{ij}^{n}(q)}{T_k} \right\rceil C_k$$

donde,

$w_{ij}(q)$ Es el tiempo de finalización de peor caso de la acción a_j (para el evento externo e_i) al que converge la ecuación cuando $w_{ij}^{n+1}(q) = w_{ij}^{n}(q)$, para el caso de la *q-ésima* activación después de un instante crítico. El valor de arranque es $w_{ij}^{0}(q) = (q+1)\ C_j$.

$hp(j)$ Es el conjunto de acciones que pueden expulsar a la acción a_j.

C_j Es el tiempo de ejecución de peor caso de la acción a_j.

C_k Es el tiempo de ejecución de peor caso de la acción a_k.

T_k Es el período la acción a_k.

B_j Es el término de bloqueo correspondiente a la acción a_j (mayor sección crítica de un semáforo cuyo techo de prioridad es mayor o igual que la prioridad de la acción a_j).

J_{ik} Es el *jitter* de peor caso para el evento externo e_i y la acción a_k. Este *jitter* es igual a la variación del tiempo de respuesta de la acción previa. Asumimos que esta variación es igual al tiempo de respuesta global de peor caso de la acción previa (R_{ik-1}) porque en general el tiempo de respuesta de mejor caso podría ser arbitrariamente pequeño. En el caso de que se conociese este tiempo de respuesta de mejor caso, o una cota inferior de este tiempo, el *jitter* se calcularía de la forma siguiente:

$$J_{ik} = R_{ik-1} - \sum_{l \in A_{ik-1}} b_{il}$$

donde b_{il} es el tiempo de respuesta de mejor caso de la acción a_l para el evento e_i, y A_{ik-1} es el conjunto de acciones de la secuencia de respuesta al evento e_i previas a la acción a_{k-1} (incluida ella misma). El *jitter* de la primera acción en la respuesta a un evento externo es cero.

Esta ecuación se aplica de forma sucesiva para valores de q iguales a 0,1,2,…, y termina cuando se verifica la desigualdad siguiente:

$$w_{ij}(q) \leq (q+1)\ T_j$$

A partir de los resultados obtenidos para los distintos valores de q se puede calcular la respuesta local de peor caso de la acción a_j al evento externo e_i de la manera siguiente:

$$r_{ij} = \max_{q=0,1,2,...} (w_{ij}(q) - qT_j)$$

De esta forma se pueden obtener los distintos tiempos de respuesta locales para cada acción (a_j) y para cada evento externo (e_i) de cuya respuesta forma parte la acción (a_j), siempre para valores del *jitter* ya determinados. Mediante un análisis iterativo se van calculando de manera sucesiva los tiempos de respuesta y los nuevos términos de *jitter* asociados (inicialmente todos los términos de *jitter* valen cero). La solución correcta es aquella que se obtiene cuando dos iteraciones sucesivas llegan a la misma solución.

Por último, diremos que para determinar la planificabilidad del sistema, se deben comparar los tiempos de respuesta globales (R_{ij}) con los plazos globales (D_{ij}) de las acciones que los tengan. Vamos a llamar ER_{ij} al tiempo de respuesta global de peor caso de la acción a_j para el evento e_i cuando es la última acción de la respuesta.

2.4. Problemática derivada del análisis

En sistemas monoprocesadores el primer problema que se plantea una vez que se han definido las técnicas de análisis, es el de la asignación de prioridades a las tareas. Tanto el problema de asignación de prioridades (de forma estática), como el de la optimización de esta asignación están bien resueltos para este tipo de sistemas de tiempo real. Liu y Layland [LIU73] demostraron que la asignación de prioridades *RMA* era óptima para un conjunto de tareas periódicas independientes con plazos iguales al período de cada tarea. Más tarde, Leung y Whitehead [LEU82] probaron que la asignación de prioridades óptima para plazos menores o iguales que el período es la que se realiza de acuerdo con estos plazos (*deadline monotonic*). Por último, Audsley [AUD91B] proporcionó un algoritmo que era capaz de encontrar una solución (si existía) al problema de la asignación de prioridades cuando uno o más plazos eran superiores a los períodos de las tareas.

Sin embargo, en sistemas multiprocesadores y distribuidos de tiempo real estricto los problemas que surgen a raíz de la definición de las técnicas de análisis son mucho más complicados, debido fundamentalmente a la fuerte interacción entre los tiempos de respuesta de acciones localizadas en recursos (procesadores o de comunicaciones) diferentes. En estos sistemas al problema de la asignación de prioridades a las acciones (suficientemente complejo por sí mismo), se le puede añadir el de asignación de estas acciones a los recursos correspondientes. Tindell, Burns y Wellings [TIN92] trataron el problema de la asignación de

prioridades y de la localización de tareas de manera combinada, aplicando como técnica de optimización el *templado simulado*. Obviamente el problema de asignación de acciones (tareas y mensajes) a recursos (procesadores y redes de comunicación) lleva implícito de alguna manera el de la asignación de prioridades.

Así pues, partiendo del *modelo lineal* definido en el apartado 2.2 y aplicando las técnicas de análisis descritas en el apartado 2.3, vamos a afrontar la optimización de las actuales técnicas de asignación de prioridades para este tipo de sistemas de tiempo real estricto. El *templado simulado* es una técnica de optimización de propósito general, y su aplicación a la asignación de prioridades en sistemas distribuidos da lugar a un algoritmo lento (en lo que se refiere al tiempo que tarda en alcanzar la solución) y del que no conocemos la bondad de las soluciones obtenidas. Como objetivo intentaremos encontrar un método heurístico para la asignación de prioridades basado en nuestro conocimiento de los tiempos de respuesta del sistema, y que comparativamente sea más rápido y obtenga mejores soluciones que el *templado simulado*. También intentaremos optimizar la aplicación del *templado simulado* a la asignación de prioridades en el sentido de alcanzar mejores soluciones después de haber alcanzado la primera (la actual aplicación [TIN92] no es capaz de distinguir entre las asignaciones que hacen que el sistema sea planificable). Consideramos que las acciones están asignadas de forma estática a los recursos, y que los mecanismos de planificación que se utilizan son de expulsión por prioridades fijas (de acuerdo con el análisis que se va a aplicar). Todo esto se tratará en el capítulo 3.

En esta misma línea podemos pensar en sistemas distribuidos de tiempo real estricto más complejos, en los que se pueda producir la sincronización entre tareas e incluso en los que las acciones se puedan disparar por combinaciones de eventos; por ejemplo, un sistema que utiliza una aproximación cliente-servidor como el de la Figura 2-2 (ver página 2-6), pero en el que haya un servidor para múltiples clientes. Por lo tanto, en primer lugar es necesario definir un modelo capaz de representar este tipo de sistemas al que vamos a llamar *modelo general* y que se puede ver como una extensión del *modelo lineal*. Más tarde, será necesario definir el análisis de tiempo real aplicable a este *modelo general* en el sentido de calcular los tiempos de respuesta de las acciones en el peor caso (tomaremos como base el análisis para el *modelo lineal*). Finalmente, extenderemos a este modelo general las técnicas de asignación de prioridades. Estos puntos serán tratados en el capítulo 4.

En principio, como vimos anteriormente, en los sistemas distribuidos el efecto del *jitter* influye de manera negativa en la planificabilidad (aumentan los tiempos de respuesta de peor caso y consiguientemente disminuyen las posibilidades de utilización de los recursos). La problemática que planteamos en torno a este efecto negativo en los sistemas distribuidos se puede resumir en los siguientes puntos:

Estudio de la influencia real del jitter. Realización de las medidas oportunas que permitan estimar el efecto negativo del *jitter* sobre la planificabilidad de los sistemas distribuidos de tiempo real estricto.

Estudio de técnicas que permitan eliminar el jitter. Concepción de los planificadores adecuados que permitan asegurar la eliminación del *jitter* tanto en los procesadores como en las redes de comunicación.

Estudio de las ventajas de la eliminación del jitter. Estimación de la sobrecarga introducida en el sistema por el uso de planificadores que permiten eliminar el *jitter*, para su comparación con el aumento en la planificabilidad conseguido por los mismos.

En el capítulo 6 veremos cómo efectivamente el efecto del *jitter* en este tipo de sistemas es enorme, y veremos además cómo eliminarlo mediante el uso del *servidor esporádico*. La estimación de la sobrecarga producida en las redes de comunicación por un planificador que utiliza *servidor esporádico* se realizará en el capítulo 7. Aunque la mejor solución para un sistema distribuido es la eliminación del *jitter*, desde el punto de vista de la asignación de prioridades se debe aplicar un análisis que tenga en cuenta este efecto para cubrir todos los casos, incluidos aquellos en los que no sea posible utilizar un planificador que lo elimine.

3. Optimización de la asignación de prioridades en sistemas distribuidos

3.1. Introducción

Tomando como referencia el modelo definido en el capítulo 2, un sistema distribuido de tiempo real estricto se compone de un conjunto de tareas localizadas en nudos procesadores que se comunican entre sí mediante mensajes enviados por las redes de comunicación. Tindell, Burns y Wellings [TIN92] consideran tres dificultades fundamentales en la construcción de este tipo de sistemas que son: la localización de tareas (asignación de tareas a procesadores), la planificación de los procesadores (asignación de prioridades a las tareas) y la planificación de las redes (asignación de prioridades a los mensajes). En general hay P^T maneras distintas de asignar T tareas en P procesadores, y el problema de encontrar una posible solución óptima de esta asignación (en la que todas las tareas cumplan los requerimientos físicos y temporales que se les han sido impuestos) pertenece al tipo de problemas conocido con el nombre de *NP-completo* (Nondeterministic Polynomial time complete). Las tres dificultades a las que hemos aludido responden a las características de los problemas *NP-completos*, para los que obtener una solución exacta requeriría una capacidad computacional prácticamente ilimitada. A estas dificultades se les podría añadir una más que es la localización de mensajes, es decir, la asignación de mensajes a las redes de comunicación (cuando haya varias redes conectando los mismos procesadores).

Tindell, Burns y Wellings [TIN92] aplicaron la técnica del *templado simulado* para resolver conjuntamente los problemas de localización de tareas, y de planificación de procesadores y redes de comunicación. Utilizaron una arquitectura distribuida muy específica constituida por un conjunto de procesadores que se comunicaban a través de un bus mediante un protocolo de paso de testigo. El *templado simulado* [KIR83] es una técnica de optimización de propósito general basada en la relación existente entre la mecánica estadística (comportamiento de sistemas con múltiples grados de libertad en equilibrio térmico a una

temperatura finita) y la optimización combinatoria o multivariable (encontrar el mínimo de una función dada que depende de muchos parámetros).

En este capítulo, nos vamos a centrar en la resolución de los problemas de la planificación, tanto de los procesadores como de las redes de comunicación que los interconectan para sistemas distribuidos que respondan al *modelo lineal* definido en el capítulo 2. Como vimos en el capítulo 1, las políticas de planificación expulsoras basadas en prioridades fijas constituyen una buena elección para los sistemas de tiempo real, entre otras razones, porque nos permiten realizar un análisis de peor caso que garantiza el cumplimiento de los requerimientos temporales del sistema. Así pues, tenemos un conjunto de tareas distribuidas en un conjunto de procesadores que se envían mensajes por las diferentes redes de interconexión, y tratamos de asignar prioridades a las tareas y a los mensajes, de manera que se verifiquen los requerimientos de tiempo real impuestos.

Para resolver este problema de asignación de prioridades a tareas y a mensajes, Tindell, Burns y Wellings [TIN92] aplicaron la técnica del *templado simulado* a la resolución del problema de asignación de prioridades a tareas y a mensajes (junto con la localización de las tareas en los procesadores). Esta es una técnica de optimización de propósito general, lenta en la obtención de soluciones, y de la que no conocemos la bondad de dichas soluciones. Nosotros vamos a desarrollar un nuevo algoritmo de planificación heurístico *HOPA* (Heuristic Optimized Priority Assignment) [GUT95A] que resuelve el problema de la asignación de prioridades basándose en el conocimiento de los tiempos de respuesta del sistema, y cuyos resultados los podremos comparar con los obtenidos por el *templado simulado*. También aplicaremos la técnica del *templado simulado* en la misma línea que en la referencia [TIN92] (extrayendo la componente correspondiente a la planificación de los procesadores), pero realizando una ligera modificación que nos va a permitir optimizar las soluciones obtenidas.

Finalmente, con motivo de poder evaluar y comparar los dos algoritmos de asignación de prioridades mencionados, desarrollaremos una herramienta en lenguaje Ada en la que se van a implementar ambos. También vamos a realizar una herramienta de generación automática de ejemplos (igualmente en lenguaje Ada) que nos va a facilitar la evaluación de dichos algoritmos. Veremos cómo el algoritmo heurístico encuentra en general más y mejores soluciones que el *templado simulado*, y además de una forma mucho más rápida (el tiempo

de CPU consumido en la ejecución de nuestro algoritmo heurístico es en promedio dos órdenes de magnitud menor que el empleado por el *templado simulado*).

3.2. Planificación mediante *templado simulado*

En este apartado vamos a describir la aplicación de la técnica denominada *templado simulado* a la asignación de prioridades a las tareas y a los mensajes en sistemas distribuidos de tiempo real estricto (que respondan al *modelo lineal* definido en el capítulo 2), tomando como referencia el trabajo desarrollado por Tindell, Burns y Wellings [TIN92].

El *templado simulado* es una técnica global de optimización que intenta encontrar el punto más bajo dentro de un paisaje de energías. Esta técnica se deriva de las observaciones efectuadas sobre el enfriamiento del metal fundido que posteriormente da como resultado una estructura cristalina regular. El *templado simulado* no es un algoritmo heurístico, ya que únicamente se fundamenta en el conocimiento de las características que debe tener una buena solución, y no en el modo en que se puede obtener ésta. El rasgo característico de este algoritmo es que incorpora saltos aleatorios hacia potenciales nuevas soluciones (puntos de menor energía), pero permite saltos hacia atrás (punto de mayor energía). Esta característica lo distingue de los algoritmos *golosos* que siempre tienden hacia potenciales nuevas soluciones.

Antes de comenzar con descripción del algoritmo propiamente dicho, vamos a dar algunas definiciones necesarias para su aplicación a la planificación de recursos (procesadores y de comunicación) en el tipo de sistemas con los que estamos trabajando. Estas definiciones son las que se exponen a continuación:

Espacio problema. Es el conjunto de posibles asignaciones de prioridades a las tareas y a los mensajes en el sistema distribuido.

Punto del espacio problema. Es cada una de las posibles asignaciones de prioridades a las tareas y a los mensajes que constituyen el espacio problema.

Espacio vecino. Para cada punto del espacio problema, es el conjunto de puntos que son alcanzables con la aplicación de la *función vecino* (transforma cualquier punto en un punto vecino). Así pues, describiremos el espacio vecino a partir de la definición de la *función vecino* que opera siguiendo los pasos que se detallan a continuación:

1.- Selecciona un recurso (procesador o red) de manera aleatoria.
2.- Selecciona dos acciones (tareas o mensajes) al azar del recurso elegido.
3.- Intercambia las prioridades de las acciones seleccionadas.

Energía. Es la medida de la bondad de la asignación de prioridades correspondiente a cada punto del espacio problema (una mala asignación se corresponde con un punto de alta energía). Esta bondad de la asignación de prioridades tiene una relación directa con la planificabilidad del sistema (cumplimiento de los requerimientos temporales), es decir, la energía de cada punto deberá de poder determinar si el sistema es planificable o no (con la asignación de prioridades correspondiente), y en cierto modo deberá de dar una idea de la distancia hasta la planificabilidad (si no es planificable) o hasta la no planificabilidad (si es planificable). La energía de un punto se calculará mediante la *función energía* que definiremos más adelante. Esta función configura la forma del espacio problema, que se puede ver como un paisaje rugoso con profundos valles que representan buenas soluciones, y altos picos que representan soluciones malas o no viables (la solución hace que el sistema no sea planificable).

Teniendo en cuenta estas definiciones, el problema de la asignación de prioridades se reduce a encontrar el punto de menor energía dentro del espacio problema. Normalmente no se encuentra el mínimo absoluto.

El modo de operación del algoritmo consiste en dar saltos desde un punto cualquiera hacia un punto vecino siguiendo el criterio que describimos a continuación. Elegimos un punto de comienzo aleatorio (p_c) y evaluamos su energía (E_c). Después elegimos un punto del espacio vecino (p_v) y también evaluamos su energía (E_v). El punto vecino se convertirá en el nuevo punto de comienzo si se verifica una de las siguientes desigualdades:

1.- $E_v < E_c$ (salto hacia adelante)

2.- $e^x \geq random\ (0,1)$ (salto hacia atrás)

con

$$x = \frac{E_c - E_v}{K}$$

y donde,

K Es una variable aleatoria de control que actúa de manera similar al factor de temperatura en un sistema termodinámico.

random (*a*,*b*) Es un generador uniforme de números aleatorios entre *a* y *b*.

La variable de control *K* se va reduciendo progresivamente durante la ejecución del algoritmo (enfriamiento del sistema) para hacer menos probables los saltos hacia energías mayores. Eventualmente, el sistema se *congela* en un estado de baja energía. La estructura del algoritmo es la que se muestra a continuación:

```
algoritmo templado simulado is
begin
      Elige un punto aleatorio inicial p_inicial;
      Elige una temperatura inicial K_inicial;
      p_actual := p_inicial;
      K_actual := K_inicial;
      loop
            loop
                  E_actual := energía del punto p_actual;
                  Elige p_vecino, un vecino de p_actual;
                  E_vecino := energía del punto p_vecino;
                  if E_vecino < E_actual then
                        p_actual := p_vecino;
                  else
                              E_actual - E_vecino
                        x = ------------------------
                                    K_actual
                        if e^x ≥ random (0,1) then
                              p_actual := p_vecino;
                        end if;
                  end if;
                  exit when equilibrio térmico;
            end loop;
            K_actual := F(K_actual); -- Enfriamiento
            exit when algún criterio de parada;
      end loop;
end templado simulado;
```

Como se puede observar el algoritmo requiere tres funciones para su operación que son las siguientes:

Función energía. Es la función principal en la asignación de prioridades, ya que da lugar al paisaje de energías sobre el cual va a tratar de alcanzar la solución el algoritmo. La componente de planificación utilizada por Tindell, Burns y Wellings [TIN92] en su definición de la función energía venía dada por la suma de las cantidades que los tiempos de respuesta sobrepasaban los plazos, para las tareas que no los cumplían. Esta definición tiene el inconveniente de que no es capaz de discriminar entre asignaciones de prioridades que hacen al sistema planificable, y por lo tanto, no permite la optimización de soluciones (todas las asignaciones de prioridades que hacen al sistema planificable tienen una energía igual a cero). De esta manera, definimos una nueva función energía que va a ser igual al opuesto del *índice de planificación* (*Ind*, definido en este trabajo). Este índice nos permite determinar la bondad de una solución para que se pueda comparar con otras (tanto si es planificable como si no lo es), y se define en función de la peor distancia entre los tiempos de respuesta de peor caso y los plazos globales para cada acción. Se puede expresar de la forma en que se muestra a continuación:

$$Ind = \begin{cases} \sum_{j \in G} \min\ (0, D_{ij} - R_{ij}) & ;\ \textit{cuando no es planificable} \\ \sum_{j \in G} \max\ (0, D_{ij} - R_{ij}) & ;\ \textit{cuando es planificable} \end{cases}$$

donde,

D_{ij} Es el plazo global de la acción a_j para el evento e_i.

R_{ij} Es el tiempo de respuesta global de peor caso de la acción a_j para el evento e_i.

G Es el conjunto de acciones que tienen requerimientos globales de tiempo (plazos globales).

Los valores negativos del índice de planificación representan soluciones que hacen que el sistema no sea planificable, y valores positivos o cero representan

soluciones planificables. Cuanto mayores sean los índices, mejores son las soluciones para el sistema en términos de distancia entre tiempos de respuesta y plazos.

Función vecino. Esta función ya la hemos descrito a través de su modo de operación (ver definición del espacio vecino en página 3-4).

Función enfriamiento F(K). El objetivo de esta función es disminuir progresivamente la temperatura del sistema a partir de una inicial ($K_{inicial}$) para permitir alcanzar puntos con energía cada vez más baja. Hay múltiples formas de construir esta función (aparecen referenciadas en [TIN92]) que abarcan un amplio rango en cuanto a su complejidad. Nosotros elegimos una de las más sencillas que consiste en multiplicar la temperatura por un número α que verifica $0 \leq \alpha \leq 1$.

Otro aspecto que podemos observar y que es necesario definir es el que se refiere a los criterios de parada de los dos lazos anidados que constituyen el algoritmo. Los intentos de salto a un punto vecino se pueden modelar mediante cadenas de Markov a partir de las que se pueden obtener estos criterios de parada, pero no existe una regla capaz de marcar una pauta clara para determinarlos. Los criterios que establecemos son los siguientes:

Equilibrio térmico. Kirkpatrick et al [KIR83] obtienen una condición de equilibrio térmico basada en el número de saltos hacia puntos de energía menor (se alcanza el equilibrio cuando este número sobrepasa un umbral). En esta misma línea, nosotros vamos a tomar como referencia el instante en el que se salta a un punto que tiene la energía mínima que se ha conseguido obtener; a partir de ese momento contamos el número de iteraciones que se realizan sin que se produzca una disminución en la energía, y consideramos que se alcanza el equilibrio térmico cuando se sobrepasa un número predeterminado de estas iteraciones (no se consigue rebajar la energía mínima obtenida).

Parada del algoritmo. Para establecer la condición de parada del algoritmo consideramos, al igual que para la condición de equilibrio térmico, el momento

en el que se salta al punto de energía mínima, y detenemos el algoritmo cuando se sobrepase un número determinado de iteraciones (mayor que el que marca la condición de equilibrio térmico) sin conseguir rebajar la energía mínima.

Tanto la evolución del algoritmo en ejecución, como la posibilidad de alcanzar soluciones van a depender de los parámetros de parada que determinan el número máximo de iteraciones permitidas sin saltar a un punto vecino (para el equilibrio térmico), y el número máximo de iteraciones permitidas sin rebajar la energía mínima conseguida (para la parada del algoritmo). También se puede tener en cuenta para establecer las condición de parada el hecho de que se haya encontrado una solución (asignación de prioridades que hace al sistema planificable), y detener el algoritmo sin permitir la optimización (si solamente interesa conseguir una solución con independencia de su bondad), o limitar el número de iteraciones después de encontrar una solución (para limitar el tiempo que tarda en obtener nuevas soluciones).

El *templado simulado* será teóricamente capaz de encontrar una solución al problema de la asignación de prioridades para cualquier sistema si es que ésta existe (puede llegar a explorar todas las asignaciones posibles). En la práctica, el inconveniente de este algoritmo radica en que para sistemas relativamente simples, el tiempo que puede emplear en encontrar una solución puede ser desmedidamente grande.

3.3. Algoritmo heurístico de planificación

En este apartado proponemos un nuevo algoritmo heurístico para resolver el problema de la asignación de prioridades en los sistemas distribuidos de tiempo real estricto que se puedan representar por el *modelo lineal* definido en el capítulo 2. Pretendemos desarrollar un algoritmo que resuelva este problema de una manera más rápida que el *templado simulado,* y que alcance además mejores soluciones que éste (evaluadas mediante el índice de planificación definido en la página 3-6). El algoritmo *HOPA* (Heuristic Optimized Priority Assignment) [GUT95A] se aprovecha del conocimiento de los factores que influyen en el comportamiento temporal del sistema para encontrar una solución optimizada al problema de

la asignación de prioridades (en oposición a la técnica de *templado simulado* que es de propósito general).

El algoritmo *HOPA* se basa en la distribución de los plazos globales de principio-a-fin de cada respuesta a un evento externo entre las diferentes acciones que componen esa respuesta. Una vez que se le ha asignado a cada acción su plazo local artificial, se asignan prioridades *deadline monotonic* en cada recurso de acuerdo con estos plazos locales, y se lleva acabo el análisis del sistema completo. Como resultado del análisis se calculan nuevos plazos intermedios. La iteración se ejecuta hasta que se encuentra una solución adecuada o se cumple algún criterio de parada. El algoritmo se puede expresar de la forma siguiente:

```
algoritmo HOPA is
begin
      Asigna plazos locales iniciales;
      loop
            Asigna prioridades de acuerdo con los plazos;
            Calcula los tiempos de respuesta de peor caso;
            exit when algún criterio de parada;
            Calcula nuevos plazos locales;
      end loop;
end HOPA;
```

En la descripción de las etapas del algoritmo vamos a suponer que en principio los requerimientos de tiempo real que se imponen son únicamente los plazos globales de principio-a-fin (plazos globales a las últimas acciones en las respuestas a los eventos) y comentaremos cómo se amplía para plazos globales en las acciones intermedias. Mantenemos además la información del evento en la nomenclatura por la misma razón que lo hicimos en la definición del modelo.

Los plazos locales iniciales se pueden asignar de alguna manera que preserven los plazos globales, es decir, que para todas las acciones a_k de la respuesta al evento e_i se tiene que cumplir:

$$ED_{ij} = \sum_k d_k$$

donde, d_k es el plazo local asignado a la acción a_k, y ED_{ij} es el plazo global de principio-a-fin de la acción a_j en la respuesta el evento e_i. Una posible asignación de estos plazos iniciales

consiste en distribuir los plazos globales de forma proporcional al tiempo de ejecución de peor caso de cada acción (C_j).

Una vez que se conocen los plazos locales para cada acción, se realiza la asignación de prioridades *deadline monotonic* de manera independiente en cada recurso de acuerdo con los plazos locales, es decir, asignando la mayor prioridad a la acción con el plazo local más pequeño (en cada recurso), y así sucesivamente.

Después de asignar todas las prioridades, se analiza el sistema usando la técnica de análisis descrita en [TIN92] que hemos examinado en el capítulo 2. El análisis proporciona los tiempos de respuesta de peor caso de cada acción para la asignación de prioridades dada. Si la solución es adecuada (o se alcanza alguna otras condición de parada), el algoritmo finaliza. En otro caso, se calculan los nuevos plazos locales de cada acción mediante la redistribución de los plazos globales entre las acciones que forman parte de la respuesta a un evento. Esta redistribución tiene en cuenta el *exceso* de cada acción, que intuitivamente mide la distancia que separa a cada acción de la planificabilidad. Los nuevos plazos intermedios se obtienen en función de dos factores: por un lado, el exceso de cada acción relativo al resto de las acciones que forman parte de su misma respuesta; por otro lado, el exceso de cada acción relativo al resto de las acciones localizadas en su mismo recurso.

Para cada acción a_j en respuesta al evento e_i definimos el *exceso* de dos maneras diferentes que son las siguientes:

Exceso de tiempo de respuesta:

$$exc(a_j)_i = \begin{cases} (r_{ij} - d_j)\dfrac{GR_{ik}}{ED_{ik}} & ;\ cuando\ \ d_j \le T_j \\ (r_{ij} + J_{ij} - d_j)\dfrac{GR_{ik}}{ED_{ik}} & ;\ cuando\ \ d_j > T_j \end{cases}$$

Exceso de tiempo de computación:

$$exc(a_j)_i = \begin{cases} (-s_{ij})\dfrac{GR_{ik}}{ED_{ik}} & ;\ cuando\ d_j \le T_j \\ (r_{ij}+J_{ij}-d_j)\dfrac{GR_{ik}}{ED_{ik}} & ;\ cuando\ d_j > T_j \end{cases}$$

donde,

T_j Es el período de la acción a_j.

r_{ij} Es el tiempo de respuesta local de la acción a_j (relativo a la activación de a_j) para el evento e_i.

J_{ij} Es el *jitter* correspondiente a la acción a_j para el evento e_i.

d_j Es el plazo local asignado a la acción a_j.

GR_{ik} Es el tiempo de respuesta global de peor caso de la acción a_k que tiene el plazo global de principio-a-fin en la respuesta al evento e_i.

ED_{ik} Es el plazo global de principio-a-fin de la acción a_k en la respuesta al evento e_i.

s_{ij} Es el *slack* de la acción a_j para el evento e_i, que se define como la cantidad de tiempo de ejecución (con signo negativo) que se debe restar a la acción a_j para que cumpla su plazo local (d_j), si a_j no es planificable, o la cantidad de tiempo de ejecución que se puede añadir a la acción a_j para que continúe siendo planificable, si ya es planificable.

Hay que reseñar que un *exceso* negativo indica que la acción correspondiente cumple sus requerimientos temporales holgadamente, mientras que un exceso positivo indica que no los cumple.

La razón de tener dos definiciones diferentes para el *exceso* correspondiente a una acción particular a_j en la respuesta a un evento e_i es que ambas hacen que el algoritmo *HOPA* se comporte de manera diferente. La primera definición, *exceso de tiempo de respuesta*, alcanza soluciones mucho más rápidamente debido a que el cálculo del *slack* es una operación bastante compleja. La segunda definición, *exceso de tiempo de computación*, puede llegar a alcanzar mejores soluciones, porque para plazos menores que los períodos el *slack* representa mucho mejor la idea del *exceso*. En el algoritmo *HOPA* se comienza utilizando la primera

definición, y después de un número determinado de iteraciones sin encontrar una solución, se intenta con la segunda. El motivo por el que se distinguen los casos en que $d_j > T_j$ o $d_j \leq T_j$ es que cuando los plazos locales son mayores que los períodos, el *jitter* tiene una enorme influencia (no en el propio tiempo de respuesta de la acción a_j, sino en el de las acciones de menor prioridad). Esta barrera para los plazos no es absoluta, y en algunos casos se puede considerar como barrera algún otro valor de la relación plazo/período. Para plazos menores o iguales que los períodos, el impacto del *jitter* no es tan fuerte y es menos importante que el *exceso* debido al *slack*, o el debido al tiempo de respuesta.

Las ecuaciones del *exceso* se pueden modificar para soportar el caso en el que existen plazos globales (distintos de los de principio-a-fin) asignados a determinadas acciones. En este caso, en lugar de usar GR_{ik} se debe usar la diferencia entre las respuestas globales (R_{ik}) de las acciones anterior y posterior a la acción a_j (de la que se está calculando el exceso), y que tienen un plazo global asignado. De la misma forma, se debe utilizar la diferencia entre los plazos globales (D_{ik}) correspondientes a esas acciones en lugar de ED_{ik}.

Para cada recurso procesador o de comunicaciones PR_k definimos el *exceso* en ese recurso de la forma siguiente:

$$exc(PR_k) = \sum_{a_j \in PR_k} exc(a_j)_i$$

Además, definimos el *máximo exceso* de todos los recursos y el *máximo exceso* de todas las acciones que pertenecen a la respuesta a un evento particular e_i, tal y como se muestra a continuación:

$$Mex(PR) = \max_{\forall PR_i} \left| exc(PR_i) \right|$$

$$Mex(e_i) = \max_{j \in A_i} \left| exc(a_j)_i \right|$$

donde, A_i es el conjunto de acciones que constituyen la respuesta al evento externo e_i.

A partir de estas definiciones de los tiempos de *exceso*, se calculan los nuevos plazos locales para cada acción como función de sus plazos locales antiguos y de sus respectivos *excesos* (el propio de la acción, y el del recurso y la secuencia de respuesta a los que

pertenece). El nuevo plazo local para la acción a_j en la respuesta al evento e_i se calcula de la manera siguiente:

$$d_{ij}(nuevo) = d_j(antiguo)\left(1+\frac{exc(PR_k)}{k_R * Mex(PR)}\right)\left(1+\frac{exc(a_j)_i}{k_a * Mex(e_i)}\right)$$

donde, $d_j(antiguo)$ es el plazo local asignado en la iteración anterior, k_r y k_a son constantes que controlan la influencia relativa de los recursos y de las acciones que componen una respuesta en el cálculo de los nuevos plazos locales. Cuanto más pequeños son los valores de estas constantes, mayor es su influencia. Los valores más adecuados para estas constantes oscilan entre 1.5 y 3.0. En el algoritmo *HOPA* comenzamos con valores de k_r=k_a=2.0, y posteriormente los vamos cambiando a 1.8, 3.0, 1.5, etc. hasta que se alcanza una solución satisfactoria.

Después de obtener los nuevos plazos locales para todas las acciones que forman parte de una respuesta, se realiza una operación de ajuste proporcional a los plazos globales. Para un plazo de principio-a-fin, el cálculo del plazo local correspondiente a la acción a_j en la respuesta al evento e_i se realizaría como sigue:

$$d_{ij} = \frac{d_{ij}(nuevo)}{\sum_{k \in A_i} d_{ik}(nuevo)} ED_{il}$$

donde, ED_{il} es el plazo de principio-a-fin de la acción a_l en la respuesta al evento e_i, y A_i es el conjunto de acciones que constituyen la respuesta al evento e_i. En el que estamos considerando correspondiente al *modelo lineal* de sistema distribuido, y dado que una acción solamente puede responder a un evento, el plazo local que se asigna a la acción a_j es d_j=d_{ij}.

Como criterio de parada para el algoritmo *HOPA* se limita el número máximo de iteraciones, y se detiene cuando se producen dos asignaciones iguales de plazos locales (en cuyo caso el algoritmo obtendría una y otra vez la misma solución), o bien cuando se encuentra una solución adecuada.

En el momento en el que el algoritmo encuentra una asignación de prioridades que hace que el sistema sea planificable, es capaz de una solución optimizada mejor mediante la

ejecución de algunas iteraciones más. Para determinar la bondad de una solución, y poder compararla con otras utilizamos el *índice de planificación* (*Ind*) definido en el apartado anterior (ver página 3-6).

3.4. Implementación de las técnicas de planificación

En los apartados anteriores de este capítulo hemos descrito dos técnicas para la asignación de prioridades a las tareas y a los mensajes que conforman los sistemas distribuidos de tiempo real estricto. Con objeto de poder evaluar la capacidad para alcanzar soluciones de estas técnicas, y también de poder realizar una comparación entre ambas, es necesario desarrollar una herramienta software que las implemente. Esta herramienta tendrá que afrontar la planificación de un sistema distribuido a partir de la descripción del mismo (según el *modelo lineal* definido en el capítulo 2), y de una serie de parámetros: unos característicos de los algoritmos que se implementan, y otros de control del tipo de planificación que se quiere realizar.

Para permitir la aplicación de esta herramienta de asignación de prioridades sobre un gran número de casos y dar mayor validez a los resultados obtenidos, se va a desarrollar también una herramienta de generación automática de ejemplos. A partir de la descripción de un sistema dado, esta herramienta actuará en el sentido de modificar de distintas maneras las cargas de los recursos de ese sistema. Esto nos permitirá ejecutar los algoritmos de planificación de forma reiterada sobre el mismo sistema, pero con diferentes cargas en sus recursos, lo que dará una mayor fiabilidad a las pruebas de los algoritmos implementados. Como lenguaje de desarrollo para ambas herramientas hemos elegido Ada.

Por último, mostraremos la aplicación de las herramientas desarrolladas a un ejemplo complejo de sistema distribuido con requerimientos de tiempo real estricto que responde al *modelo lineal*.

3.4.1. Herramienta de planificación para sistemas distribuidos

Llegados a este punto, vamos a desarrollar una herramienta para la planificación de sistemas distribuidos de tiempo real estricto que incluye los algoritmos de asignación de

prioridades *HOPA* y *templado simulado*. La herramienta de planificación toma como entrada la descripción del sistema distribuido (con toda la información referente al mismo) más los parámetros de control de los algoritmos; le aplica la técnica o técnicas de planificación seleccionadas, y da como resultado si el sistema es o no planificable (en caso afirmativo se da también la asignación de prioridades encontrada), junto con cierta información adicional acerca del propio sistema y de los algoritmos de planificación empleados.

Los algoritmos de planificación se apoyan en el análisis continuado del sistema, por lo que se tiene que implementar la técnica de análisis para este tipo de sistemas. Así pues, se puede pensar en utilizar esta herramienta de planificación también para el análisis de sistemas distribuidos de tiempo real. Por otra parte, en algunas ocasiones puede resultar interesante que determinadas acciones del sistema tengan un orden relativo en sus prioridades. Para resolver estos casos introducimos el concepto de *prioridad preasignada* que es una prioridad dada a priori a una acción del sistema relativa a otras acciones con prioridad preasignada dentro de un recurso. La herramienta de planificación tratará de asignar prioridades a las acciones preservando el orden relativo a las prioridades preasignadas en cada recurso del sistema. Si todas las acciones del sistema tuvieran una prioridad preasignada la herramienta realizaría únicamente el análisis, ya que sería imposible intercambiar ninguna prioridad.

Los elementos principales de esta herramienta de planificación se pueden agrupar en los siguientes puntos:

Descripción del sistema distribuido. En este punto se encuadran los aspectos referentes a la representación del sistema distribuido para que se puedan aplicar de forma automática los algoritmos de análisis y planificación.

Técnica de análisis. Aquí describimos los detalles de la técnica de análisis que se utiliza.

Templado simulado. Características particulares de la implementación de este algoritmo de planificación.

Algoritmo HOPA. Al igual que en el punto anterior, aquí se agrupan las características particulares de la implementación de este algoritmo.

Presentación de resultados. En este punto se describe el tipo de resultados que se obtienen, además de cierta información adicional sobre la evolución de los algoritmos.

A continuación desarrollaremos cada uno de estos puntos con mayor profundidad.

3.4.1.1. Descripción del sistema distribuido

La descripción del sistema distribuido debe recoger además de la información correspondiente a la arquitectura tanto hardware como software del sistema, la correspondiente a los eventos y a los requerimientos temporales que se imponen sobre ellos. El sistema distribuido que se pretende analizar o planificar se describe en un fichero en el que aparece la siguiente información:

- Dimensiones del sistema distribuido:
  - Número secuencias de eventos externos que llegan al sistema.
  - Número de tareas de que consta el sistema.
  - Número de mensajes que se transmiten entre las tareas.
  - Número de procesadores.
  - Número de redes de comunicación.

- Características de cada secuencia de eventos:
  - Identificador del evento.
  - Período.
  - Identificador de la acción a la que activa (primera acción en su secuencia de respuesta).
  - Plazo de principio-a-fin (sobre la última acción de la secuencia de respuesta, que se puede extraer de la conectividad de las acciones).
  - Pares (identificador de acción, plazo global). Así se describen los plazos globales que los eventos tienen sobre determinadas acciones.

- Características de cada tarea:
    - Identificador de la acción.
    - Tiempo de ejecución de peor caso.
    - Secciones críticas (para el cálculo de términos de bloqueo).
    - Prioridad preasignada (en su caso).
    - Identificador del procesador en el que se localiza.
    - Conectividad con los diferentes mensajes. Para el *modelo lineal* de sistema distribuido consistirá en los identificadores del mensaje de entrada y del mensaje de salida.

- Características de cada mensaje:
    - Identificador de la acción.
    - Tiempo de ejecución de peor caso.
    - Término de bloqueo. Si consideramos que los mensajes se transmiten partidos en paquetes de comunicaciones de longitud fija, el término de bloqueo será el tiempo correspondiente a la transmisión de un paquete.
    - Prioridad preasignada (en su caso).
    - Identificador de la red en la que se localiza.
    - Conectividad con las tareas, es decir, identificadores de la tarea que origina el mensaje y de la tarea a la que va dirigido éste.

A partir de esta descripción del sistema, se pueden extraer las características particulares de cada acción (período, plazos, etc) necesarias para poder aplicar el análisis y los algoritmos de planificación.

Vemos ahora cómo se pueden obtener los términos de bloqueo para el análisis a partir de las secciones críticas. Vamos a considerar en el análisis los términos de bloqueo que aparecen como consecuencia de la utilización de objetos compartidos (en el mismo procesador). Así pues, la información necesaria para el cálculo de los términos de bloqueo consiste en asociar a cada tarea una lista de secciones críticas en las que a cada una se le hace corresponder el identificador del objeto compartido y la duración de dicha sección crítica. Si suponemos que estamos utilizando un protocolo de protección de prioridad, a partir de esta

lista y conocida la asignación de prioridades, se pueden calcular los términos de bloqueo correspondientes a cada tarea. El procedimiento para calcular estos términos sería el siguiente:

1.- Se calculan los techos de prioridad para cada objeto compartido.

2.- Se calculan los términos de bloqueo de acuerdo con la siguiente fórmula:

$$B_i = \max_{\forall j} (cs_j \mid Techo(cs_j) \geq P_i \wedge P_j < P_i)$$

es decir, el bloqueo de la tarea *i* (B_i) es la máxima sección crítica correspondiente a una tarea *j* (cs_j), cuyo techo de prioridad para esa sección crítica ($Techo(cs_j)$) es mayor o igual que la prioridad de la tarea *i* (P_i), y cuya prioridad (P_j) es menor que la de la tarea *i*.

La versión actual de nuestra herramienta es experimental y no tenemos en cuenta este cálculo para los términos de bloqueo por considerar que los valores que se pueden llegar a manejar son despreciables comparados con los tiempos de respuesta de peor caso. Evidentemente, en una versión definitiva de la herramienta habría que incorporar estos efectos. Por lo tanto, despreciamos los términos de bloqueo o si fuera necesario podríamos considerar términos de bloqueo estáticos (calculados a priori) que llegan a producir una situación peor que el peor caso.

3.4.1.2. Técnica de análisis

La técnica de análisis que se utiliza es la que describimos en el capítulo 2, es decir, se pretenden calcular los tiempos de respuesta de peor caso para todas las acciones del sistema distribuido cuando estas tienen plazos arbitrarios (pueden ser tanto mayores como menores o iguales que los períodos). Este cálculo se basa únicamente en la asignación de prioridades que se haya realizado previamente al análisis.

El análisis se va realizar de dos maneras diferentes dependiendo del valor de un parámetro que hay que suministrar a la herramienta, y que especifica si se utiliza o no algún mecanismo para eliminar el *jitter* (como el servidor esporádico). Por lo tanto, las dos maneras de realizar el análisis son:

Sin jitter. El análisis se reduce al cálculo por una sola vez de los tiempos de respuesta de peor caso con un valor para el *jitter* de las acciones igual a cero.

Con jitter. El análisis consiste en el cálculo iterativo de los tiempos de respuesta de peor caso seguido del cálculo del *jitter* de las acciones, hasta que se alcance un valor estable para las respuestas y los *jitter*. El valor inicial para el *jitter* de cada acción es cero.

3.4.1.3. *Templado simulado*

El algoritmo de *templado simulado* (ver página 3-5) comienza con la elección de un punto inicial que es una asignación de prioridades aleatoria (compatible con las prioridades preasignadas).

Establecemos una serie de parámetros que tienen incidencia en el control del algoritmo, y que son los siguientes:

Temperatura inicial. Temperatura a partir de la que va a comenzar el enfriamiento.

Número de iteraciones sin disminuir la energía mínima para el equilibrio térmico. Determina la condición de equilibrio térmico.

Número de iteraciones sin disminuir la energía mínima para la parada del algoritmo. Determina la condición de parada del algoritmo.

Número de veces que ejecuta un salto aleatorio. Es el número de veces que se va a ejecutar el algoritmo. Cuando el algoritmo se detiene sin alcanzar una solución o sin alcanzar las iteraciones de optimización, se vuelve a ejecutar pero a partir de un punto inicial diferente (un salto aleatorio).

Número de iteraciones sobre planificable. Es el número de iteraciones que ejecutará el algoritmo después de encontrar una solución para la asignación de

prioridades de un sistema. Nos permite delimitar el tiempo que el algoritmo va a emplear en encontrar una solución optimizada.

En nuestra implementación añadimos un lazo más para permitir la ejecución del algoritmo un número determinado de veces (mediante salto aleatorio) a partir de puntos iniciales distintos. La estructura de nuestra implementación del algoritmo de *templado simulado* es la que se muestra a continuación:

```
algoritmo templado simulado is
begin
      Elige un punto aleatorio inicial;
      Establece parámetros del algoritmo;
      loop
            loop
                  loop
                        E_actual := energía del punto p_actual;
                        Elige p_vecino, un vecino de p_actual;
                        E_vecino := energía del punto p_vecino;
                        if E_vecino < E_actual then
                              p_actual := p_vecino;
                        else
                                    E_actual - E_vecino
                              x = ---------------------
                                        K_actual
                              if e^x ≥ random (0,1) then
                                    p_actual := p_vecino;
                              end if;
                        end if;
                        exit when equilibrio térmico;
                  end loop;
                  K_actual := F(K_actual); -- Enfriamiento
                  exit when criterio de parada;
            end loop;
            exit when salida del algoritmo;
            Salto aleatorio a un punto;
      end loop;
end templado simulado;
```

La implementación del algoritmo de *templado simulado* viene caracterizada fundamentalmente por la implementación de las funciones y de las condiciones de parada propias del mismo, que se realiza de la forma siguiente:

Función energía. Para calcular la energía de un punto, en primer lugar se realiza el análisis del sistema con las prioridades establecidas para ese punto, después se calculan las respuestas globales de peor caso, y finalmente se

evalúan las distancias entre los plazos globales y estas respuestas para calcular el índice de planificación, cuyo valor es el opuesto de la energía.

Función vecino. Esta función elige un recurso y dos acciones dentro de ese recurso para intercambiar sus prioridades. Este intercambio de prioridades debe ser compatible con las prioridades preasignadas, es decir, no se puede realizar el intercambio de prioridades si da como resultado un orden relativo diferente al establecido por las prioridades preasignadas.

Función enfriamiento. La función que utilizamos es: $F(K) = 0.9\ K$. Este valor se ha obtenido experimentalmente como un valor que conduce a la mejor relacion entre velocidad de cálculo y calidad de las soluciones encontradas.

Equilibrio térmico. Consideramos que se alcanza el equilibrio térmico cuando el número de iteraciones sin disminuir la energía mínima alcanzada toma el valor del parámetro establecido para este caso. También se abandona el lazo controlado por esta condición en el caso de que se produzca alguna de las condiciones que determinan la parada del algoritmo.

Parada del algoritmo. El algoritmo se detiene cuando el número de iteraciones sin disminuir la energía mínima alcanzada toma el valor del parámetro establecido para este caso. También se detiene cuando se alcanza el número de iteraciones predeterminado para encontrar una solución optimizada.

Salida del algoritmo. La condición de salida del algoritmo viene determinada por la obtención de una solución (o solución optimizada en su caso), o por haber sobrepasado ya el número de intentos permitido sin obtenerla.

Por último, únicamente queda comentar que el salto aleatorio que se realiza cuando se detiene el algoritmo para obtener el nuevo punto de comienzo, se puede hacer aplicando un número determinado de veces la función vecino (tomando como nuevo punto en cada aplicación de la función el obtenido en la aplicación anterior).

3.4.1.4. Algoritmo *HOPA*

Para comenzar con la descripción de la implementación del algoritmo *HOPA*, vamos a ver la manera en la que se definen y se aplican los valores de las constantes k_r y k_a que forman parte del algoritmo, así como la manera en la que se utilizan las dos definiciones del exceso de una acción. El algoritmo evoluciona de acuerdo con dos listas de parámetros definidos en la etapa de inicialización, que son las siguientes:

Lista k. Es una lista de parejas de valores (k_r,k_a). Damos al algoritmo una lista que contiene los valores de las constantes con los que va a operar. Estas parejas se irán aplicando consecutivamente de la forma en que se mostrará más adelante en el modo de operación del algoritmo. Un ejemplo de estas parejas de valores es el siguiente: (2.0,2.0) (1.8,1.8) y (3.0,3.0).

Lista de iteraciones. Es una lista de iteraciones consecutivas para cada valor de una pareja (k_r,k_a). En esta lista cada elemento representa el número de iteraciones consecutivas que se aplica con una pareja de valores (k_r,k_a). El algoritmo comienza con el primer elemento de la lista y termina con el último. Un ejemplo de esta lista puede ser: 10, 20, 30, 40, 50.

También en este algoritmo se define un parámetro que establece el número de iteraciones que ejecutará el algoritmo después de encontrar una solución para la asignación de prioridades que hace que el sistema sea planificable (*número de iteraciones sobre planificable*). De igual manera que en el *templado simulado*, este parámetro nos permite delimitar el tiempo que el algoritmo va a emplear en optimizar la solución encontrada.

Según estos parámetros podemos describir el modo de operación del algoritmo *HOPA* de la forma en que se muestra a continuación:

```
algoritmo HOPA is
begin
      Establece los valores de los parámetros;
      Asigna plazos locales iniciales;
      for dimensión de lista de iteraciones
      loop
            Establece número de iteraciones;
```

```
            for dimensión de lista k
            loop
                  Establece los valores de k_r y k_a;
                  for definición de exceso
                  loop
                        Establece definición a aplicar;
                        for número de iteraciones
                        loop
                              Asigna prioridades de acuerdo con los
                              plazos;
                              Calcula los tiempos de respuesta de
                              peor caso (análisis);
                              exit when criterio de parada;
                              Calcula nuevos plazos locales;
                        end loop;
                  end loop;
            end loop;
      end loop;
end HOPA;
```

El lazo correspondiente a la definición del *exceso* solamente se realiza dos veces: la primera con el *exceso de tiempo de respuesta*, y la segunda con el *exceso de tiempo de computación*. A continuación vamos a describir brevemente la implementación de las operaciones que se realizan en el algoritmo:

Asignación de plazos locales iniciales. La asignación de plazos locales debe cumplir dos requisitos: preservar los plazos globales, y dar lugar a una asignación de prioridades compatible con las prioridades preasignadas. Esta asignación se lleva a cabo en los siguientes pasos:

1.- Asignación de plazos locales mediante el reparto de los plazos globales proporcionalmente a los tiempos de ejecución de peor caso entre las acciones que forman parte de la secuencia de respuesta al evento.

2.- Asignación de plazos forzados a las acciones con prioridades preasignadas y marcado de dichas acciones. Se recorre la lista de acciones con prioridades preasignadas de menor a mayor prioridad y se va chequeando que los plazos son compatibles con la preasignación. Si un plazo no es compatible se le fuerza a un valor que lo haga compatible (se reduce el plazo de acción con mayor prioridad preasignada para que se sitúe por debajo del plazo de la acción cuya prioridad preasignada es menor). En esta asignación forzada de plazos locales puede cierta cantidad de plazo global sobrante (no repartido).

3.- Se reparten las cantidades de plazo global sobrante entre las acciones que no están marcadas (no tienen prioridad preasignada) de manera proporcional a los plazos asignados en el paso 1.

Cálculo de nuevos plazos locales. El cálculo de los nuevos plazos locales se realiza siguiendo las mismas pautas que la asignación inicial de estos plazos. En este cálculo se llevan a cabo los siguientes pasos:

1.- Cálculo de los plazos locales a partir de los *excesos* tal como se muestra en la página 3-13.
2.- Asignación de plazos forzados a las acciones con prioridades preasignadas y marcado de dichas acciones. Igual que en la asignación inicial de plazos locales.
3.- Reparto de las cantidades de plazo global sobrante. Igual que en la asignación inicial de plazos locales.

Asignación de prioridades. Las prioridades se asignan de manera independiente en cada recurso según los plazos locales (*deadline monotonic*).

Cálculo de tiempos de respuesta de peor caso. Para el cálculo de las respuestas de peor caso se aplica el análisis implementado.

Criterio de parada. El algoritmo se detiene cuando se verifica alguna de las circunstancias siguientes:

- Se agota el número de iteraciones el algoritmo.
- Se encuentra una asignación de prioridades que hace que el sistema sea planificable y no se ha elegido la opción de optimizar la solución.
- Se sobrepasa el número preestablecido de iteraciones después de encontrar una asignación de prioridades que hace al sistema planificable (límite de iteraciones para la optimización).
- Se producen dos asignaciones consecutivas de plazos locales que son idénticas (el algoritmo obtendría siempre la misma asignación).

3.4.1.5. Presentación de resultados

En una herramienta de planificación que trata de encontrar una asignación de prioridades que haga que el sistema sea planificable, el principal resultado lógicamente es esta asignación de prioridades. Sin embargo, resulta interesante además tener cierta información sobre el comportamiento del sistema con esa asignación, e incluso también sobre los propios algoritmos utilizados en la planificación cuando se trata de evaluarlos.

Por consiguiente, vamos a dar a continuación una relación de los aspectos más importantes que ofrece como resultado la herramienta de planificación:

- Asignación de prioridades tanto si el sistema es planificable como si no lo es. Cuando el sistema no es planificable se ofrece la mejor solución encontrada acorde con el índice de planificación.

- Indice de planificación para la asignación de prioridades.

- Resultados correspondientes al análisis para la asignación de prioridades que se presenta como solución. Estos resultados consisten en una relación de respuestas globales de peor caso para todas las acciones que constituyen el sistema acompañadas de los plazos globales que deben cumplir.

- Porcentaje de utilización de cada recurso, y porcentaje promedio de utilización de todos los recursos.

- Medidas de los tiempo de CPU consumidos por los algoritmos de planificación empleados en la obtención de las soluciones.

3.4.2. Herramienta de generación automática de ejemplos

En este apartado describimos las características de la herramienta de generación automática de ejemplos que hemos desarrollado con el fin de permitir el estudio del

funcionamiento de la herramienta de planificación de una forma más exhaustiva y sobre un número mayor de casos.

Esta herramienta actúa sobre una descripción de sistema distribuido de tiempo real estricto como la que vimos en el apartado anterior (ver página 3-16). A partir de esta descripción, la herramienta puede generar ejemplos de diferentes maneras según el modo de operación elegido. Estos modos de operación son los siguientes:

Variación de períodos. Consiste en la variación aleatoria del período y de los plazos globales correspondientes a un evento (seleccionado al azar). Esta variación se realiza multiplicando tanto el período como los plazos globales por un factor aleatorio (por ejemplo en el rango 0.5-2.0). Se modifican también los tiempos de ejecución de peor caso de las acciones que constituyen la respuesta a ese evento con motivo de que se mantengan invariables los porcentajes de carga de los recursos del sistema distribuido. La aplicación de este modo de operación sobre un sistema dado está limitada por un número que se suministra inicialmente (número de ejemplos que se quieren generar). Utilizando este modo de operación se puede verificar la estabilidad de los algoritmos ante diferentes valores de períodos y plazos que representan la misma carga para el sistema.

Variación de carga simple. Consiste en el aumento progresivo (en cada aplicación de esta herramienta) de la carga de un recurso en una cantidad que se especifica inicialmente (en tantos por cien). El recurso se elige aleatoriamente la primera vez que se aplica la herramienta sobre un sistema dado. Para realizar el aumento de carga se selecciona al azar una acción de ese recurso, y se ajusta su tiempo de ejecución de peor caso de manera siguiente:

$$C_i(nuevo) = C_i(antiguo) + (0.01 * porcentaje * T_i)$$

En este modo de operación el límite de aplicación de la herramienta sobre un problema dado es cuando se alcanza el 100% de utilización del recurso en el que se aumenta la carga.

Si se usa este modo de operación se puede observar cómo influye en la obtención de soluciones para los algoritmos de planificación el hecho de que exista una sobrecarga en un único recurso del sistema.

Variación de carga múltiple. Este caso es como el anterior con la diferencia de que el recurso en el que se aumenta la carga se elige aleatoriamente cada vez que se aplica la herramienta para obtener el nuevo ejemplo. Aquí el límite de aplicación de la herramienta se alcanza cuando todos los recursos llegan al 100% de utilización.
Este es el modo de generación de ejemplos más versátil en el sentido de que por un lado, permite un chequeo global de los algoritmos al generar ejemplos más generales (el aumento de carga es aleatorio, pero sobre todos los recursos), y por otro, permite realizar un estudio sobre los límites de planificabilidad de los sistemas (al aplicar los algoritmos de planificación se puede observar la distribución de carga a partir de la cual dejan de ser planificables).

La información sobre el modo de operación y sobre el parámetro que necesita cada modo de operación (número de ejemplos que se generan o porcentaje que se debe incrementar) se suministra en a través de un fichero.

La implementación de esta herramienta se realiza sobre la estructura de datos de la herramienta de planificación, por lo que únicamente se define un procedimiento que genera el próximo ejemplo y devuelve como parámetro si se debe terminar (cuando se alcanzan las condiciones de final de generación en el modo de operación establecido).

Esta herramienta nos permite ejecutar los algoritmos de planificación de forma reiterada sobre el mismo sistema con diferentes cargas en sus recursos, o con las mismas cargas pero distintos valores en los períodos y plazos. A continuación mostramos un ejemplo de utilización de estas herramientas:

```
loop
      Carga descripción del sistema en la estructura de datos;
      Ejecuta la planificación;
      Próximo ejemplo (out terminar);
      exit when terminar;
end loop;
```

3.5. Comparación de las técnicas de planificación

En este apartado vamos a comparar el funcionamiento de nuestro algoritmo heurístico de planificación con el derivado de la aplicación de la técnica de *templado simulado*. Como plataforma de comparación vamos a utilizar un ejemplo de sistema distribuido formado por ocho procesadores interconectados mediante tres redes de comunicación (Figura 3-1). Cualquier procesador puede enviar mensajes a cualquier otro procesador a través de cualquiera de las redes de comunicación. El sistema debe responder a siete secuencias de eventos y cada uno de estos eventos genera una secuencia de respuesta compuesta por tareas y mensajes de la forma en que se muestra en la Figura 3-2. La asignación de tareas a procesadores y de mensajes a redes de comunicación es la que se refleja en la Figura 3-3.

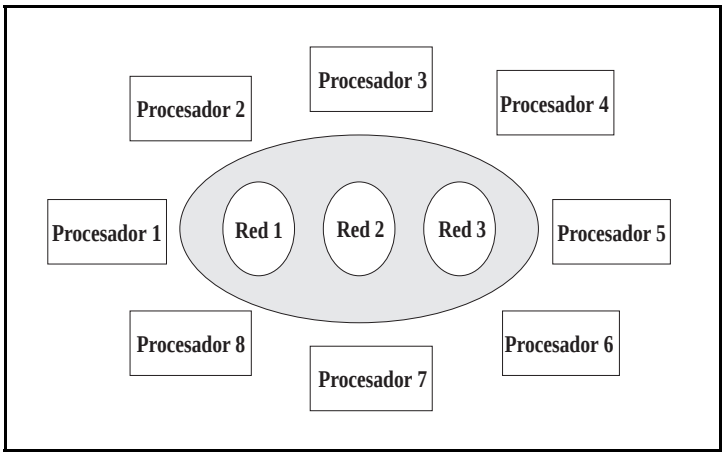

Figura 3-1. Ejemplo de sistema distribuido de tiempo real.

Los requerimientos temporales que definimos sobre los eventos del sistema son los siguientes:

e_1	-->	$T_1 = 400$	$D_{1\ 15} = 4000$
e_{16}	-->	$T_{16} = 500$	$D_{16\ 30} = 5000$
e_{31}	-->	$T_{31} = 700$	$D_{31\ 45} = 7000$
e_{46}	-->	$T_{46} = 450$	$D_{46\ 60} = 4500$
e_{61}	-->	$T_{61} = 550$	$D_{61\ 75} = 5500$
e_{76}	-->	$T_{76} = 350$	$D_{76\ 84} = 3500$
e_{85}	-->	$T_{85} = 500$	$D_{85\ 93} = 5000$

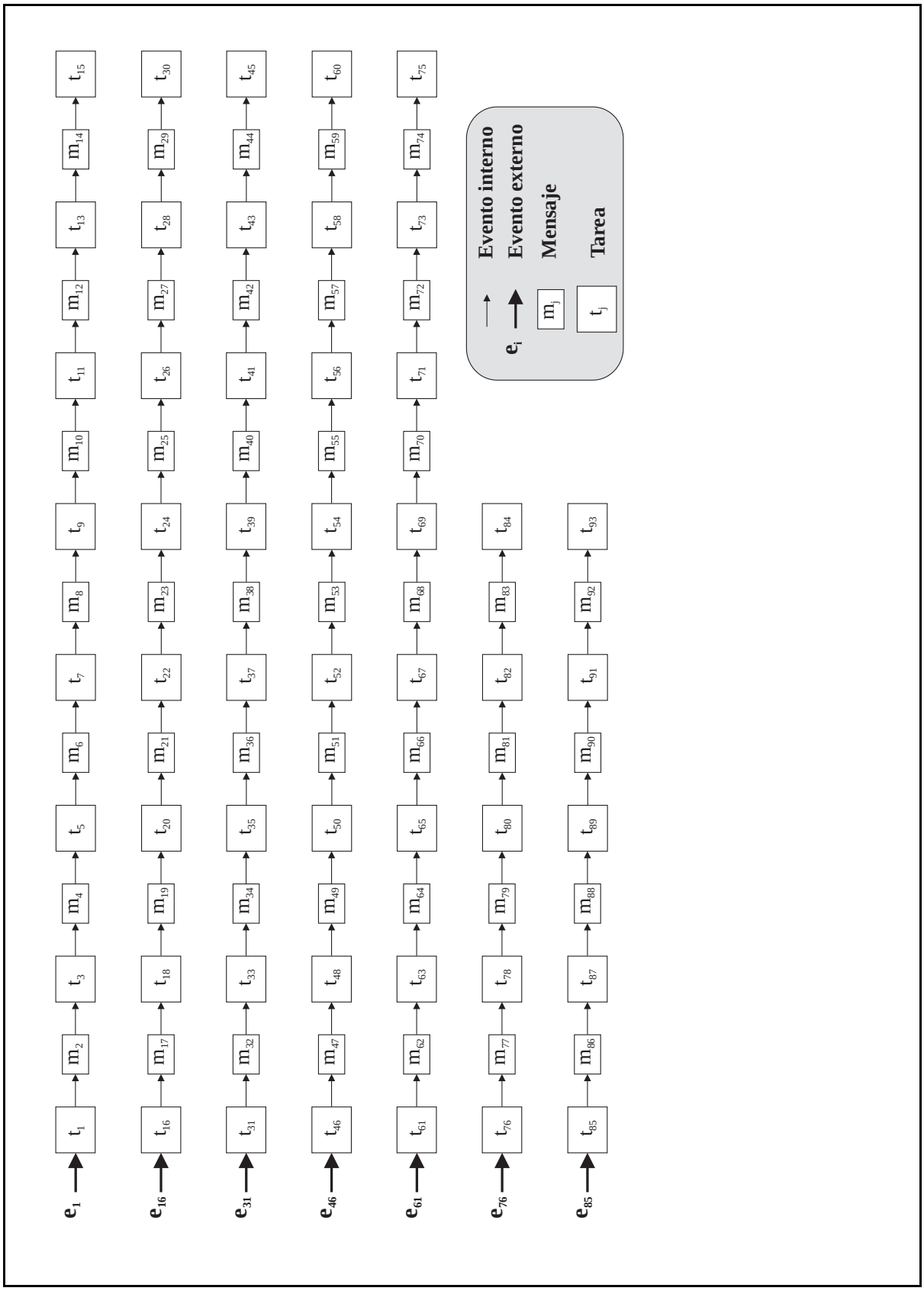

Figura 3-2. Estructura de las secuencias de respuesta del ejemplo.

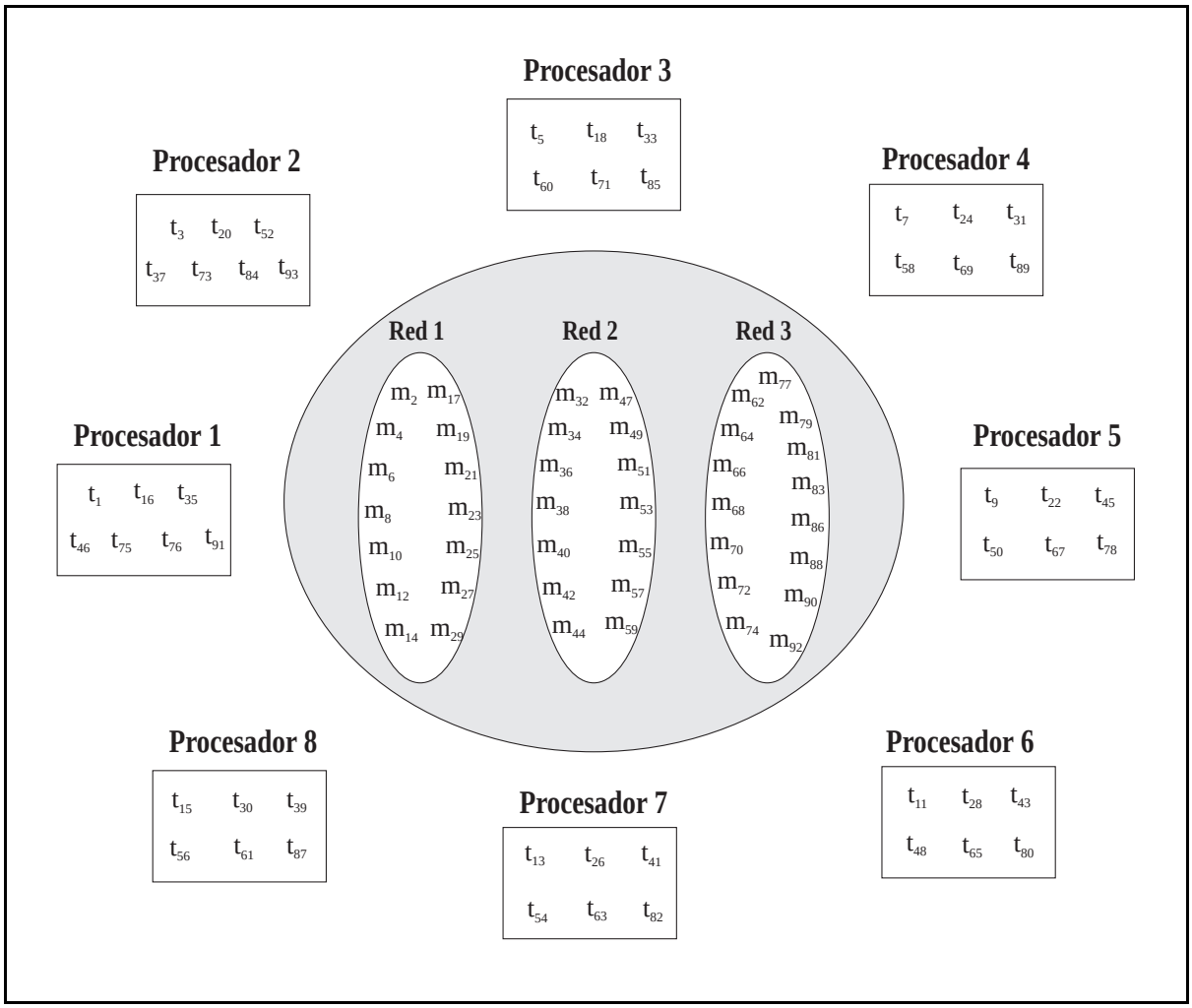

Figura 3-3. Asignación de tareas a procesadores y de mensajes a redes.

Aplicamos ambos algoritmos para resolver el problema de la asignación de prioridades a las acciones que componen el sistema. Para ello, utilizamos las herramientas de planificación y de generación de ejemplos desarrolladas. A partir de una carga inicial de sistema, aplicamos en primer lugar la herramienta de planificación para resolver el problema de la asignación de prioridades mediante las dos técnicas implementadas (incluyendo los efectos del *jitter*), y después aplicamos la herramienta de generación de ejemplos en el modo de *variación de carga múltiple* (con un aumento de carga del 2.35 %). Este proceso se repite hasta que se alcanza un valor de la carga para la que el sistema deja de ser planificable (ninguna de las dos técnicas encuentra una solución que haga al sistema planificable). La aplicación de la herramienta de planificación se va a realizar permitiendo la optimización de

las soluciones (5 iteraciones sobre una solución planificable para el algoritmo *HOPA*, y 15 para el *templado simulado*). Para dar una idea de la magnitud del problema que se está resolviendo, mencionaremos que el número de posibles asignaciones de prioridad para este ejemplo es del orden de 10^{58}.

Antes de mostrar los resultados obtenidos recordaremos que el índice de planificación representa una medida de la bondad de las soluciones en términos de la distancia entre los plazos globales y los tiempos de respuesta globales de peor caso. Los índices de planificación negativos representan que no se ha encontrado una solución planificable. Los resultados obtenidos de la aplicación de las herramientas al ejemplo (de la manera que hemos expuesto), son los que se muestran a continuación:

- La aplicación de la técnica de *templado simulado* a la asignación de prioridades con la función energía que hemos definido permite optimizar la asignación de prioridades. La Figura 3-4 muestra una gráfica en la que se representan los índices de planificación obtenidos para las diferentes cargas del ejemplo con la aplicación del algoritmo de *templado simulado* (se representa el índice de la primera solución encontrada y el de la solución después de la optimización). En la Figura 3-5 aparece una gráfica con los tiempos empleados en obtener la primera solución y la solución optimizada para las cargas del ejemplo. La diferencia entre estos tiempos es función del *número de iteraciones sobre planificable* que hayamos elegido (para este caso esa diferencia vale aproximadamente 20 s., y debido a la magnitud de los tiempos empleados apenas se aprecia en la gráfica).

- La aplicación del algoritmo *HOPA* también permite la optimización de la asignación de prioridades en el sentido de mejorar el resultado después de alcanzar la primera solución. La Figura 3-6 muestra la gráfica en la que se representan los índices de planificación de las soluciones obtenidas para las diferentes cargas del ejemplo (primera solución y solución optimizada), y en la Figura 3-7 se representan los tiempos empleados para obtener esas soluciones. Como se puede observar, el coste en tiempo de CPU empleado en la optimización de soluciones es realmente bajo comparado con la mejora lograda en ellas.

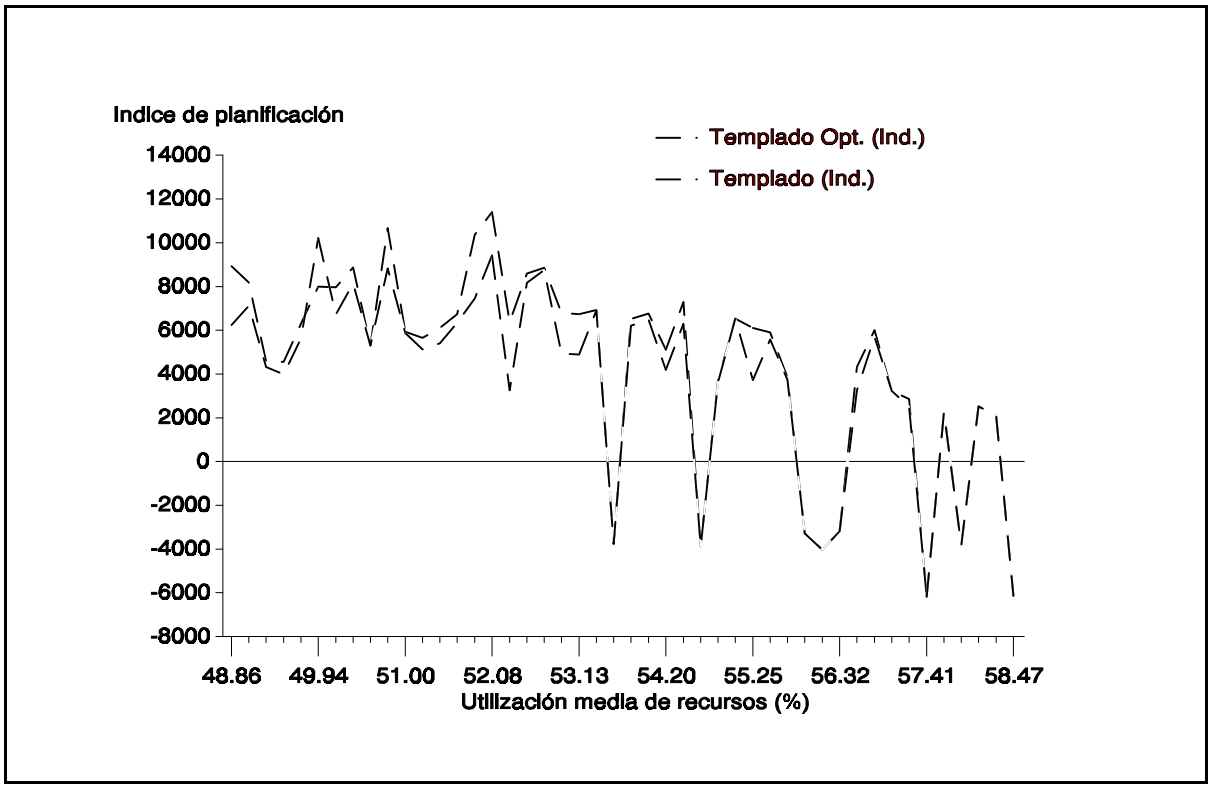

Figura 3-4. Indices de planificación para el *templado simulado*.

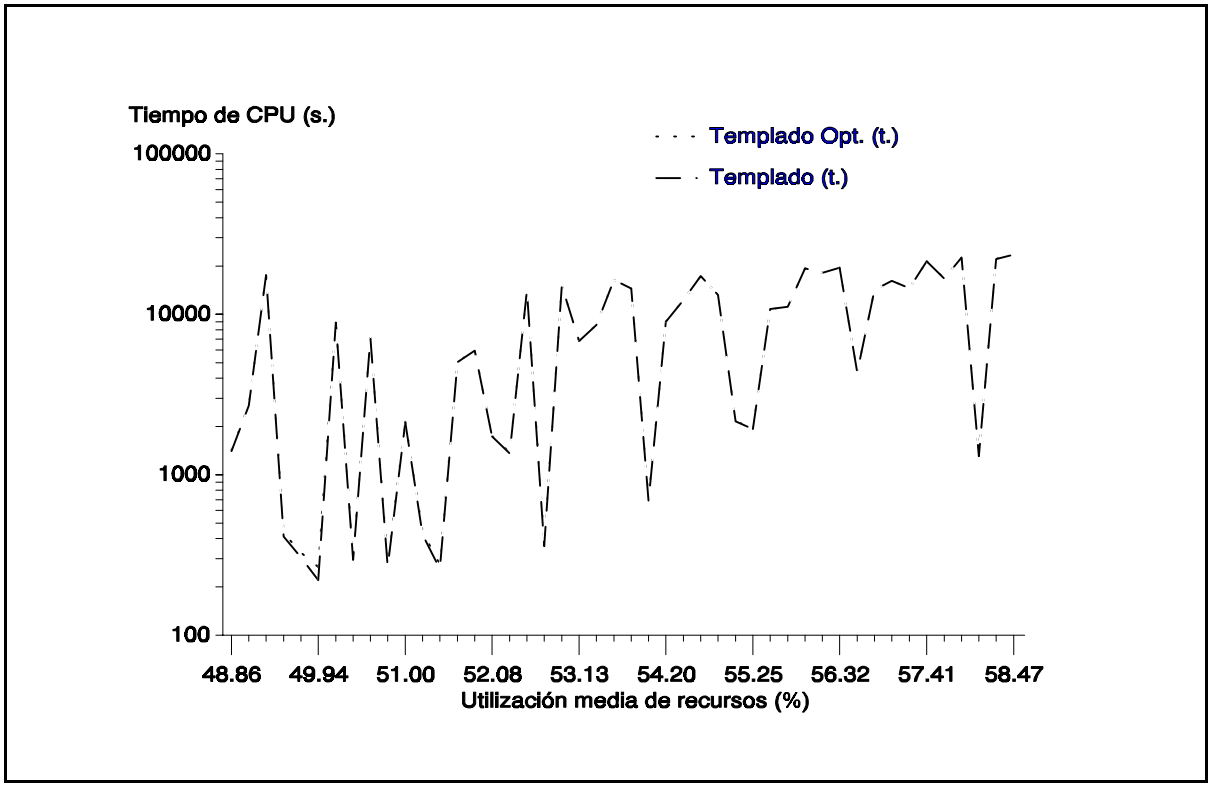

Figura 3-5. Tiempos de CPU para el *templado simulado*.

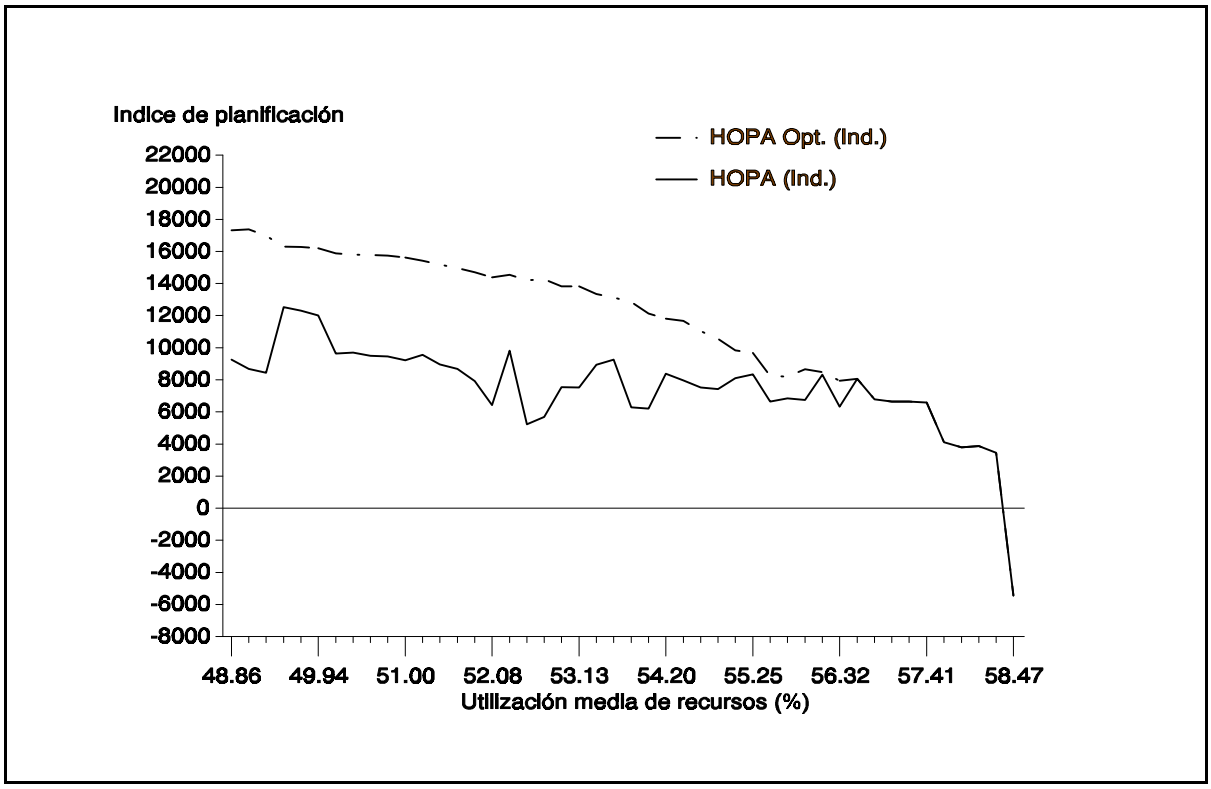

Figura 3-6. Indices de planificación para el algoritmo *HOPA*.

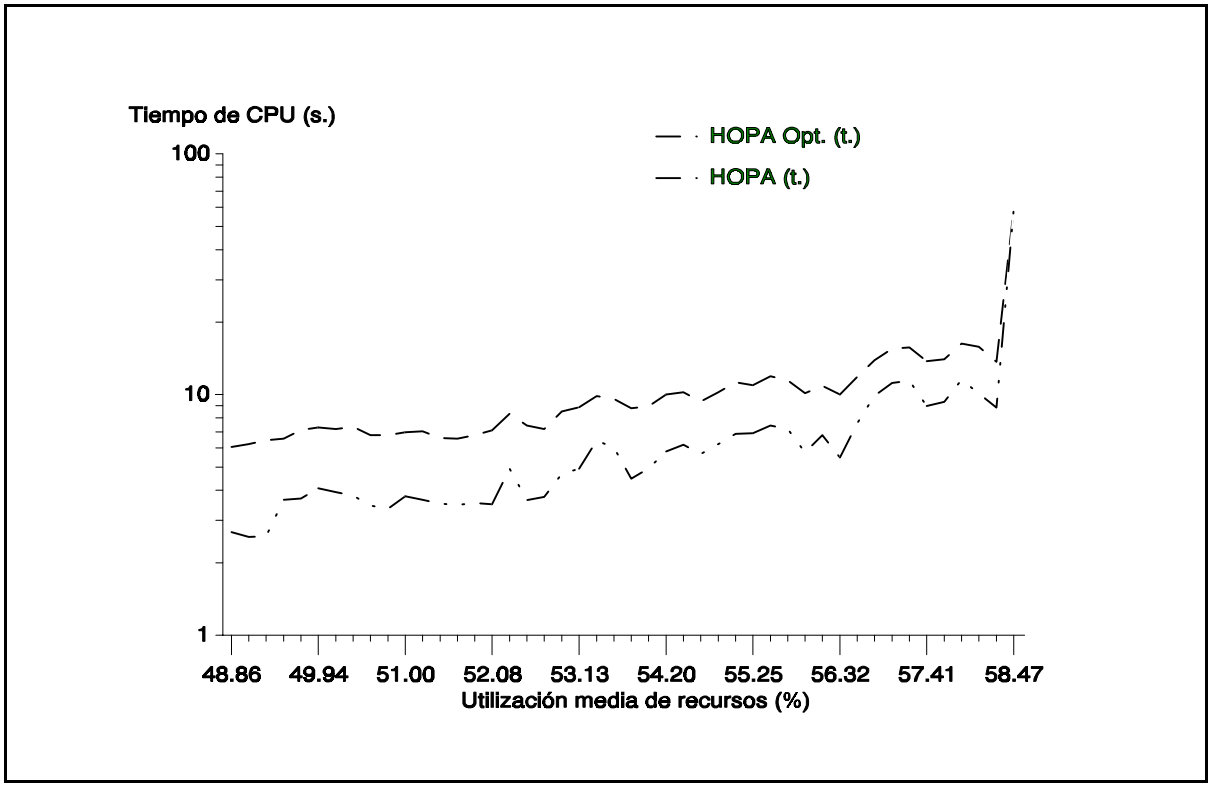

Figura 3-7. Tiempos de CPU para el algoritmo *HOPA*.

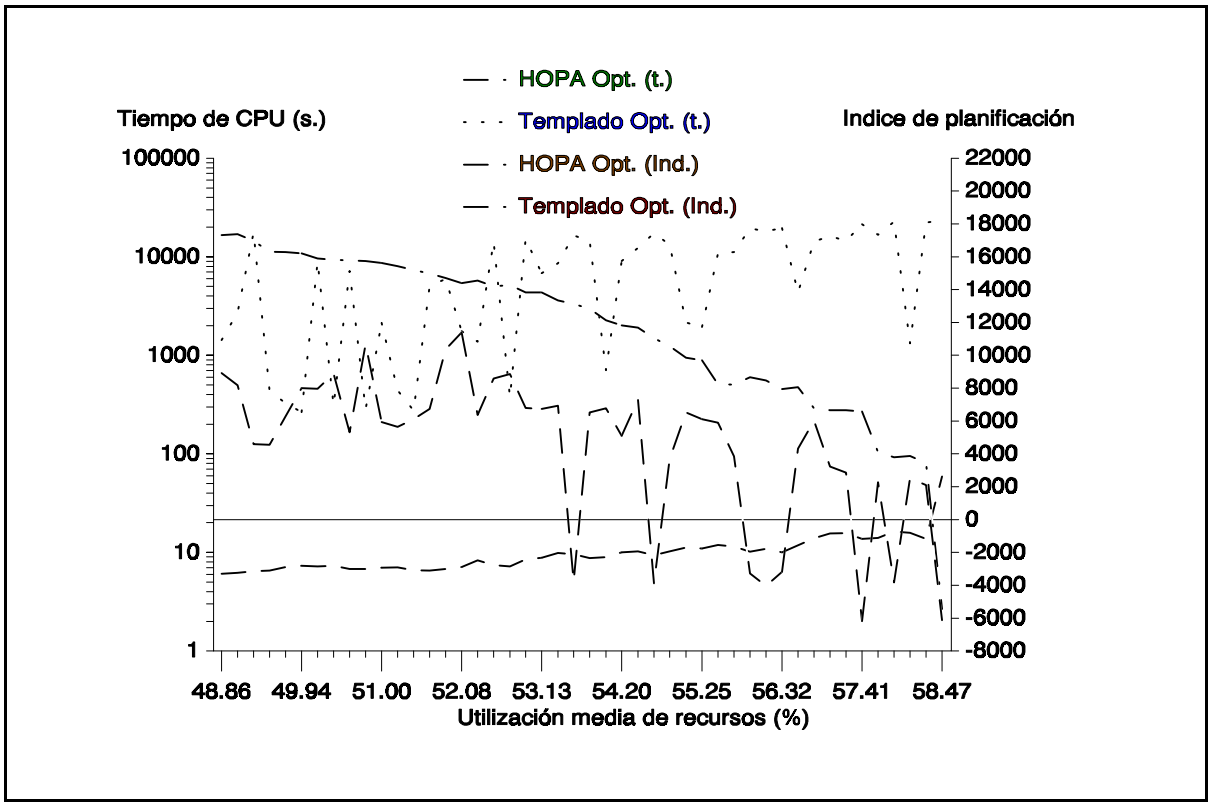

Figura 3-8. Comparación del algoritmo heurístico con el *templado simulado*.

- En la Figura 3-8 aparece una comparación entre las prestaciones de los dos algoritmos implementados en su aplicación a este ejemplo. En la gráfica se muestran los índices de planificación de las soluciones optimizadas y los tiempos de CPU que se han empleado en obtenerlas. Hay que destacar que en muchos de los casos en los que el *templado simulado* no ha sido capaz de encontrar una solución planificable, nuestro algoritmo heurístico sí que la ha encontrado. A partir de la Figura 3-8 se pueden obtener dos resultados importantes que son:
  - El algoritmo heurístico encuentra más y mejores soluciones que el *templado simulado*.
  - El tiempo de CPU consumido en la ejecución de nuestro algoritmo heurístico es en promedio dos órdenes de magnitud menor que el empleado por el *templado simulado*. Como ejemplo podemos poner que los tiempos de CPU consumidos por los dos algoritmos para generar los puntos de la gráfica que se muestra en la Figura 3-8 son:

HOPA	:	492.3 s.
Templado simulado	:	425284.0 s.

La herramienta de planificación se ha aplicado a muchos más ejemplos obteniéndose resultados similares. En la comparación de los dos métodos observamos que para bajas utilizaciones de los recursos, ambos encuentran soluciones factibles muy rápidamente, aunque las alcanzadas por el algoritmo heurístico son mejores (mayor índice de planificación). Sin embargo, cuando la utilización de los recursos es alta (y por tanto el problema más difícil de resolver), la diferencia en el rendimiento entre el *templado simulado* y el algoritmo heurístico es enorme. Esta diferencia en favor del algoritmo heurístico es tanto mayor cuanto más dilatados sean los plazos con respecto a los períodos (situación muy común en los sistemas distribuidos).

4. Extensión del análisis de tiempo real y de la planificación de sistemas distribuidos

4.1. Introducción

Como hemos visto en los capítulos anteriores el análisis para sistemas distribuidos que responden al *modelo lineal* está relativamente bien definido, aunque es ligeramente pesimista y en cierta medida complejo. Sin embargo, este modelo no considera la posible sincronización entre tareas (salvo la exclusión mutua en el acceso a recursos compartidos), ni la activación de las mismas por combinaciones de eventos, y tampoco considera la posibilidad de generación de múltiples eventos por parte de una tarea. Por esta causa, vamos a ampliar el actual modelo a un *modelo general* que cubra ese rango de posibilidades que hemos comentado, y de esta manera, seamos capaces de analizar además un mayor número de los sistemas de tiempo real que se presentan en la práctica.

La definición de un nuevo modelo requiere el desarrollo de un nuevo análisis aplicable a ese modelo. De esta forma, definiremos un análisis que nos permita calcular las respuestas de las acciones a los eventos en el peor caso para el *modelo general*. Este análisis se apoyará en el utilizado para el *modelo lineal*, y por supuesto, también tendrá en cuenta los efectos del *jitter*. Con el aumento de la complejidad del modelo y de los sistemas que somos capaces de representar, aumenta también la complejidad y la dificultad al realizar el análisis, que será ligeramente pesimista al tomar como referencia el aplicable al *modelo lineal*.

Una vez definido el análisis, extenderemos las técnicas de asignación de prioridades expuestas en el capítulo 3 (*templado simulado* y *HOPA*) para que se puedan aplicar sobre el *modelo general* de sistema distribuido de tiempo real estricto.

También ampliaremos las herramientas de planificación y de generación automática de ejemplos para poder evaluar algunos casos sencillos de sistemas distribuidos que respondan

al *modelo general*, y por último, daremos un ejemplo en el que a través de la aplicación de estas herramientas se muestre cómo se realizan tanto el análisis como la planificación desarrollados para el *modelo general* de sistema distribuido.

4.2. Modelo general de sistema distribuido

En este apartado vamos a extender el *modelo lineal* aplicable a sistemas distribuidos de tiempo real estricto, a otro modelo más general y representativo de un número mayor de este tipo de sistemas. De la misma forma que en el capítulo 2, consideramos un sistema distribuido constituido por varios recursos procesadores y uno o más recursos de comunicación. El *modelo general* que vamos a definir para estos sistemas distribuidos se asienta en los mismos principios que el *modelo lineal*. Estos principios son los que se enumeran y describen brevemente a continuación:

- Asumimos una planificación expulsora basada en prioridades fijas.

- Utilizamos la teoría *RMA* para el análisis del comportamiento temporal de peor caso.

- Tratamos a los mensajes en los recursos de comunicaciones de la misma manera que a las tareas en los recursos procesadores, excepto por el término de bloqueo que se añade debido a que los paquetes en los que se divide cada mensaje no son expulsables.

- Suponemos que las tareas se asignan estáticamente a los procesadores, y de igual manera los mensajes a las redes de comunicación.

Al igual que para el *modelo lineal*, definimos un modelo de sistema distribuido gobernado por eventos en el vamos a tener una serie de *eventos externos*. A estos eventos responden un conjunto de *tareas* y de *mensajes* que se activan entre sí mediante la generación de *eventos internos* (los pueden producir tanto las tareas como los mensajes). Suponemos también que todas las secuencias de eventos externos que llegan al sistema se conocen por adelantado, y que también se conocen los ritmos máximos con los que llegan estos eventos con requerimientos de tiempo real estricto. En este modelo, a diferencia del *modelo lineal*,

las características de los eventos internos (períodos y plazos) se pueden derivar de un solo evento externo o de una combinación de ellos. Además, suponemos que los eventos son instantáneos y por lo tanto no tienen influencia alguna en el análisis del sistema (cálculo de las respuestas de peor caso de las acciones a las que activan). Como en el *modelo lineal*, el efecto de sobrecarga producido por el hecho de que los eventos no sean instantáneos se puede añadir fácilmente al análisis considerando un tiempo de ejecución extra en las acciones sobre las que actúan los eventos.

Cada evento externo que llega al sistema genera una *respuesta* en forma de una *secuencia de acciones* que se activan entre sí mediante eventos internos. Como para el *modelo lineal*, cada una de estas acciones puede ser una tarea (o una parte de una tarea) ejecutada en un procesador, o un mensaje enviado a través de una red de comunicaciones (normalmente la primera acción de una respuesta es una tarea ejecutando en algún procesador determinado). De la misma manera, llamamos e_i al evento externo que activa a la acción a_i, y e_{jk} al evento interno generado por la acción a_j y que activa a la acción a_k. Vamos a considerar el caso más general en el que una acción puede generar uno o varios eventos internos y activar por tanto a otras acciones en el mismo o en diferentes *recursos* (procesadores o redes), y también puede ser activada por uno o varios eventos (internos, externos o una combinación de ambos). Los requisitos temporales que consideramos son los mismos que para el *modelo lineal*, es decir, los *plazos globales* (relativos a la llegada del evento), los *plazos locales* (relativos a la activación de cada acción en la respuesta) y los *plazos de principio-a-fin* (plazos globales correspondientes a las acciones terminales en la respuesta a un evento). Seguiremos llamando D_{ij} al plazo global de la acción a_j relativo a la llegada del evento e_i, d_{ij} al plazo local de la acción a_j para el evento e_i (relativo a la activación de la acción a_j), y d_j al menor de los plazos locales d_{ij} para cada acción a_j. El plazo de principio-a-fin también continuaremos denominándolo ED_{ij} (para la acción a_j relativo al evento e_i). Hay que observar que en el *modelo general*, una acción determinada a_j puede ejecutarse como consecuencia de la llegada de diferentes eventos, y por lo tanto, puede tener varios plazos locales o globales asignados.

De acuerdo con esta definición que hemos dado para el *modelo general* de sistema distribuido, la Figura 4-1 muestra los diferentes patrones a los que responden los eventos a la entrada y a la salida de las acciones, es decir, las diferentes combinaciones de eventos que pueden activar a una acción o las combinaciones de eventos que puede generar ésta. A

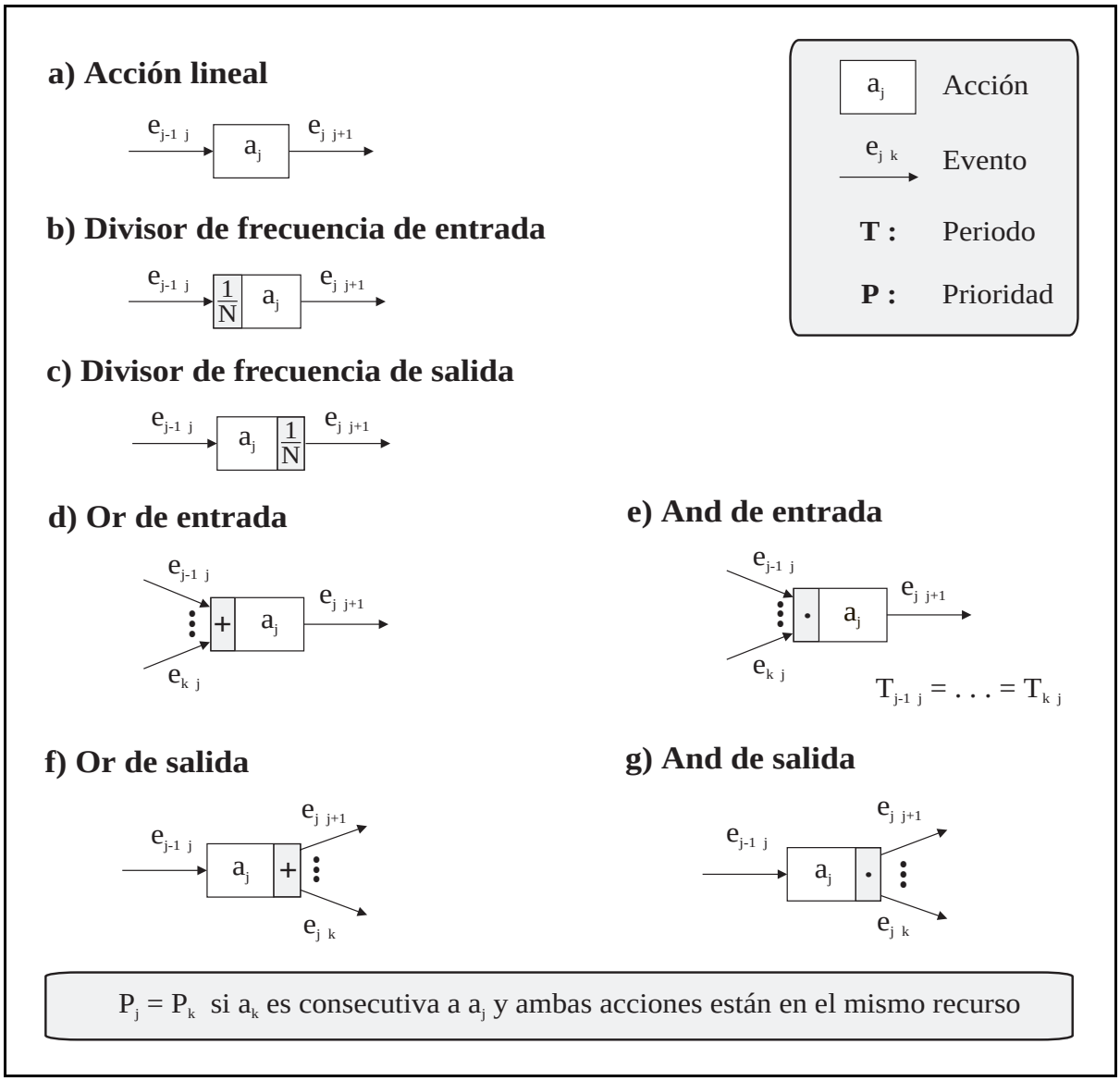

Figura 4-1. Modelo general de sistema distribuido gobernado por eventos.

continuación se describen de forma exhaustiva cada uno de los patrones que se contemplan en este *modelo general* de sistema distribuido:

a) *Acción lineal*. El caso de acción lineal es el correspondiente al *modelo lineal* descrito en el capítulo 2. La acción es disparada por un único evento y solamente puede generar un evento.

b) *Divisor de frecuencia de entrada*. Es aquella acción que se activa cuando ha recibido un número *N* de veces la llegada de un determinado evento. Al igual que para el caso

lineal también existe un único evento de entrada, aunque el disparo de la acción no sucede por la ocurrencia de un solo evento. Un ejemplo de este caso es un muestreo periódico de alguna señal efectuado por una tarea de manera que solamente cuando se ha recogido un determinado número de muestras, se va a ejecutar la tarea que posteriormente generará el evento o los eventos de salida.

c) *Divisor de frecuencia de salida*. Es aquella acción que se activa y ejecuta con la llegada de un evento, pero únicamente cuando se haya ejecutado un número *N* de veces se va a generar el evento o los eventos de salida correspondientes. Como para el caso anterior existe un único evento de entrada. Un ejemplo para este caso es el muestreo periódico de alguna señal efectuado por una tarea, en el que cada muestra necesita un preprocesado y solamente cuando se ha recogido un número determinado de muestras se genera el evento o los eventos de salida.

d) *Or de entrada*. Es el caso de una acción que puede ser disparada indistintamente por cada evento de un conjunto de ellos que constituyen los eventos de disparo de la acción. La acción se dispara con la llegada de cualquiera de los eventos de entrada. Como ejemplo para este caso podemos considerar un sistema de procesado de imagen en el que se tienen varias cámaras y una tarea por cada cámara recogiendo imágenes periódicamente. Estas tareas pueden realizar un procesado previo de las imágenes para enviárselas después a una única tarea que las va tratando una por una y toma algún tipo de decisión respecto a ellas (por ejemplo archivarlas). Esta tarea se activaría cada vez que hubiera una imagen disponible, y por lo tanto, responde a este patrón de eventos en la entrada.

e) *And de entrada*. Este caso corresponde a una acción que es disparada por una combinación de eventos diferentes pero con idénticos períodos. La acción se ejecuta únicamente en el momento en el que han llegado todos los eventos de entrada. No tiene sentido considerar que los eventos de entrada posean períodos diferentes, ya que esto supondría la acumulación de aquellos eventos más rápidos. Este caso responde a la típica situación en la que es necesario distribuir una serie de cálculos entre distintos procesadores y finalmente hay una tarea que recoge los resultados. Solamente se ejecuta la tarea cuando todos los resultados están disponibles.

f) *Or de salida*. Se corresponde con el caso en el que una acción puede generar distintos eventos de salida, pero genera solamente uno en cada período. Un ejemplo de este caso puede ser una cola de mensajes a la que una tarea envía mensajes (determinados datos para que sean procesados) y hay varias tareas dispuestas para recibirlos. La primera tarea que esté preparada para la recepción de un mensaje será la que procese los datos.

g) *And de salida*. Este es el caso en el que una acción dispara simultáneamente varios eventos. En el modelo de colas de mensajes, este caso se corresponde con una tarea que deposita mensajes en varias colas en las que hay una tarea por cola esperando recibirlos.

La última acción en la respuesta a un evento (generalmente una tarea) es un caso particular que responde a los patrones descritos para los eventos de entrada en los que no se genera evento alguno a la salida.

Sobre este modelo imponemos la restricción de que las acciones consecutivas localizadas en el mismo recurso deben tener la misma prioridad. Esta restricción nos va a permitir construir de una manera más sencilla la situación de peor caso para los sistemas en los que se den estas secuencias de acciones, y por lo tanto, nos facilitará el análisis de este *modelo general*. Se podría eliminar esta restricción incluyendo en el análisis de cada recurso el efecto de las prioridades variables mediante el uso de [GON91A].

Este modelo permite describir sistemas en los que aparezca cualquier combinación de acciones y eventos que respondan a los patrones mencionados. Hay que considerar como excepción aquella combinación en la que un evento que es generado por una acción con *or de salida* da lugar a la activación de una acción con *and de entrada* en algún punto de su secuencia de respuesta. En la Figura 4-2 se muestra esta situación y, como podemos observar, si existieran plazos globales sobre la acción a_k o sobre alguna otra acción posterior, no podríamos asegurar su cumplimiento porque no podemos precisar la periodicidad con la que se va a producir el evento $e_{k-1\ k}$ ($e_{j\ j+2}$ coincide con $e_{k-1\ k}$ si las acciones a_j y a_k son inmediatamente consecutivas). Hay que tener en cuenta la condición impuesta sobre las acciones con *and de entrada*, en las que todos los eventos de entrada deben tener el mismo período. Si no

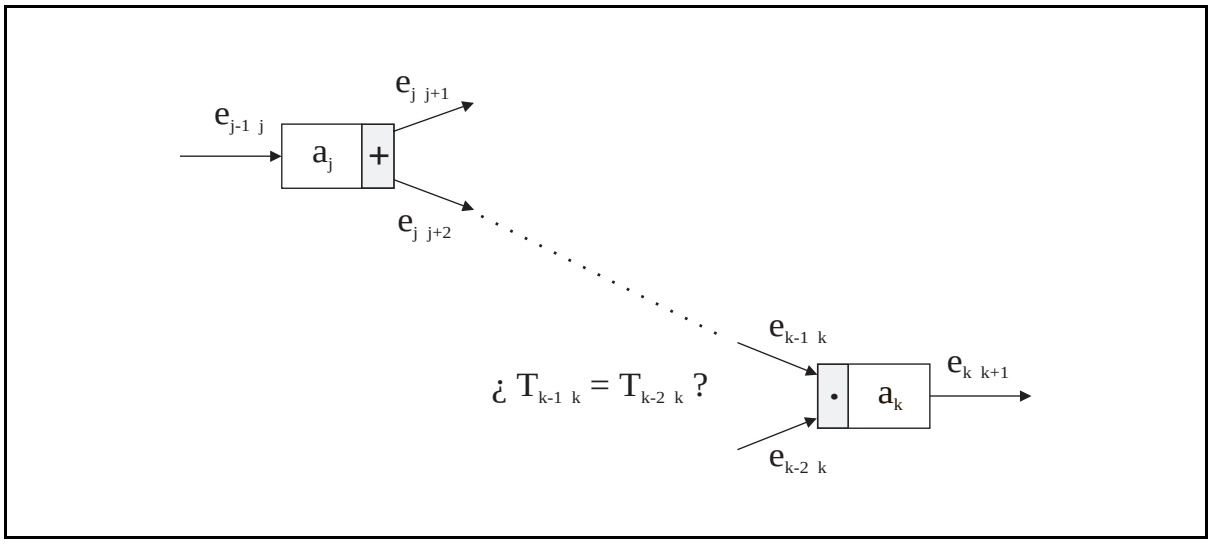

Figura 4-2. Combinación de patrones de eventos no permitida.

tuviéramos requerimientos de tiempo real sobre la acción a_k o posteriores, entonces podríamos permitir la existencia de esta combinación de patrones llegada y generación de eventos.

El modelo descrito se adapta bien para representar a un gran número de arquitecturas de tiempo real que se pueden encontrar en la práctica. Si consideramos un sistema distribuido en el que la comunicación entre las tareas se realiza a través de colas de mensajes [GUT95B], podemos describir mediante el *modelo general* todas las situaciones que se pueden encontrar en su implementación, tal y como se muestra en la Figura 4-3. Cuando dos tareas localizadas en el mismo procesador se comuniquen a través de una cola de mensajes, el mensaje se modelará como un evento y una cierta cantidad de tiempo de ejecución que habrá que añadir a cada tarea (sobrecarga debida al uso de las colas de mensajes).

De la misma manera que para el *modelo lineal*, podemos considerar el ejemplo de un sistema que utiliza una aproximación cliente-servidor con servidores remotos. También aquí, se puede descomponer cada parte de una tarea que solicita un servicio de un servidor remoto en la siguiente secuencia de acciones: la acción anterior a la solicitud del servicio, el mensaje enviado al servidor, la acción del servidor, la respuesta del servidor, y la acción posterior a la solicitud del servicio. Sin embargo, en este caso los servidores pueden dar servicio a más de un cliente, y por lo tanto, se van a modelar como una tarea con una *or de entrada* y una *or de salida* con ciertas restricciones sobre los eventos de salida. Estas restricciones deben tener en cuenta la correspondencia entre las entradas y las salidas para evitar las combina-

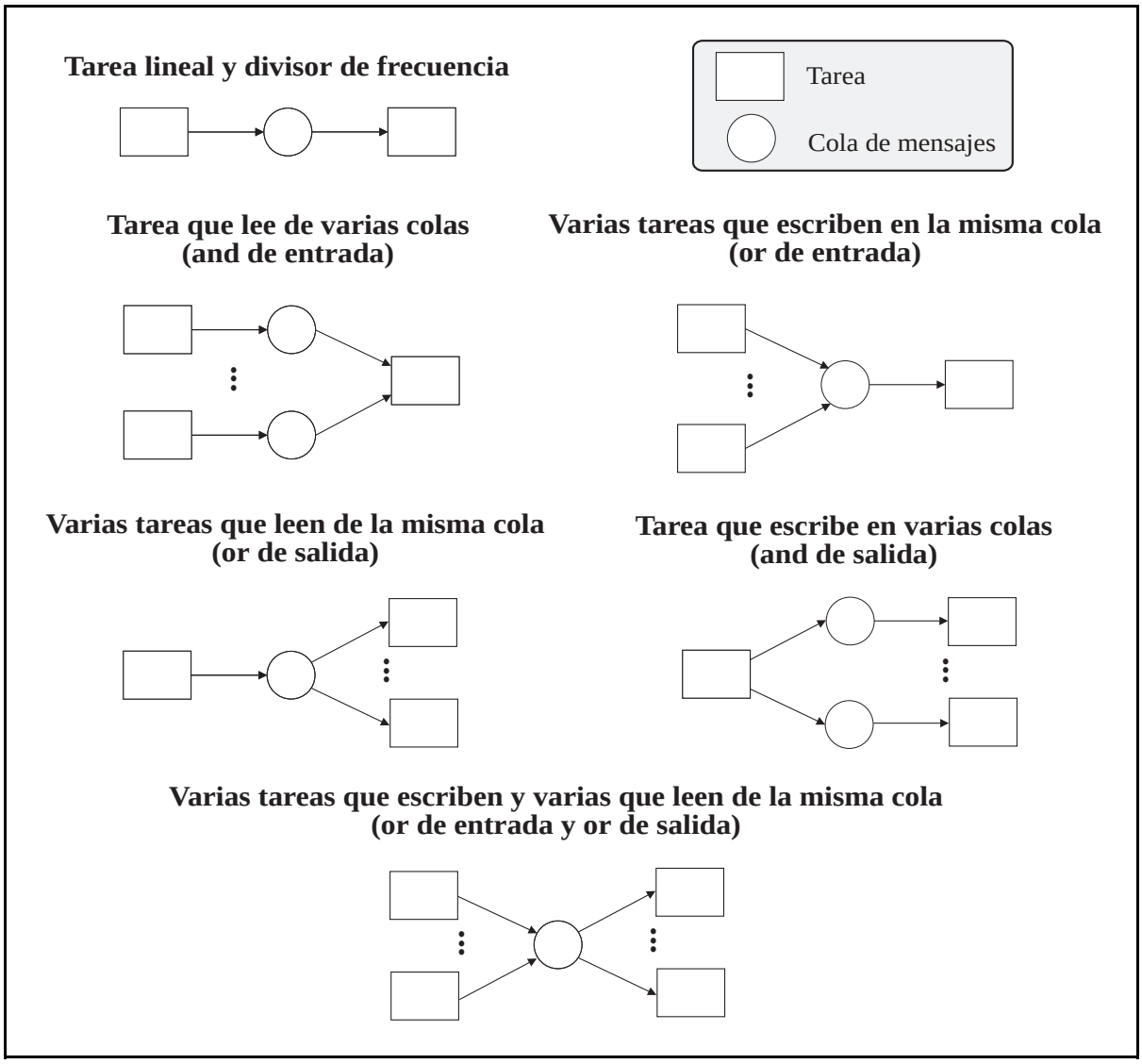

Figura 4-3. Modelo de colas de mensajes.

ciones de entradas y salidas que no son reales (a cada entrada le corresponde una salida, que siempre es la misma). La Figura 4-4 muestra el modelado de un sistema distribuido con dos nudos y una red de comunicaciones que utiliza la aproximación cliente-servidor.

Una vez definido el *modelo general* para los sistemas distribuidos, podemos considerar el *modelo lineal* como un caso particular de éste en el que únicamente se pueden dar las acciones lineales (con un único evento a la entrada y un único evento a la salida).

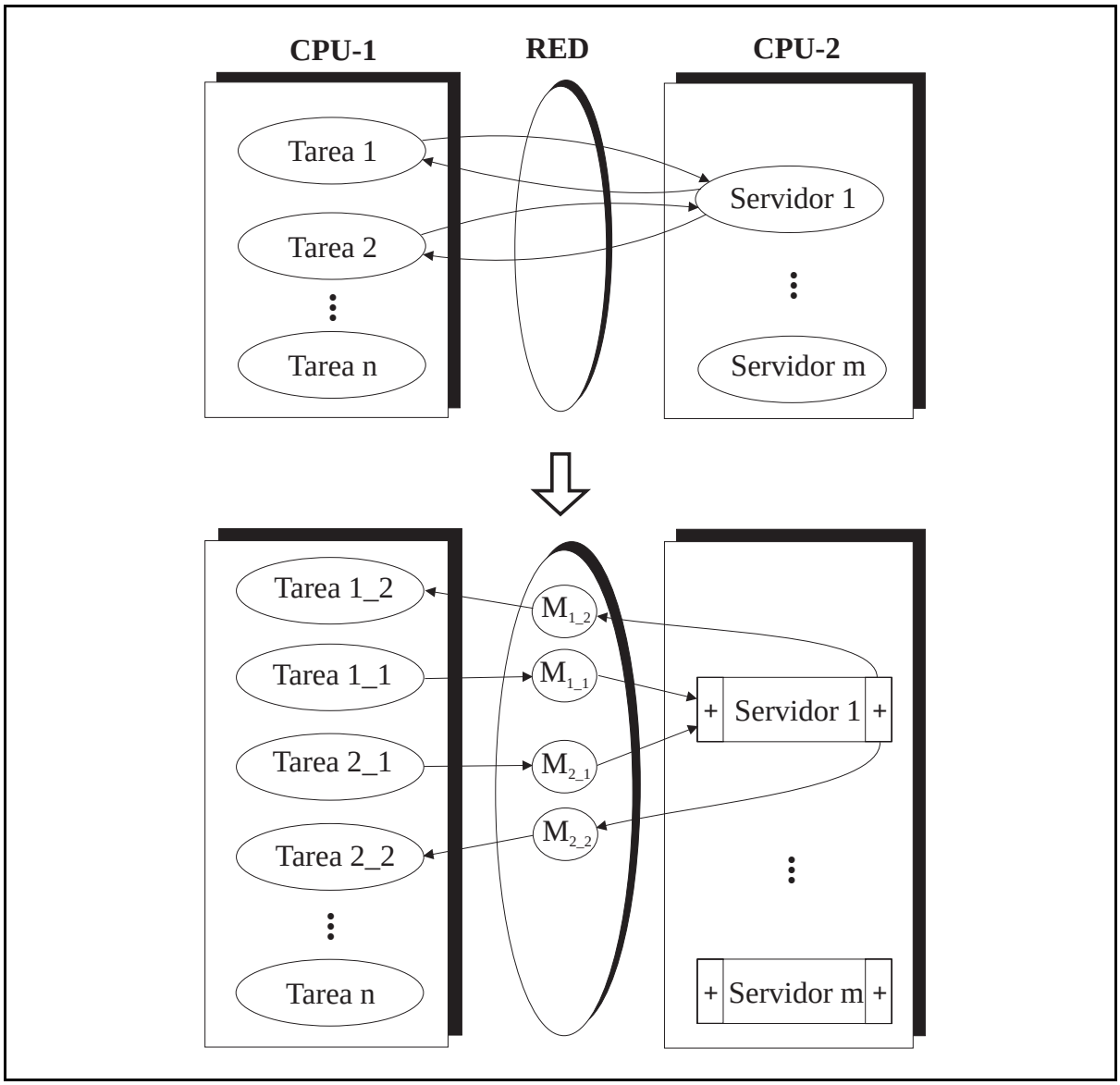

Figura 4-4. Modelo cliente-servidor con servidores remotos.

4.3. Extensión del análisis de tiempo real

Para la extensión del análisis de tiempo real al *modelo general* de sistema distribuido, nos apoyamos en el análisis desarrollado para el *modelo lineal*, y de la misma manera, consideramos cada recurso procesador y cada recurso de comunicaciones del sistema como si fueran independientes. La aproximación seguirá siendo ligeramente pesimista pero nos permitirá garantizar la planificabilidad de estos sistemas de tiempo real. Por consiguiente, el análisis se llevará a cabo aplicando las técnicas *RMA* introducidas en el capítulo 1 y aplicadas al *modelo lineal* en el capítulo 2 (por supuesto, teniendo en cuenta los efectos del *jitter*).

Sin embargo, no es posible construir una situación de peor caso mediante la aplicación de estas técnicas de análisis directamente sobre el *modelo general* de sistema distribuido. Por tanto, vamos a realizar la transformación de este modelo en un *modelo equivalente* sobre el que se podrá definir una técnica de análisis que por un lado, nos dará una situación de peor caso, y por otro, será capaz de reducir en algunos casos el grado de pesimismo introducido por el uso del análisis para el *modelo lineal*.

Así pues, a continuación veremos cómo extraer el *modelo equivalente* para cada uno de los patrones de llegada y generación de eventos definidos en el *modelo general*, y a partir de este *modelo equivalente* veremos cómo se define el análisis de tiempo real que nos va a permitir determinar la planificabilidad de este tipo de sistemas distribuidos.

4.3.1. Modelo equivalente para el análisis

Tal como hemos mencionado anteriormente, el *modelo equivalente* es una transformación del *modelo general* que permite la aplicación de una técnica de análisis de tiempo real sobre la descripción realizada del sistema distribuido. Este *modelo equivalente* establece los parámetros para el análisis correspondientes a las acciones (períodos y plazos) a partir de la descripción del sistema realizada mediante el *modelo general* (características de los eventos del sistema y de sus respuestas), y también determina las transformaciones que se deben realizar y el modo en el que se debe aplicar el análisis sobre algunos patrones de eventos que no son analizables directamente mediante la técnica descrita en el capítulo 2. A grandes rasgos el *modelo equivalente* pretende linealizar los patrones de llegada y generación de eventos definidos en el *modelo general* para que sean analizables.

A partir del *modelo general* de sistema distribuido de tiempo real estricto, la obtención del *modelo equivalente* para el análisis se lleva a cabo de la forma mostrada en la Figura 4-5 y en la Figura 4-6. Seguidamente describimos de manera detallada este *modelo equivalente* a partir de cada uno de los patrones definidos en el *modelo general*:

a) ***Acción lineal***

Este caso es el correspondiente al *modelo lineal* descrito en el capítulo 2, en el que los períodos de la acción y del evento que genera ésta, son iguales al período

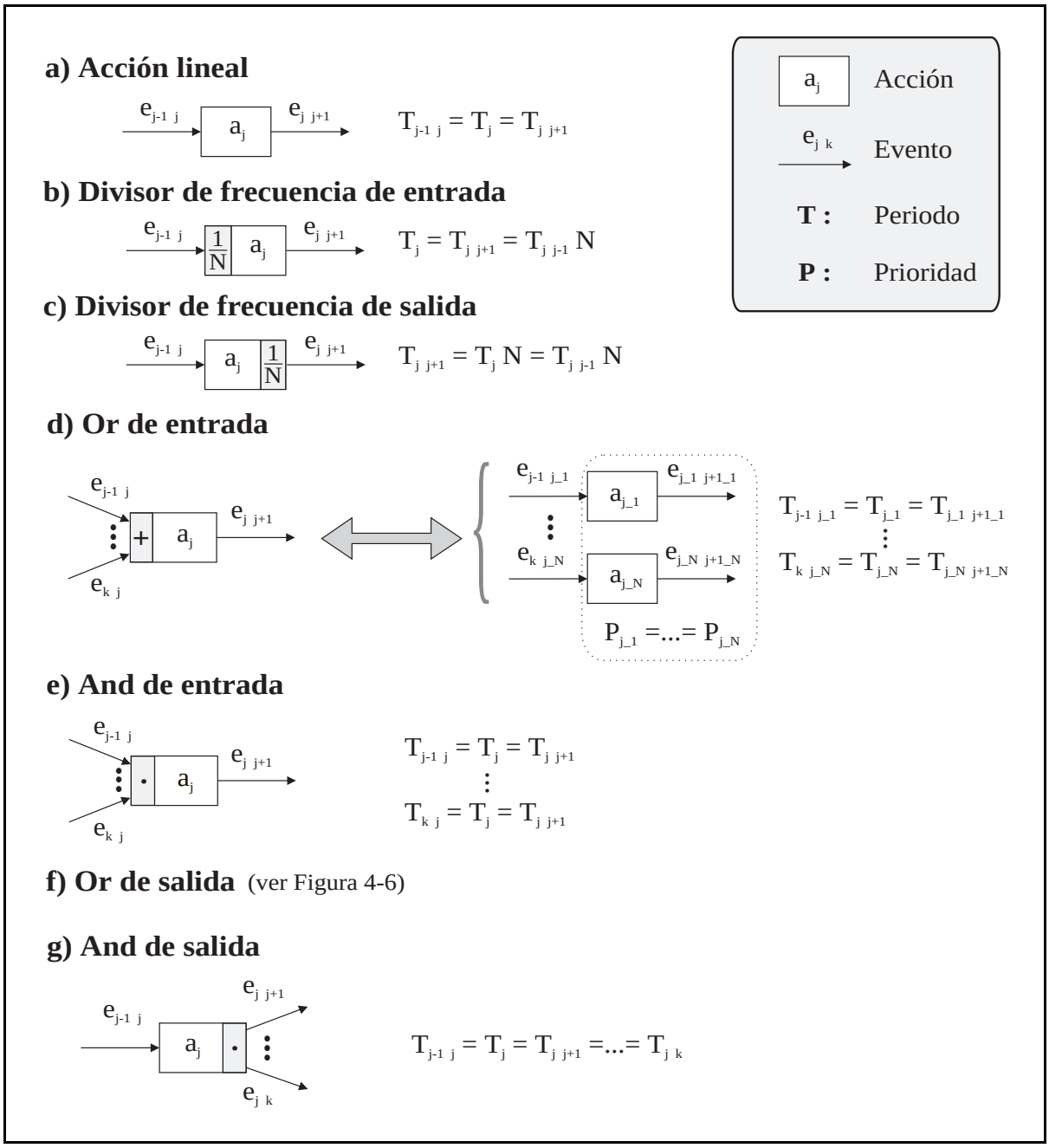

Figura 4-5. *Modelo equivalente* para el análisis.

del evento de entrada.

b) ***Divisor de frecuencia de entrada***

En este caso la acción se activa cuando ha recibido un número *N* de veces la llegada de un determinado evento, y por consiguiente, tanto su período como el de los eventos que puede generar resultan de multiplicar por *N* el período del evento que la

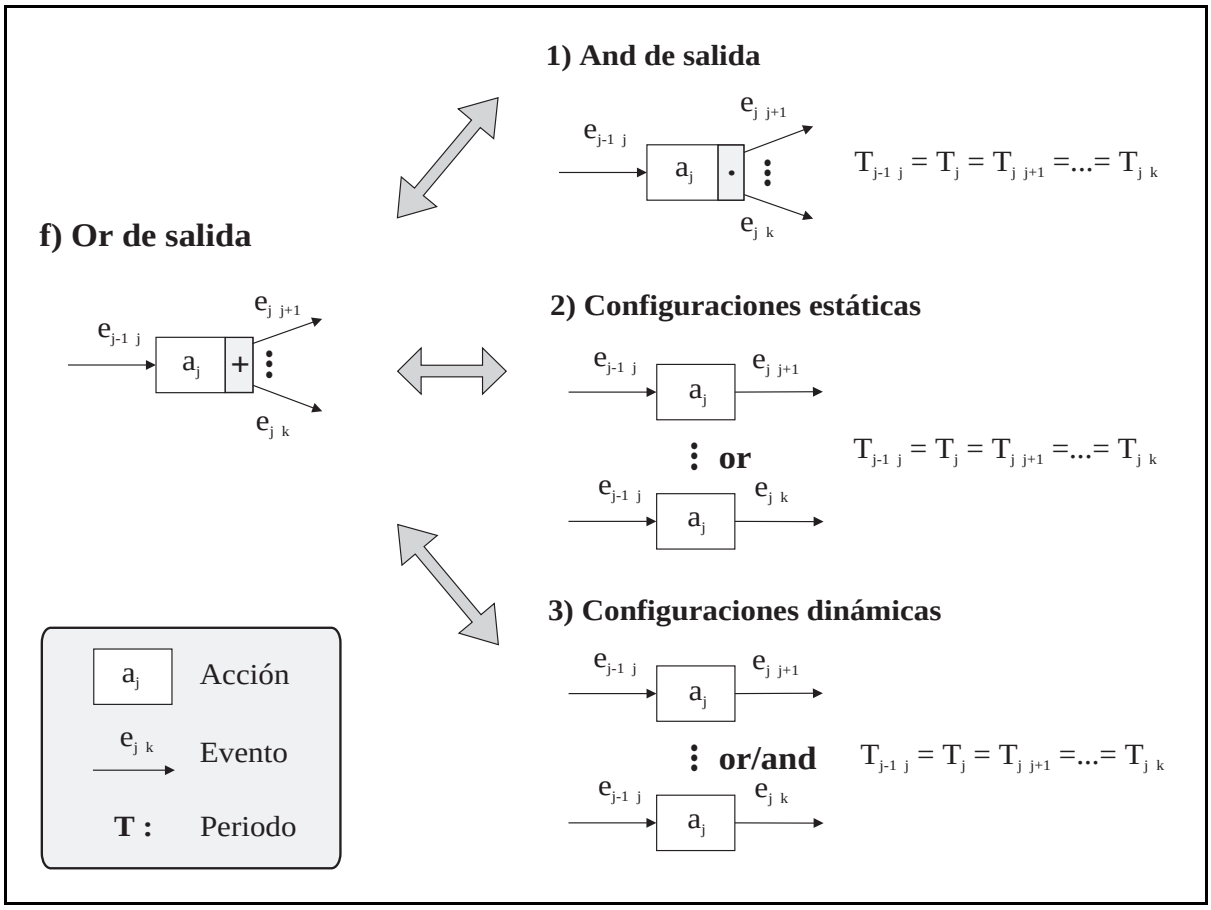

Figura 4-6. *Or de salida* en el *modelo equivalente* para el análisis.

dispara. El cambio que se produce en los períodos de las acciones posteriores al divisor en la secuencia de respuesta a un evento hace que estas acciones se activen únicamente con algunos de los disparos del evento, y por lo tanto, es necesario definir un criterio para establecer los plazos globales. Los plazos globales para las acciones posteriores al divisor se van a estimar de acuerdo con una suposición de peor caso, desde que se produce el primer evento hasta que finaliza la ejecución de la acción. Esta estimación de los plazos globales influirá en el cálculo de las respuestas globales de peor caso con las que se comparan estos plazos al realizar el análisis.

c) ***Divisor de frecuencia de salida***

Esta acción se activa y ejecuta con la llegada de un evento, pero únicamente cuando se haya ejecutado un número *N* de veces se va a generar el evento o los eventos de salida correspondientes. Por lo tanto, su período coincide con el del evento que la activa, y el de los eventos que puede generar resulta de multiplicar por *N* el

período del evento que la dispara. Los plazos globales se consideran de forma análoga al divisor de frecuencia de entrada.

d) ***Or de entrada***

En este caso la acción puede ser disparada indistintamente por cualquiera de los eventos de entrada. En el *modelo equivalente* para el análisis pretendemos separar las secuencias de respuesta a los diferentes eventos de entrada de manera que se obtenga una estructura lineal analizable. Para ello, duplicamos la secuencia de acciones y eventos un número de veces N a partir de la acción con *or de entrada* (N es el número de eventos de entrada), obteniendo respuestas (secuencias de acciones) independientes para cada evento.

En la duplicación de una acción, a_j da lugar al conjunto de acciones $a_{j_1}\ldots a_{j_N}$ en el que cada una pertenece a una de las secuencias de respuesta. Además, se cambia el nombre $e_{ij}\ldots e_{kj}$ de los eventos de entrada por el de $e_{ij_1}\ldots e_{kj_N}$, debido a la aparición de nuevas acciones. Las acciones duplicadas se localizan en el mismo recurso y se les asigna la misma prioridad que la acción de la que se derivan. Cada secuencia de acciones dependiente de uno de los eventos de entrada de la acción con *or de entrada* hereda el período del evento, y cada acción resultante de la duplicación hereda el plazo global o el plazo de principio-a-fin de la acción original para el evento a cuya secuencia de respuesta pertenece.

La duplicación de una cadena de acciones que implican a un único recurso y que no se suspenden a sí mismas, nos conduce a un *modelo equivalente* con idénticos tiempos de respuesta que el modelo original, si se utiliza una política de planificación por prioridad de tipo FIFO (en la que una tarea no puede expulsar a otra tarea de su misma prioridad). Esta política de planificación impediría que acciones duplicadas estuvieran simultáneamente activas, ya que las acciones duplicadas tienen todas la misma prioridad. Sin embargo, si la acción duplicada se suspende a sí misma o afecta a más de un recurso, es posible que acciones que en el *modelo equivalente* se ejecutan de forma simultánea, no pudieran en el modelo original ser simultáneas, porque son secuenciales. Ello provoca que el *modelo equivalente* para estos casos sea ligeramente pesimista, debido a que al realizar el análisis, la posible simultaneidad de las acciones provoca tiempos de respuesta más altos.

e) ***And de entrada***

El caso de una acción que se ejecuta únicamente en el momento en el que han llegado todos los eventos de entrada (con idénticos períodos) se refleja en el *modelo equivalente* haciendo que el período de la propia acción y de las acciones y eventos posteriores se corresponda con el de los eventos de entrada. Para obtener la situación de peor caso en la respuesta de las acciones con este tipo de patrón de llegada de eventos, será necesario tener en cuenta el máximo desfase (ver página 4-22) que pueda existir entre los eventos externos que dan lugar a los eventos de entrada que disparan a la acción con *and de entrada*. Por lo tanto, este patrón de llegada de eventos hace necesaria la incorporación al análisis de la información correspondiente a los máximos desfases existentes entre cada uno de los eventos externos que pueden confluir en una *and de entrada*.

f) ***Or de salida***

En el caso en el que una acción puede generar distintos eventos de salida, pero genera solamente uno en cada período vamos a considerar tres situaciones en el *modelo equivalente* (Figura 4-6) que dan lugar a diferentes modos de análisis dependiendo de los períodos, plazos y tiempos de respuesta globales de peor caso de las acciones. A continuación vemos en detalle estas situaciones:

1) ***And de salida***

Siempre podemos considerar como una situación de peor caso que en lugar de dispararse uno solo de los eventos de salida de la acción con *or de salida* se disparen todos, ya que esto significa la activación de todas las acciones de la respuesta y no solamente de unas cuantas. Por lo tanto, podemos tratar la acción con *or de salida* como una acción con *and de salida*. Los períodos de las acciones y eventos posteriores son iguales al período de la acción con *or de salida*. El análisis en este caso va a ser un tanto pesimista, por lo que seguidamente vamos a considerar dos situaciones especiales (usuales en aplicaciones distribuidas de tiempo real estricto) en las que se podrá eliminar en cierta medida este pesimismo.

2) *Configuraciones estáticas*

Para aquellos casos en los que todas las acciones posteriores (en la secuencia de respuesta a un evento) a la *or de salida* tengan plazos globales menores o iguales que los períodos ($D_{ij} \leq T_j$) podemos considerar la existencia de diferentes *configuraciones* del sistema (en principio una por cada evento que puede producir la acción con *or de salida*).

En cada configuración se considera que están activas las acciones posteriores a uno solo de los eventos generados por la acción con *or de salida*, y que no están activas las acciones posteriores correspondientes al resto de los eventos que podría haber generado esa acción. Desde el punto de vista del análisis, habrá que tener en cuenta únicamente aquellas acciones que estén activas. Los períodos de las acciones y eventos posteriores son iguales al período de la tarea con *or de salida*.

La consideración de plazos globales menores o iguales que los períodos nos garantiza la división del modelo en configuraciones para el análisis, ya que sería imposible que dos secuencias de respuesta correspondientes a dos eventos diferentes generados por la acción con *or de salida* estén simultáneamente activas y cumplan a la vez sus plazos globales.

El número de configuraciones que posee un sistema depende de las acciones con *or de salida* (bajo el supuesto $D_{ij} \leq T_j$ para las acciones de las respuestas a los eventos de salida), de su anidamiento y de su aparición en secuencias de respuesta a diferentes eventos externos. En realidad es el número de alternativas posibles que se pueden tomar en las respuestas a los eventos externos como consecuencia de las acciones con *or de salida* que tenga el sistema.

Las configuraciones que estamos considerando diremos que son *estáticas* porque se pueden determinar previamente a la realización del análisis. Como veremos más adelante, el análisis de esta situación reduce el pesimismo al considerar fuera del análisis aquellas acciones que no están activas.

3) *Configuraciones dinámicas*

En el caso anterior hemos visto que bajo ciertas condiciones ($D_{ij} \leq T_j$ para las acciones de las respuestas a los eventos de salida) podemos identificar en el modelo un cierto número de configuraciones estáticas para el análisis. Si en estas condiciones cambiamos los plazos globales por las respuestas globales de peor caso de las acciones, es decir, $R_{ij} \leq T_j < D_{ij}$ para las acciones de las respuestas a los eventos de salida, podríamos seguir aplicando el mismo tipo de análisis que para el caso de configuraciones estáticas (puesto que se garantiza la imposibilidad de que dos secuencias de respuesta correspondientes a dos eventos diferentes generados por la acción con *or de salida* estén simultáneamente activas y cumplan a la vez sus plazos globales).

Ahora sin embargo, el cumplimiento de las condiciones que identifican a las configuraciones se debe realizar durante el análisis, ya que es necesario conocer los tiempos de respuesta globales de peor caso. Por lo tanto, a partir de una situación inicial en la que se considera que todas las *or de salida* dan lugar a configuraciones, se aplicará un método iterativo de análisis (para conocer las respuestas) e identificación de configuraciones (si para una *or de salida* no se verifica que $R_{ij} \leq T_{j,}$ se sustituye por una *and de salida*), hasta que se alcance una solución estable. A estás configuraciones las vamos a denominar *dinámicas*, porque se identifican dinámicamente en el análisis del sistema (el número de configuraciones dinámicas no se puede determinar a priori).

Los períodos de las acciones y eventos posteriores son iguales al período de la tarea con *or de salida*. De la misma forma que las configuraciones estáticas, la identificación dinámica de configuraciones también reduce el pesimismo en el análisis.

En un mismo sistema es posible la coexistencia de configuraciones estáticas (identificadas antes de realizar el análisis) y dinámicas (no se pueden identificar a priori, y aparecen en la realización del análisis).

g) ***And de salida***

Para el caso en el que una acción dispara simultáneamente varios eventos consideramos que tanto las acciones como los eventos posteriores a la acción con *and de salida* tienen los mismos períodos que ésta.

Al obtener el *modelo equivalente* para una operación *or de salida* con restricciones, como en el ejemplo del sistema que utiliza una aproximación cliente-servidor mostrado en la Figura 4-4 (ver página 4-9), podemos hacerlo de dos maneras diferentes:

- *A partir del modelo general con el que representamos la aproximación cliente-servidor*. Teniendo en cuenta la restricción impuesta sobre los eventos de salida, la duplicación de acciones para este caso debida a las *or de entrada*, implica únicamente la duplicación de los servidores y no de los clientes. Además, en esta duplicación hay que realizar correctamente la conexión entre los eventos de salida y los servidores duplicados, de manera que la acción de solicitud de servicio y la posterior al servicio correspondan al mismo cliente. En la Figura 4-7 se muestra cómo se obtiene el *modelo equivalente* para el ejemplo de la Figura 4-4 (ver página 4-9).

- *Directamente del modelo cliente-servidor*. A partir del modelo cliente-servidor podemos llegar al mismo *modelo equivalente* mediante la duplicación de los servidores para los clientes que requieren sus servicios.

En cualquier caso, desde el punto de vista del análisis, la duplicación de servidores puede ser (ver *or de entrada* en la página 4-13):

- *Equivalente*. Si el servidor se puede representar por una acción que no sufre suspensiones.

- *Pesimista*. Si el servidor se representa mediante una acción que sufre suspensiones en su ejecución. Un ejemplo de este caso es un servidor que requiere el servicio de otro servidor.

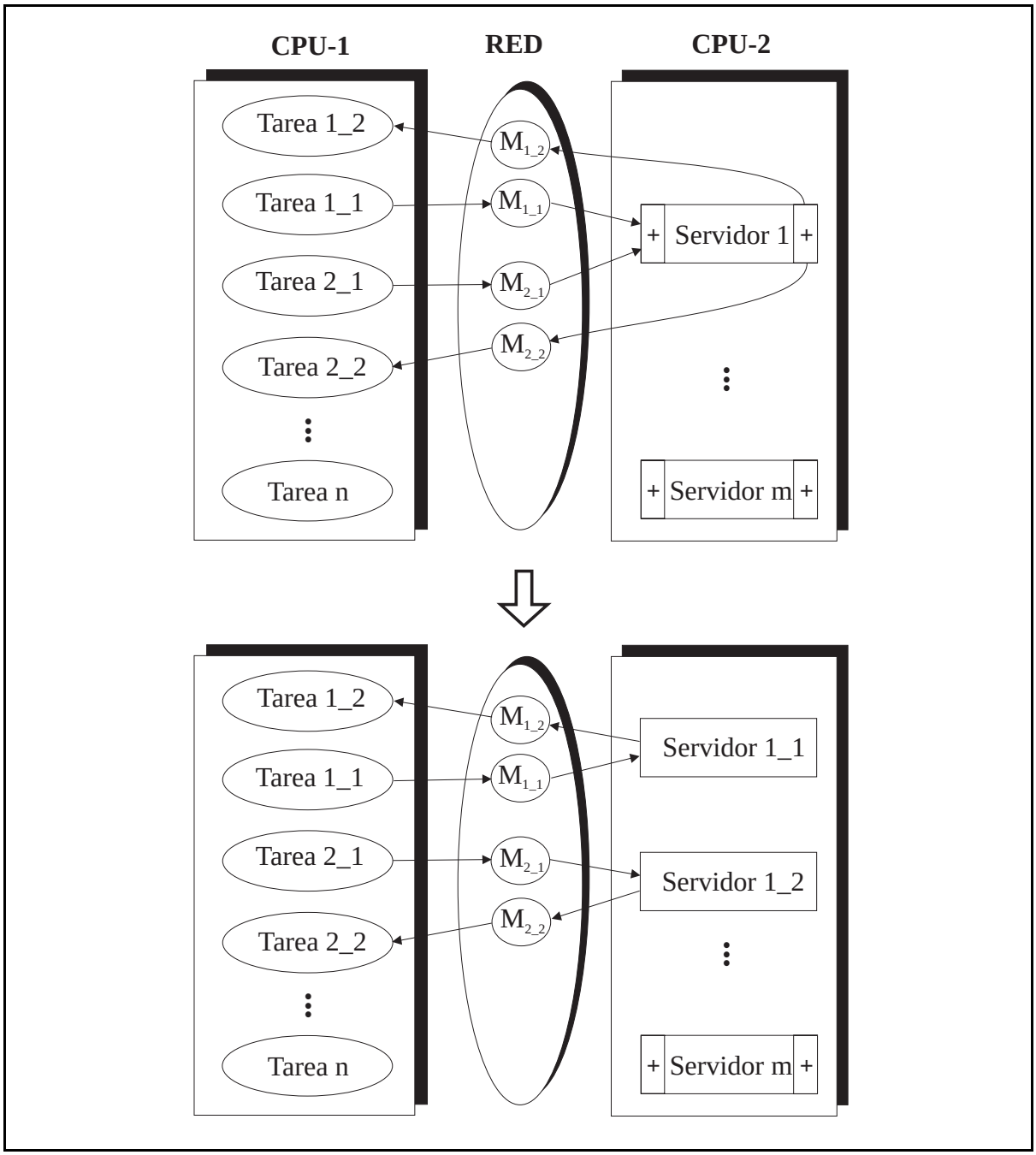

Figura 4-7. Modelo equivalente para una aproximación cliente-servidor.

El *modelo equivalente* hace posible el análisis de tiempo real de los sistemas cuyas acciones responden a los patrones de llegada y generación de eventos definidos en el *modelo general* de sistema distribuido. A continuación vamos a definir el análisis aplicable sobre este *modelo equivalente*.

4.3.2. Análisis de tiempo real para el modelo general

A partir del *modelo general* de sistema distribuido y para poder realizar el análisis, extraemos el *modelo equivalente* de la forma descrita en el apartado anterior, tanto en lo referente a la duplicación de acciones y eventos, como al establecimiento (si procede) de las distintas configuraciones estáticas (las dinámicas van apareciendo en la propia ejecución del análisis).

Para comenzar con la definición del análisis de tiempo real aplicable al *modelo equivalente*, vamos a ver por un lado cuáles son los problemas (íntimamente ligados a la estructura del modelo) que se plantean en esta definición a la hora de calcular los tiempos de respuesta para el peor caso y determinar la planificabilidad de un sistema, y por otro lado, cómo se pueden afrontar dichos problemas. En esta problemática nos vamos a centrar en los siguientes puntos:

- *Análisis de las configuraciones del modelo*. Las configuraciones aparecen como una división del sistema para el análisis motivada por las acciones con *or de salida*. Por consiguiente, en el caso de que se puedan identificar configuraciones en el sistema (estáticas, dinámicas o ambas), la técnica de análisis para el sistema completo se basará en la aplicación del análisis por separado a cada configuración del *modelo equivalente*, ya que estas configuraciones se pueden considerar como sistemas independientes. Diremos que el sistema es planificable cuando lo sean todas y cada una de las configuraciones. Por lo tanto, es necesario definir el análisis para cada configuración por separado de manera que se establezca así su planificabilidad. El análisis para cada configuración se realiza únicamente sobre aquellas acciones que pueden ser activadas en esa configuración de acuerdo con el criterio mostrado en el apartado anterior (*or de salida* en el *modelo equivalente*). Cuando el sistema no se pueda descomponer en configuraciones, el análisis se realizará considerando que todas las acciones están activas (como si únicamente existiera una configuración). Desde el punto de vista del *modelo equivalente*, el *modelo lineal* solamente tiene una configuración en la que todas las acciones están activas. El análisis de una configuración se realiza como el de un sistema completo en el que no hay acciones con *or*

de salida. Esto constituye el resto de la problemática que se plantea para el análisis, y en ella entraremos seguidamente.

- *Análisis de los divisores de frecuencia*. La existencia de un divisor de frecuencia en la respuesta a un evento no sólo debe ser tenida en cuenta en la obtención de los períodos de las acciones posteriores al divisor en la secuencia de respuesta al evento, sino que también se debe considerar en el cálculo de las respuestas globales de peor caso de estas acciones. Si consideramos un divisor de frecuencia con período T y constante N (veces que se debe activar para generar el evento o eventos de salida), las acciones posteriores al divisor en la secuencia de respuesta al evento nunca van a ejecutarse hasta después de producirse la *N-ésima* activación del divisor de frecuencia. Por consiguiente, en los tiempos de respuesta globales de peor caso de estas acciones posteriores debemos tener en cuenta un término adicional igual a $(N-1)T$, que representa una espera de $N-1$ períodos.

- *Análisis de los patrones and de entrada*. Los patrones de llegada de eventos de este tipo dan lugar a cruces en las secuencias de respuesta a diferentes eventos. Estos cruces nos conducen inexorablemente a la incorporación al análisis de ciertas medidas de la sincronización para hacer válido el cálculo de las respuestas globales de peor caso. Aún cuando todos los eventos de entrada a una acción con *and de entrada* tienen el mismo período, hay que tener en cuenta que la acción se activará con la llegada del evento más lento (el último en llegar). Si suponemos que todos los eventos externos que dan lugar a eventos de entrada a una acción con *and de entrada* se disparan a la vez, el tiempo de respuesta global de peor caso para esta acción sería sencillamente el mayor de los tiempos de respuesta globales de la acción para cada evento externo (suma de los tiempos de respuesta locales de peor caso de las acciones precedentes en la respuesta al evento, si estas acciones fueran lineales). Sin embargo, en los sistemas reales no siempre podremos asegurar que el disparo de los eventos sea simultáneo, sino que lo normal será que exista un cierto desfase entre ellos. Por lo tanto, para asegurar la creación de una situación de peor caso será necesario tener en cuenta los máximos desfases (ver página 4-22) entre los eventos externos en el cálculo de los tiempos de respuesta globales. Idéntico tratamiento se debe aplicar a las

acciones posteriores a una acción con *and de entrada*, ya que de igual manera formarán parte de la respuesta a varios eventos externos.

- *Análisis lineal*. Los casos que no hemos considerado hasta ahora (*lineal*, *or de entrada* y *and de salida*) dan lugar a un análisis de tipo lineal similar al utilizado para el *modelo lineal*.

Teniendo en cuenta la problemática junto con las soluciones que acabamos de exponer, vamos a ver ahora cómo se calculan las respuestas globales de peor caso para cada configuración o para el sistema completo en el caso de que únicamente exista una configuración. De esta manera, el análisis general en cualquier caso se reduce a establecer las condiciones de planificabilidad para las configuraciones. Decimos que una configuración es planificable cuando se verifican los plazos globales asignados a las acciones que están activas, es decir, las respuestas globales de peor caso de las acciones que se pueden disparar en esa configuración son menores o iguales que sus plazos globales.

Para cada evento externo e_i definimos su período T_i como el inverso de la frecuencia de peor caso con la que el evento puede llegar al sistema. De igual manera definimos el período T_{jk} correspondiente al evento interno e_{jk}. También definimos la fase ϕ_i de cada evento externo e_i como la distancia entre el tiempo cero (instante arbitrario en el que se considera el origen de tiempos), y el instante en el que se produce el primer evento. En muchas ocasiones no se conoce la fase, pero es posible determinar una cota superior para el máximo desfase entre distintos eventos que como veremos, será el único factor que tiene influencia en el análisis.

A partir del *modelo general* (y de su transformación en el *modelo equivalente*) y para cada acción a_j en la respuesta a un evento externo e_i, definimos el tiempo de respuesta global de peor caso R_{ij} (relativo a la llegada del evento e_i) como el máximo entre las sumas de las respuestas locales de la propia acción y de todas las anteriores para cada evento a cuya respuesta pertenece a_j (teniendo en cuenta los desfases entre los eventos):

$$R_{ij} = \max_{l \in E_j} \left(\Phi_{il} + \sum_{k \in A_{ij}} r_{lk} + (N_f - 1) T_{f-1f} \right)$$

donde,

r_{lk} Es el tiempo de respuesta local de la acción a_k (relativo a la activación de a_k) para el evento e_l.

Φ_{il} Es el máximo desfase entre los eventos e_i y e_l ($\Phi_{il} = 0$, si $i = l$). El máximo desfase Φ_{il} entre los eventos externos e_i y e_l se estima como la máxima diferencia en valor absoluto entre las fases (es igual a Φ_{li} entre e_l y e_i):

$$\Phi_{il} = \max \left| \phi_i - \phi_l \right|$$

El máximo desfase para eventos periódicos que se generan de forma continua es igual al período.

A_{lj} Es el conjunto de acciones de la secuencia de respuesta al evento externo e_l previas a la acción a_j (incluida ella misma).

E_j Es el conjunto de eventos externos a cuya respuesta pertenece la acción a_j.

N_f Es la constante del divisor de frecuencia más cercano (a partir de los eventos externos) a la acción a_j o la suya si a_j es un divisor de frecuencia (N_f vale 1 si no hay divisor de frecuencia).

$T_{f\text{-}1f}$ Es el período del evento de entrada del divisor de frecuencia al que corresponde la constante N_f (si fuera externo sería T_f).

En esta definición para el tiempo de respuesta global de peor caso están contemplados todos los tipos de acciones representados en el *modelo general* de sistema distribuido. De acuerdo con la problemática que habíamos expuesto a la hora de afrontar el análisis, podemos establecer las siguientes relaciones entre la definición dada para el tiempo de respuesta global de peor caso y los puntos de la problemática que hacen referencia a éste:

- El análisis de los divisores de frecuencia está representado por el término $(N_f\text{-}1)T_{f\text{-}1\,f}$.

- El análisis de los patrones and de entrada se refleja en la aparición del máximo desfase (Φ_{il}) y en la elección del máximo tiempo de respuesta global para el conjunto E_j.

- El análisis lineal se plasma en el término $\sum_{k \in A_{lj}} r_{lk}$.

Para el cálculo de las respuestas locales aplicamos la teoría *RMA* a cada recurso por separado tal como se describe en el capítulo 2 (ver apartado 2.3). De esta forma se pueden obtener los distintos tiempos de respuesta locales (r_{ij}) para cada acción (a_j) y para cada evento externo (e_i) de cuya respuesta forma parte la acción (a_j), siempre para valores del jitter ya determinados. De la misma forma que en el *modelo lineal*, se aplica un análisis iterativo en el que se van calculando de manera sucesiva los tiempos de respuesta y los nuevos términos de jitter asociados [TIN94F] (el tiempo de respuesta depende del jitter y éste a su vez del tiempo de respuesta).

Por último, se determina la planificabilidad de una configuración del sistema (o del sistema completo si no se han identificado configuraciones estáticas o dinámicas) mediante la comparación de los tiempos de respuesta globales (R_{ij}) con los plazos globales (D_{ij}) de las acciones activas en esa configuración. También aquí vamos a llamar ER_{ij} al tiempo de respuesta global de peor caso de la acción a_j para el evento e_i cuando es la última acción de la respuesta.

4.4. Algoritmos de planificación para el modelo general

En este apartado describiremos cómo se realiza la asignación de prioridades al *modelo general* de sistema distribuido. Más concretamente, extenderemos las técnicas de planificación que vimos en el capítulo 3, es decir, el algoritmo heurístico para la optimización de la asignación de prioridades (*HOPA*), y el algoritmo basado en la técnica del *templado simulado*.

En el marco de la metodología de análisis y diseño de sistemas distribuidos de tiempo real que estamos desarrollando, y antes de pasar a la descripción de la extensión de los algoritmos, vamos a dar una serie de pautas generales para la aplicación de cualquier algoritmo de asignación de prioridades sobre el *modelo general* de sistema distribuido que hemos definido.

Todo algoritmo para la asignación de prioridades que opere sobre el *modelo general* de sistema distribuido debe tener en cuenta los siguientes aspectos:

- Las técnicas de análisis de tiempo real que se deben utilizar son las descritas en el apartado anterior.

- El primer paso que hay que dar para realizar la planificación del sistema completo es la identificación de las configuraciones estáticas de que se compone el sistema.

- En el caso de que se hayan identificado configuraciones estáticas en el sistema, es preciso tener en cuenta que se debe mantener la coherencia en la asignación de prioridades a las diferentes configuraciones, es decir, cuando se encuentre una solución a la asignación de prioridades, cada una de las acciones presentes en más de una configuración deberán lógicamente tener la misma prioridad asignada. Para conseguir que la solución final cumpla esta propiedad, aplicaremos un algoritmo que se describe a continuación.

  El algoritmo se inicia procediendo a la *asignación inicial* de prioridades a cada una de las configuraciones por separado, utilizando para ello el algoritmo pertinente de asignación de prioridades. En esta asignación inicial no se tienen en cuenta las acciones que no estén activas en la configuración correspondiente. Si existe una configuración para la que no se encuentra una asignación inicial de prioridades con la que sea planificable, concluiremos que el sistema no es planificable (si no es planificable parcialmente, tampoco lo va a ser globalmente). En la asignación inicial de prioridades a cada configuración se debe tener en cuenta la posible identificación de configuraciones dinámicas únicamente a la hora de realizar el análisis. La inclusión de las configuraciones dinámicas en la planificación sería muy compleja y de resultados impredecibles, ya que podríamos obtener una asignación de prioridades a partir de una configuración dinámica que no existe para esa asignación (posible lazo infinito entre asignación de prioridades e identificación de configuraciones dinámicas).

- Para la asignación de prioridades al sistema completo, vamos a recordar el concepto de *prioridad preasignada* introducido en el capítulo 3, y que en este caso tiene una utilización plena. La *prioridad preasignada* es una prioridad dada a priori a una acción del sistema relativa a otras acciones con prioridad preasignada dentro de un recurso. En la planificación se tratará de asignar prioridades a las acciones preservando el orden relativo a las prioridades preasignadas en cada recurso del sistema.

Las asignaciones de prioridades que hacen que el sistema completo sea planificable, se obtienen a partir de las asignaciones iniciales de prioridades de la manera que se muestra a continuación. Para cada configuración estática:

1.- Se toman las prioridades correspondientes a su asignación inicial como prioridades preasignadas.

2.- Para cada una de las demás configuraciones estáticas:
Se aplica el algoritmo de asignación de prioridades, pero tomando como prioridades preasignadas las prioridades obtenidas en la asignación de la configuración anterior. Al igual que para las asignaciones iniciales, las configuraciones dinámicas solamente se deben tener en cuenta en el análisis.

Aplicando este procedimiento, por cada configuración estática se puede obtener una asignación de prioridades que haga al sistema planificable. Basta con hallar una de estas asignaciones para determinar que el sistema es planificable.

- Hay que poner especial cuidado al asignar las prioridades a las acciones duplicadas. Debemos tener en cuenta que todas las acciones duplicadas en el *modelo equivalente* deben llevar la misma prioridad que la acción de la que proceden.

Para la elección de la mejor asignación de prioridades definimos el *índice de planificación global* (*GInd.*) como la suma de los índices de planificación obtenidos en el análisis de cada configuración por separado para conseguir esa asignación de prioridades, en el caso de que la asignación haga que el sistema sea planificable (en otro caso, le damos un valor fijo ya que no se ha realizado la asignación completa de prioridades). El índice de planificación de cada configuración se calcula de acuerdo con la definición dada en el capítulo 3 (ver apartado 3.2) para las acciones que estén activas en esa configuración. Así pues, podemos expresar el índice de planificación global de la manera siguiente:

$$GInd = \begin{cases} -1 & ; \quad Si\ no\ es\ planificable \\ \sum_{i \in 1..Nconf} Ind_i & ; \quad Si\ es\ planificable \end{cases}$$

donde,

Nconf Es el número de configuraciones (estáticas y dinámicas) identificadas en el sistema para conseguir la asignación de prioridades.

Ind_i Es el índice de planificación de la configuración i.

El índice global de planificación así definido nos permite cuantificar aspectos meramente cualitativos de la realización del análisis del sistema. Por ejemplo, vamos a suponer que tenemos un sistema en el que podemos identificar dos configuraciones estáticas (a partir de una acción con *or de salida*), y además suponemos que en la realización del análisis identificamos otra acción con *or de salida* que da lugar a configuraciones dinámicas (el número total de configuraciones es cuatro). Podríamos analizar este sistema de tres maneras diferentes, que darían lugar a índices globales de planificación distintos, y que son las siguientes:

- Consideramos las acciones con *or de salida* como *and de salida*. Obtendríamos el índice global de planificación con una sola configuración que constituye el sistema completo.

- Consideramos únicamente las configuraciones estáticas. Obtendríamos el índice global de planificación con dos configuraciones. Este índice global de planificación será potencialmente mayor que el obtenido en el caso anterior por ser el resultado de la suma de dos índices de planificación.

- Consideramos las configuraciones estáticas y las dinámicas. Obtendríamos el índice global de planificación con cuatro configuraciones. El índice global de planificación obtenido será potencialmente mayor los de los casos anteriores por ser el resultado de la suma de cuatro índices de planificación.

Por lo tanto, este índice global de planificación está proclamando como mejores soluciones para la asignación de prioridades en un sistema (mayores índices) aquellas que se obtienen con la identificación de un mayor número de configuraciones. Esto viene a representar de una manera exacta el significado que tiene la identificación de configuraciones en el sistema, sobre todo en lo referente a la reducción del pesimismo en el análisis, y como veremos a continuación, a la obtención de menores utilizaciones de los recursos.

Un último aspecto que debemos considerar es el cálculo de las utilizaciones de los recursos en los sistemas que responden al modelo de sistema distribuido que estamos tratando. La utilización de un recurso se define como la utilización correspondiente a la configuración con mayor carga para ese recurso. De esta forma, podemos expresar la utilización para el recurso procesador o de comunicaciones PR_i de la manera siguiente:

$$U(PR_i) = \max_{j \in Nconf} \left[U(PR_{ij}) \right]$$

donde, $U(PR_{ij})$ es la utilización del recurso PR_i para la configuración j del sistema. De esta manera la existencia de acciones no activas en las configuraciones da lugar a menores utilizaciones de los recursos.

4.4.1. Algoritmo heurístico

La adaptación del algoritmo *HOPA* debe responder a las pautas que hemos marcado para los algoritmos de planificación que actúan sobre el *modelo general*. Así pues, es necesario realizar la asignación de prioridades para todas las configuraciones estáticas de la manera que hemos explicado, por lo que se deben hacer algunos cambios sobre todo en lo que se refiere al cálculo de los plazos locales en los que se basa este algoritmo. Estos cambios en el cálculo de los plazos locales pretenden recoger la existencia de acciones que pertenecen a las respuestas a varios eventos, la existencia de prioridades preasignadas (en este caso como parte fundamental del algoritmo), y la existencia de acciones duplicadas (que deben tener la misma prioridad, y por lo tanto el mismo plazo local).

El algoritmo *HOPA extendido* se basa en el algoritmo *HOPA* (ver apartado 3.3 del capítulo 3) para el cálculo de las prioridades de cada configuración (obviamente con los cambios pertinentes en el cálculo de los plazos locales artificiales que se asignan a cada acción). Si consideramos la identificación de configuraciones estáticas en el sistema (las configuraciones dinámicas solamente se utilizan en el análisis), este algoritmo se puede expresar de la forma en que se muestra a continuación:

```
algoritmo HOPA extendido is
begin

      --    Asignación inicial de prioridades a cada configuración
            estática.
```

```
        for i in 1..número de configuraciones estáticas
        loop
              HOPA;
              if i no es planificable then termina el algoritmo;
        end loop;

        --    Asignación de prioridades al sistema completo.

        for i in 1..número de configuraciones estáticas
        loop
              Establece prioridades preasignadas con la asignación
              inicial de prioridades de la configuración i;
              for j in número de configuraciones estáticas (j ≠ i)
              loop
                    HOPA;
                    exit when j no es planificable;
                    Establece prioridades preasignadas con la última
                    asignación de prioridades realizada;
              end loop;
        end loop;
  end HOPA extendido;
```

La única modificación que debemos hacer en el algoritmo *HOPA* en su aplicación al algoritmo *HOPA extendido* radica en el cálculo de los plazos locales. Vamos a ver ahora cómo se realiza esta modificación, y para ello, suponemos que en principio los requerimientos de tiempo real impuestos son únicamente los plazos globales de principio-a-fin (plazos globales a las últimas acciones en las respuestas a los eventos).

Los plazos locales iniciales (d_{ij}) se pueden asignar de alguna manera que preserven los plazos globales de principio-a-fin, es decir, $ED_{ij} = \sum d_{ik}$ para todas las acciones a_k de la respuesta al evento e_i que concluye en la acción a_j). Una posible asignación es distribuir los plazos globales de forma proporcional al tiempo de ejecución de peor caso de cada acción (C_k). A partir de los plazos locales asignados d_{ij} se calculan los plazos d_j. Sin embargo, todavía no se pueden asignar las prioridades porque para el cálculo de los plazos d_j no hemos tenido en cuenta ni las prioridades preasignadas ni las acciones duplicadas. Como consecuencia, aplicamos un algoritmo de asignación de plazos locales forzados a cada recurso del sistema, que consta de los siguientes pasos:

1.- Se fuerzan los plazos locales de las acciones duplicadas. A todas las acciones que provengan de la duplicación de la misma acción se les debe asignar el mismo plazo local (porque deben llevar la misma prioridad).

2.- Se fuerzan los plazos locales de las acciones con prioridades preasignadas. Comenzando por la acción con menor prioridad preasignada, se va chequeando si el plazo local calculado para la siguiente (con mayor prioridad) tiene realmente un plazo más pequeño. Si no lo tiene se fuerza este plazo local a un valor inmediatamente por debajo que el de la acción con menor prioridad.

3.- Si se han forzado nuevos plazos locales en alguno de los pasos anteriores se ejecuta de nuevo el paso 1; en caso contrario se continúa.

4.- Se distribuye el plazo global sobrante entre las acciones con plazo local no forzado (acciones no duplicadas ni con prioridades preasignadas).

El cálculo de los nuevos plazos intermedios se realiza aplicando las mismas fórmulas (basadas en los excesos y en los plazos asignados con anterioridad) que se utilizaron en el capítulo 3 (apartado 3.3). En este caso, una vez obtenidos los nuevos plazos d_{ij} se calculan los plazos d_j y después se aplica el algoritmo de asignación de plazos locales forzados que acabamos de describir.

La aparición de varias posibilidades en la asignación de plazos locales para la misma acción, junto con la necesidad de forzar esta asignación de plazos debida a la existencia de acciones duplicadas y de prioridades preasignadas, lleva consigo una cierta carga de incertidumbre que puede hacer que la aplicación del algoritmo heurístico a la asignación de prioridades para el *modelo general* no obtenga tan buenos resultados como su aplicación para el *modelo lineal*. La aplicación de algoritmos como el *templado simulado* al problema de la asignación de prioridades al *modelo general* elimina esta incertidumbre.

4.4.2. *Templado simulado*

La extensión del algoritmo de asignación de prioridades basado en la técnica del *templado simulado* también debe responder a las pautas que hemos marcado para los algoritmos de planificación que actúan sobre el *modelo general*. Por lo tanto, podemos hacer una definición del algoritmo de *templado simulado extendido* similar a la que hicimos para el algoritmo *HOPA extendido*. En este caso nos apoyamos en el algoritmo de *templado*

simulado desarrollado para el *modelo lineal* de sistema distribuido (ver apartado 3.2 del capítulo 3). De la misma manera que para el algoritmo *HOPA extendido*, si consideramos la identificación de configuraciones estáticas en el sistema (las configuraciones dinámicas solamente se utilizan en el análisis), este algoritmo extendido se puede expresar de la forma en que se muestra a continuación:

```
algoritmo templado simulado extendido is
begin

      --    Asignación inicial de prioridades a cada configuración
            estática.

      for i in 1..número de configuraciones estáticas
      loop
            templado simulado;
            if i no es planificable then termina el algoritmo;
      end loop;

      --    Asignación de prioridades al sistema completo.

      for i in 1..número de configuraciones estáticas
      loop
            Establece prioridades preasignadas con la asignación
            inicial de prioridades de la configuración i;
            for j in número de configuraciones estáticas (j ≠ i)
            loop
                  templado simulado;
                  exit when j no es planificable;
                  Establece prioridades preasignadas con la última
                  asignación de prioridades realizada;
            end loop;
      end loop;
end templado simulado extendido;
```

En la aplicación del algoritmo de *templado simulado* desarrollado en el capítulo 3 debemos tener en cuenta las nuevas condiciones que aparecen en la asignación de prioridades al *modelo general*. Estas condiciones están focalizadas en la existencia de acciones duplicadas a las que se debe asignar la misma prioridad, y en la aparición de las prioridades preasignadas como parte fundamental del algoritmo de planificación (en el *modelo lineal* se presentó como una opción de la herramienta de planificación). Ambas condiciones se recogen en la redefinición de la *función vecino* (y como consecuencia del *espacio vecino*). Por consiguiente, la función vecino operará ahora de acuerdo con los siguientes pasos:

1.- Selecciona un recurso (procesador o red) de manera aleatoria.

2.- Selecciona dos acciones (tareas o mensajes) al azar del recurso elegido.

3.- Intercambia las prioridades de las acciones seleccionadas teniendo en cuenta los puntos siguientes:
   - El intercambio debe ser compatible con la preasignación de prioridades.
   - Si las acciones seleccionadas pertenecen a un grupo de acciones duplicadas se deben intercambiar también las prioridades de estas acciones.

   Si no se pueden intercambiar las prioridades se vuelve al paso 1.

Dadas las particulares características de la técnica del *templado simulado* (como técnica de optimización de propósito general) y las pautas marcadas para la definición de algoritmos de planificación sobre el *modelo general* de sistema distribuido, la extensión del algoritmo para su aplicación a este modelo resulta relativamente sencilla. Además, podemos prever que el comportamiento y prestaciones de este algoritmo en la asignación de prioridades al *modelo general* va a ser el mismo que para el *modelo lineal*, por supuesto salvando las diferencias inherentes al aumento de la complejidad del problema a resolver.

4.5. Aplicación de la planificación al modelo general

De igual manera que para el *modelo lineal* (ver capítulo 3), es necesario realizar ahora la implementación de las técnicas de análisis y planificación definidas en los apartados anteriores para el *modelo general* de sistema distribuido de tiempo real, con objeto de poder evaluar la operatividad de dichas técnicas.

Por lo tanto, vamos a proceder a la extensión de la herramienta de planificación desarrollada en el capítulo 3, para que incorpore el nuevo modelo de sistema distribuido junto con las técnicas de análisis y planificación que para él hemos desarrollado. Esta extensión de la herramienta de planificación debe ir acompañada por la extensión de la herramienta de generación automática de ejemplos para que sea operativa sobre el nuevo modelo de sistema distribuido. Dada la complejidad que representa la implementación completa de las extensiones realizadas, aplicamos algunas restricciones que hagan más sencilla la extensión de estas herramientas experimentales manteniendo la validez de las pruebas que con ellas se pretenden realizar. Así pues, no vamos a considerar los divisores de frecuencia en la

representación del sistema (únicamente implican un cambio de período sobre el caso lineal), y tampoco el análisis y la planificación con identificación de configuraciones dinámicas (la resolución de este caso es idéntica a la que se aplica sobre las configuraciones estáticas).

Finalmente, mostraremos un ejemplo de sistema distribuido que responda al *modelo general*, y sobre él vamos a aplicar las técnicas de análisis y planificación desarrolladas. En primer lugar, veremos cómo se obtiene el *modelo equivalente* para poder realizar el análisis de tiempo real, y después, comentaremos los resultados obtenidos de la aplicación de las herramientas de planificación y generación automática de ejemplos.

4.5.1. Extensión de las herramientas de planificación y generación de ejemplos

La extensión de la herramienta de planificación desarrollada en el capítulo 3 (ver apartado 3.4.1) consiste en la incorporación a esta herramienta de las técnicas de análisis y de asignación de prioridades definidas en los apartados anteriores de este capítulo para el *modelo general* de sistema distribuido. Para describir esta extensión nos centramos en los aspectos más importantes que caracterizan a esta herramienta y que han sido objeto de ampliación. Siguiendo el esquema utilizado para la descripción de la herramienta en el capítulo 3 (ver apartado 3.4.1), agrupamos los cambios realizados en los puntos que se relacionan a continuación:

1.- Descripción del sistema distribuido.

En este punto se incorporan los aspectos correspondientes al *modelo general* de sistema distribuido que son diferentes de los del *modelo lineal*. Estos aspectos hacen referencia a la conectividad de las tareas (ya que la conectividad de los mensajes no cambia debido a que continúan teniendo una sola tarea de entrada y una sola tarea de salida) y a la identificación de las diferentes configuraciones estáticas.

En la implementación que realizamos, solamente vamos a representar las operaciones múltiples tanto de entrada como de salida, pero no vamos a representar los divisores de frecuencia. Por lo tanto, en la descripción que hicimos del sistema habrá que ampliar la información correspondiente a la

conectividad de las tareas para incorporar las operaciones *or de entrada, or de salida, and de entrada* y *and de salida*.

Para la identificación y manipulación de las configuraciones incorporamos una serie de utilidades que son las siguientes:

- Procedimiento de duplicación de acciones a partir de las acciones del sistema que respondan al patrón *or de entrada*.
- Procedimiento de identificación de configuraciones estáticas a partir de las acciones del sistema que respondan al patrón *or de salida*. Aquí se deben chequear además las combinaciones de patrones que puedan ser incompatibles.
- Procedimiento de marcado de acciones activas para una configuración dada.

2.- Técnica de análisis.

Para la implementación de la técnica de análisis para el *modelo general* se utiliza íntegramente la técnica de análisis implementada para el *modelo lineal* de la manera en que se ha definido en el apartado 4.3 de este capítulo. Lo primero que hay que hacer para poder realizar el análisis sobre el *modelo general* es obtener el *modelo equivalente* a partir de la descripción del sistema. Para esto, se aplican las utilidades de identificación y manipulación de configuraciones que se facilitan junto con la descripción del sistema. Después, el análisis consiste solamente en pasar por las distintas configuraciones marcando las acciones que estén activas para esa configuración y realizando el cálculo de las respuestas globales de peor caso para que se puedan comparar con los plazos globales que hayamos impuesto.

3.- *Templado simulado extendido*.

La implementación del algoritmo de *templado simulado extendido* se corresponde con la descripción realizada en el apartado 4.4.2 de este capítulo, y las características particulares de la implementación del algoritmo de *templado simulado* que se utiliza, son las que ya comentamos en el capítulo 3 (ver apartado 3.4.1.3). El único cambio importante radica en la redefinición de la *función vecino* para que incorpore la existencia de prioridades forzadas

por las acciones duplicadas en combinación con el orden establecido por las prioridades preasignadas.

4.- Algoritmo *HOPA extendido*.

Al igual que para el *templado simulado extendido*, la implementación del algoritmo *HOPA extendido* se corresponde con la descripción realizada en el apartado 4.4.1 de este capítulo, y las características específicas de la implementación del algoritmo *HOPA* que se utiliza, son las que ya comentamos en el capítulo 3 (ver apartado 3.4.1.4). Es necesario redefinir tanto la asignación de los plazos locales iniciales, como el cálculo de los nuevos plazos locales para tener en cuenta el efecto de las acciones duplicadas junto con las prioridades preasignadas.

5.- Presentación de resultados.

La información que se presenta como resultado del análisis y de la planificación aplicados al *modelo general* aumenta y cambia sensiblemente debido a la posible aparición de configuraciones. A continuación damos una relación de los aspectos más destacables que se presentan como resultado de la planificación:

- Asignaciones de prioridades que hacen que el sistema sea planificable (indicando la configuración de partida para la asignación).
- Indice global de planificación para cada asignación de prioridades que haga al sistema planificable. El índice global de planificación nos muestra cuál es la mejor asignación de prioridades encontrada.
- Resultados correspondientes al análisis para las asignaciones de prioridades que se presentan como solución. Estos resultados consisten en una relación de respuestas globales de peor caso para todas las acciones que constituyen el sistema acompañadas de los plazos globales que deben cumplir.
- Porcentaje de utilización de cada recurso, y porcentaje promedio de utilización de todos los recursos. Este cálculo se realiza de acuerdo con la definición de la utilización de un recurso dada en la página 4-27.

- Medidas de los tiempo de CPU consumidos por los algoritmos de planificación empleados en la obtención de las soluciones. Se miden los tiempos parciales para cada configuración y el tiempo total.

La extensión de la herramienta de generación automática de ejemplos pasa por su adaptación al nuevo modelo de sistema distribuido que se representa. Su operación en los distintos modos (*variación de períodos, variación de carga simple* y *variación de carga múltiple*) se ve afectada por la existencia de acciones duplicadas. Los cambios en la carga de un recurso no dependen ahora solamente de la acción seleccionada, sino también de las posibles acciones duplicadas que tengan relación con ella. La única modificación que realizamos se centra en mantener la coherencia en los tiempos de ejecución de peor caso de acciones duplicadas. Así pues, en los modos de variación de carga se podrán producir cambios en las utilizaciones de los recursos con porcentajes diferentes a los especificados, y en el modo de variación de períodos es posible que se produzcan cambios en las utilizaciones de los recursos (para el *modelo lineal* se seguirán manteniendo constantes).

4.5.2. Planificación de un sistema distribuido de tiempo real estricto

En este apartado vamos a aplicar las herramientas de planificación y generación de ejemplos que hemos desarrollado al análisis y planificación de un ejemplo de sistema distribuido de tiempo real estricto que responde a la estructura del *modelo general* definido. A través de este ejemplo vamos a mostrar cómo se lleva a cabo el análisis con la obtención previa del *modelo equivalente* a partir de la especificación inicial del problema. Además, este ejemplo nos permitirá realizar la evaluación de los algoritmos de planificación que utilizan esta técnica de análisis sobre el *modelo general*.

El ejemplo que vamos a utilizar corresponde a un sistema distribuido constituido por cinco procesadores y dos redes de comunicación (ver Figura 4-8). Como en el ejemplo presentado en el capítulo 3 (ver apartado 3.5) suponemos que cualquier procesador puede enviar mensajes a cualquier otro procesador a través de cualquiera de las redes de comunicación. El sistema debe responder a cinco secuencias de eventos a partir de las cuales se generan las respuestas, que se componen de tareas y mensajes enlazados de la forma que se muestra en la Figura 4-9. En la estructura correspondiente a las respuestas a los eventos

(Figura 4-9), podemos observar que aparecen varias tareas que responden a los patrones definidos en el *modelo general*.

Continuando con la descripción del sistema, definimos los requerimientos temporales que se imponen sobre los eventos del sistema de la manera siguiente:

e_1 -->	$T_1 = 2000$	$D_{1\ 7} = 8000$		
e_8 -->	$T_8 = 3000$	$D_{8\ 7} = 12000$		
e_{12} -->	$T_{12} = 5000$	$D_{12\ 18} = 5000$	$D_{12\ 20} = 5000$	
		$D_{12\ 24} = 5000$	$D_{12\ 26} = 5000$	
e_{27} -->	$T_{27} = 2500$	$D_{27\ 33} = 2500$	$D_{27\ 35} = 2500$	
e_{36} -->	$T_{36} = 1000$	$D_{36\ 42} = 4000$		

Seguidamente, vamos a detallar cómo se obtiene el *modelo equivalente* a partir de la descripción de un sistema que responde al *modelo general* de sistema distribuido (operación previa a la realización del análisis y de la planificación).

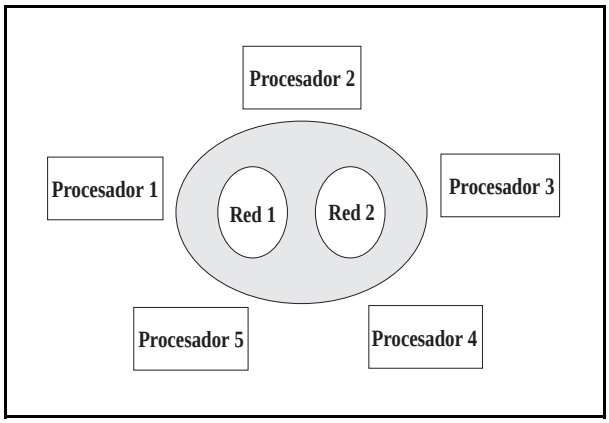

Figura 4-8. Sistema distribuido de tiempo real.

En primer lugar, se procede a la identificación de las acciones que presentan un patrón *or de entrada* para llevar a cabo la duplicación de acciones según el procedimiento descrito en la definición del *modelo equivalente* (ver página 4-13). La Figura 4-10 muestra la estructura de las secuencias de respuesta a los eventos externos después de la duplicación de las acciones.

El siguiente paso que hay que dar, previo a la realización del análisis, es la identificación de las configuraciones estáticas a partir de las acciones que respondan a patrones *or de salida*. En la Figura 4-11 se muestran las acciones que se encuentran activas en cada una de las seis configuraciones estáticas que se pueden identificar en nuestro ejemplo (todas las acciones con *or de salida* dan lugar a configuraciones estáticas). Sobre esta estructura modificada que hemos obtenido para el sistema (*modelo equivalente*) ya se pueden

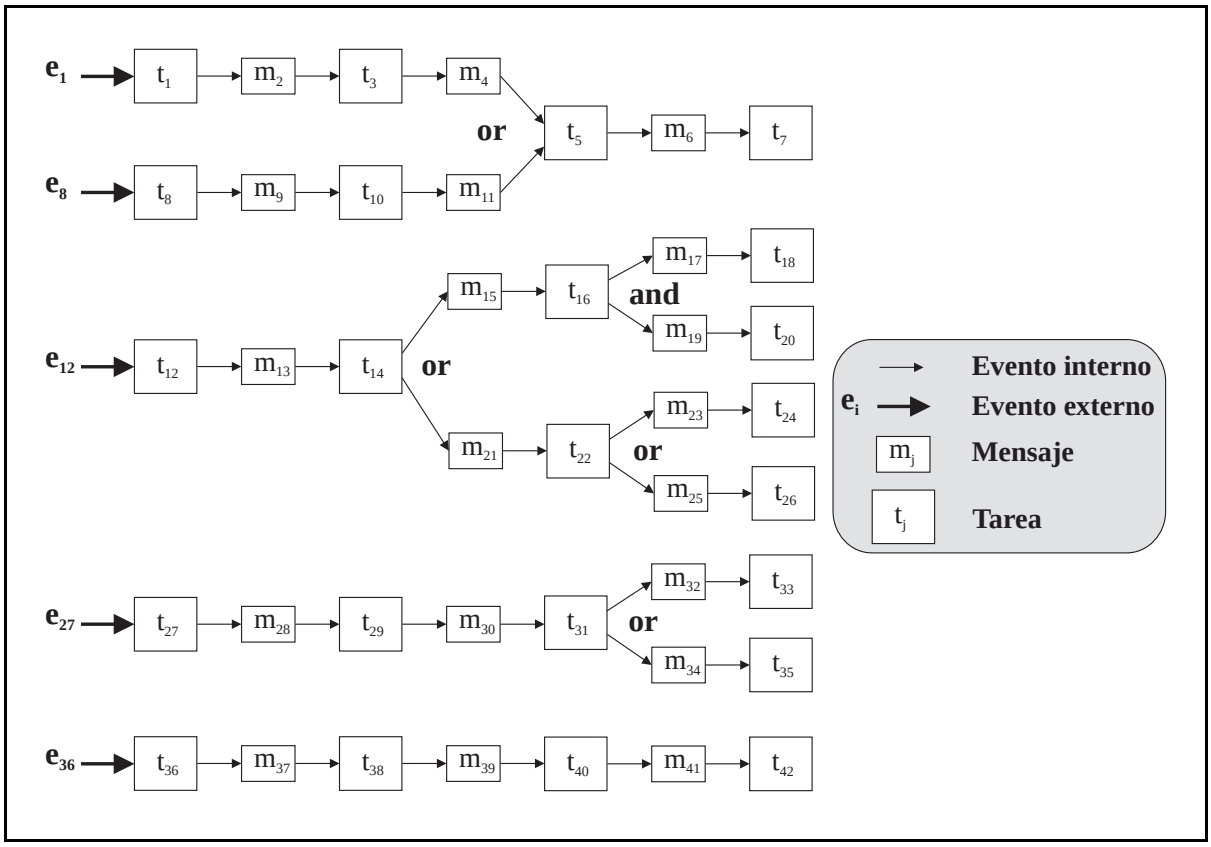

Figura 4-9. Estructura de las secuencias de respuesta del sistema.

aplicar las técnicas de análisis y de planificación.

Para terminar con la descripción del sistema del ejemplo, damos la asignación de tareas a procesadores y de mensajes a redes de comunicación que se muestra en la Figura 4-12. En esta figura aparecen también las acciones duplicadas del *modelo equivalente* localizadas en los recursos correspondientes en función de la localización previa de las acciones de las que proceden.

Aplicamos ahora las técnicas desarrolladas para resolver el problema de la asignación de prioridades a las acciones que componen este sistema. Para ello, utilizamos las herramientas extendidas de planificación y de generación de ejemplos implementadas. El procedimiento que seguimos para la aplicación de las herramientas al ejemplo es similar al que se realizó con el ejemplo mostrado en el capítulo 3 (ver apartado 3.5). A partir de una carga inicial de sistema, aplicamos la herramienta de planificación para resolver el problema de la asignación de prioridades mediante las dos técnicas implementadas (incluyendo los efectos del *jitter*), y seguidamente, aplicamos la herramienta de generación de ejemplos en

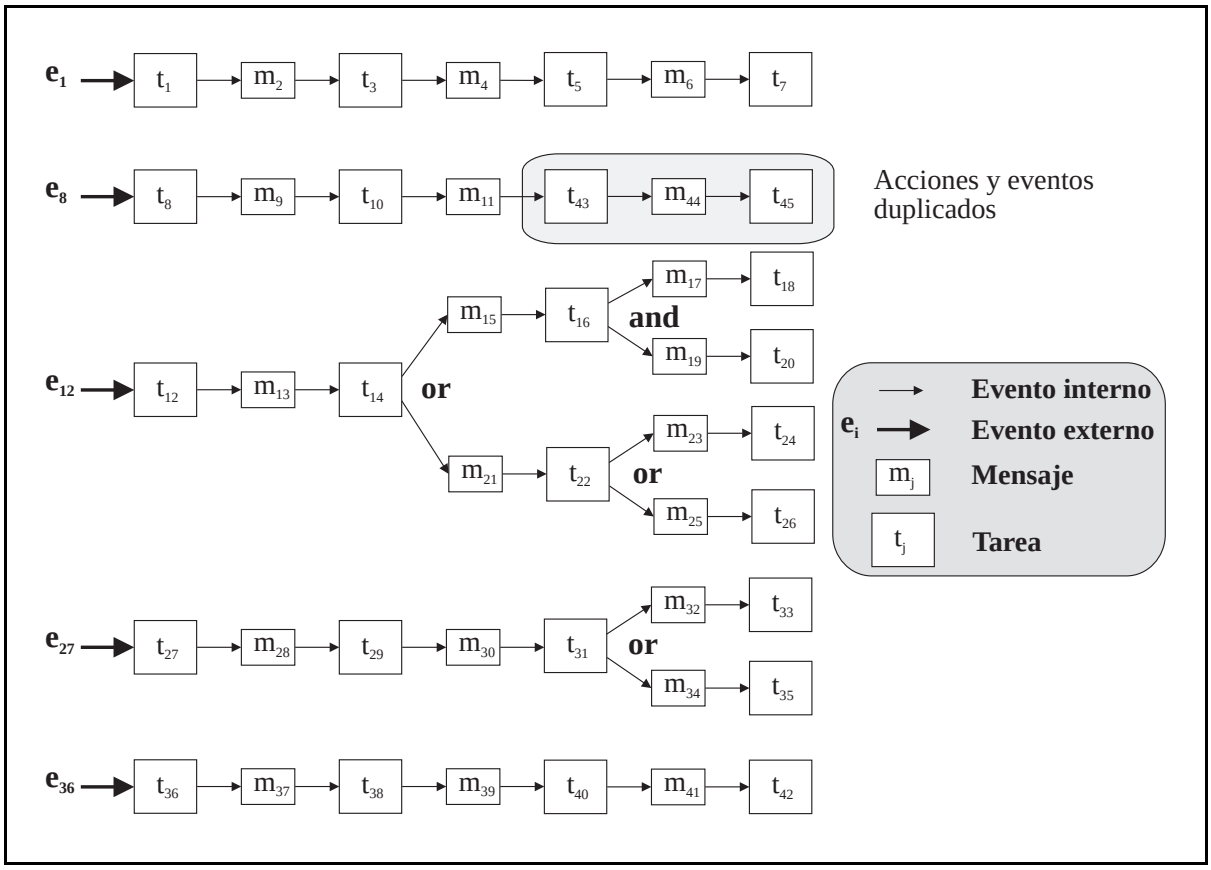

Figura 4-10. Acciones duplicadas en el *modelo equivalente*.

el modo de *variación de carga múltiple* (con un aumento de carga del 3.45 %). Este proceso se repite hasta que se alcanza un valor de la carga para el cual el sistema deja de ser planificable (ninguna de las dos técnicas encuentra una solución que haga al sistema planificable). Para dar una idea de la magnitud del problema que estamos tratando, diremos que el número de posibles asignaciones de prioridad es del orden de 10^{21}.

Las pruebas que vamos a realizar sobre este ejemplo pretenden por un lado, evaluar el comportamiento tanto de la técnica de análisis como de los algoritmos de planificación desarrollados para el *modelo general* de sistema distribuido, y por otro lado, establecer una comparación entre los algoritmos de planificación en cuanto a su capacidad para obtener soluciones y a la calidad de dichas soluciones. Un punto interesante en la evaluación de la técnica de análisis será comprobar cómo la introducción de las configuraciones en el *modelo equivalente* reduce realmente el pesimismo propio del análisis, y en qué magnitud se reduce este pesimismo.

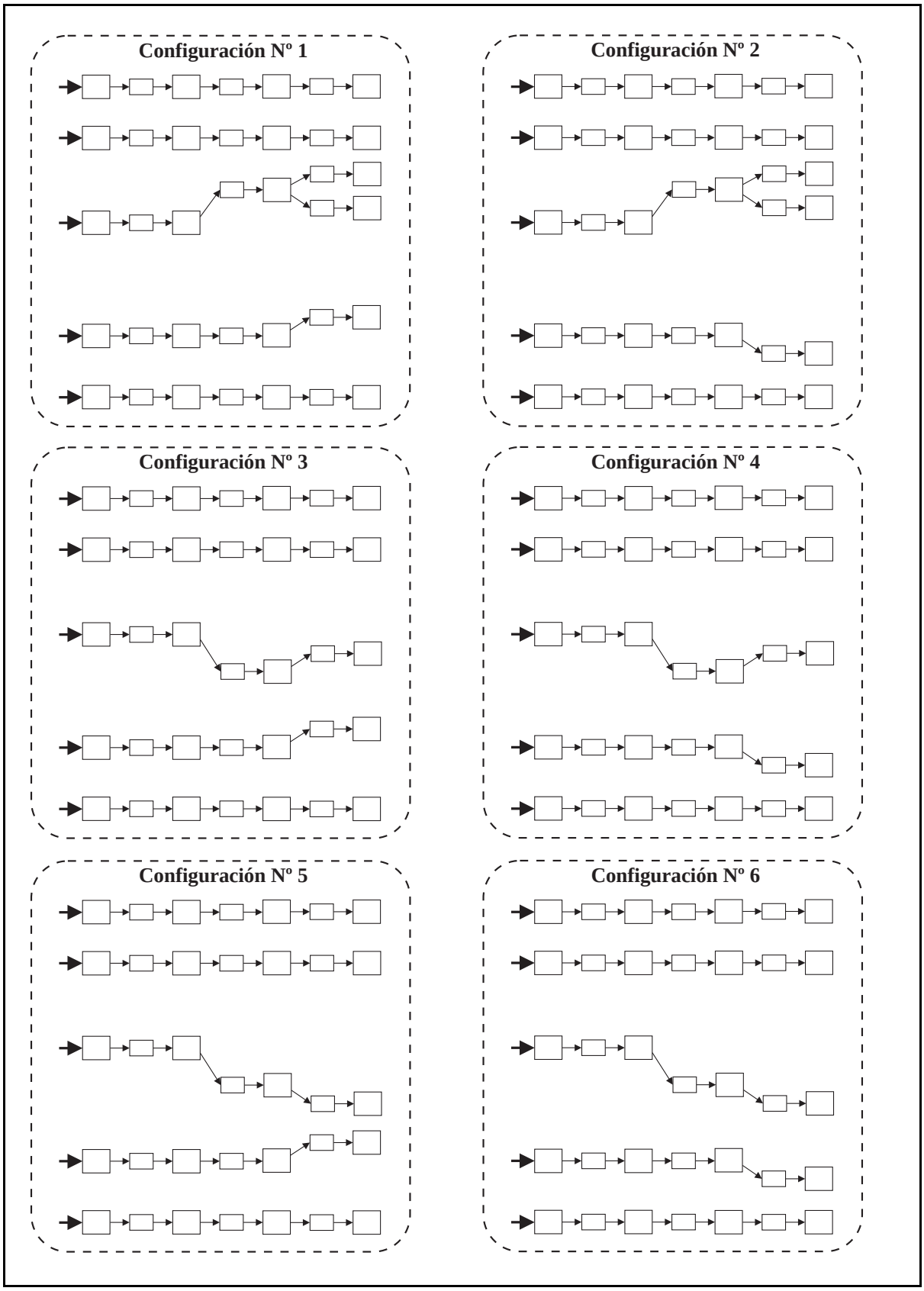

Figura 4-11. Configuraciones del *modelo equivalente*.

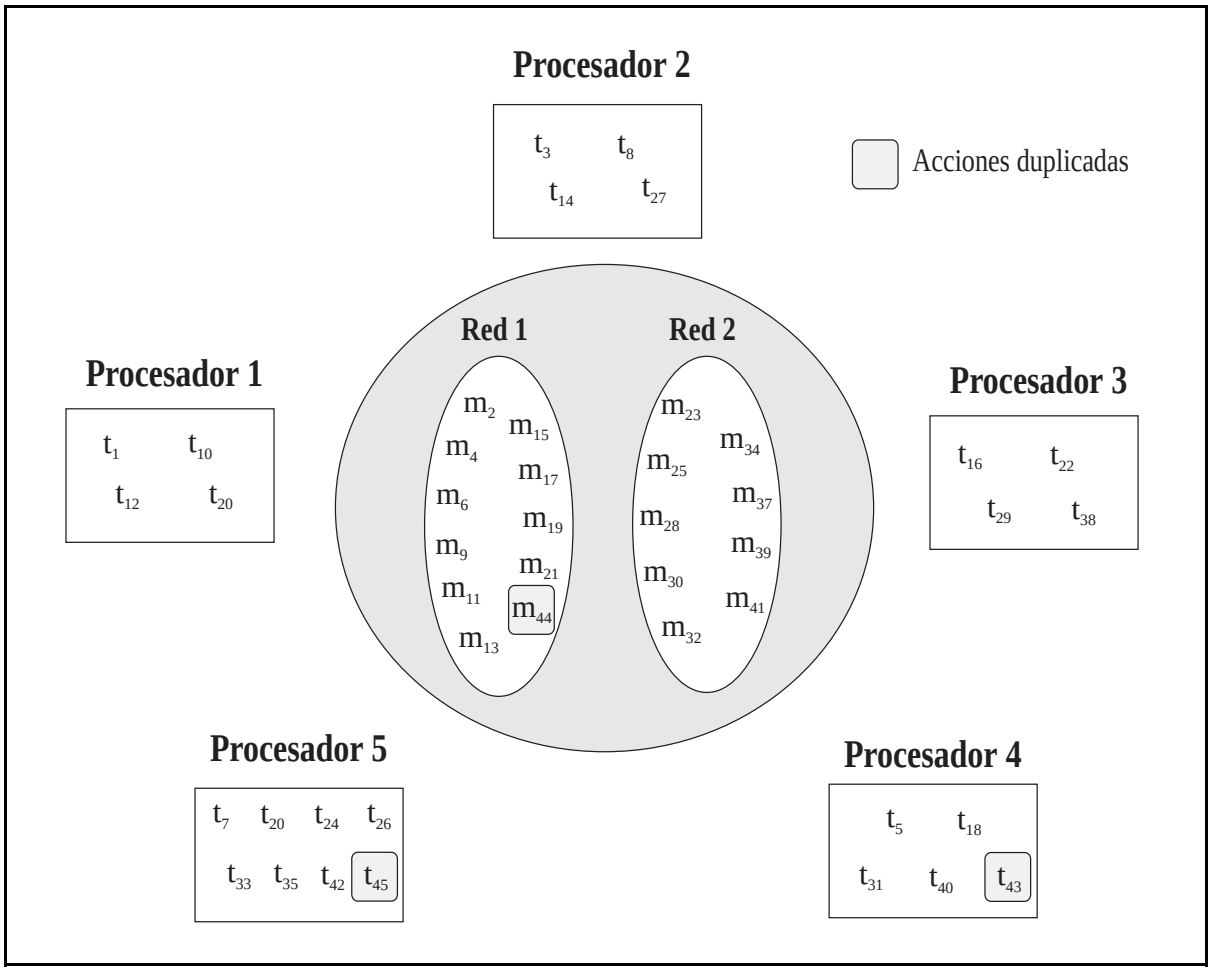

Figura 4-12. Asignación de acciones a recursos.

En primer lugar evaluaremos separadamente los algoritmos aplicando las herramientas desarrolladas sobre el ejemplo y comparando las soluciones que se obtienen permitiendo o sin permitir la optimización (número determinado de iteraciones sobre una solución planificable). El número de iteraciones sobre una solución planificable que emplearemos para la optimización será de 5 para el algoritmo *HOPA extendido*, y 15 para el algoritmo de *templado simulado extendido* (este último necesita más iteraciones para alcanzar soluciones).

En la Figura 4-13 se muestran los resultados obtenidos por el algoritmo *HOPA extendido* con y sin la optimización de soluciones. La gráfica representa los índices globales de planificación de las soluciones obtenidas y los tiempos de CPU que se han empleado en obtenerlas. Como se puede observar en la gráfica, la optimización no solamente da lugar a mejores soluciones, sino que incluso se emplean menores tiempos en obtenerlas cuando nos acercamos al límite de planificabilidad del sistema. Este cruce en los tiempos de CPU

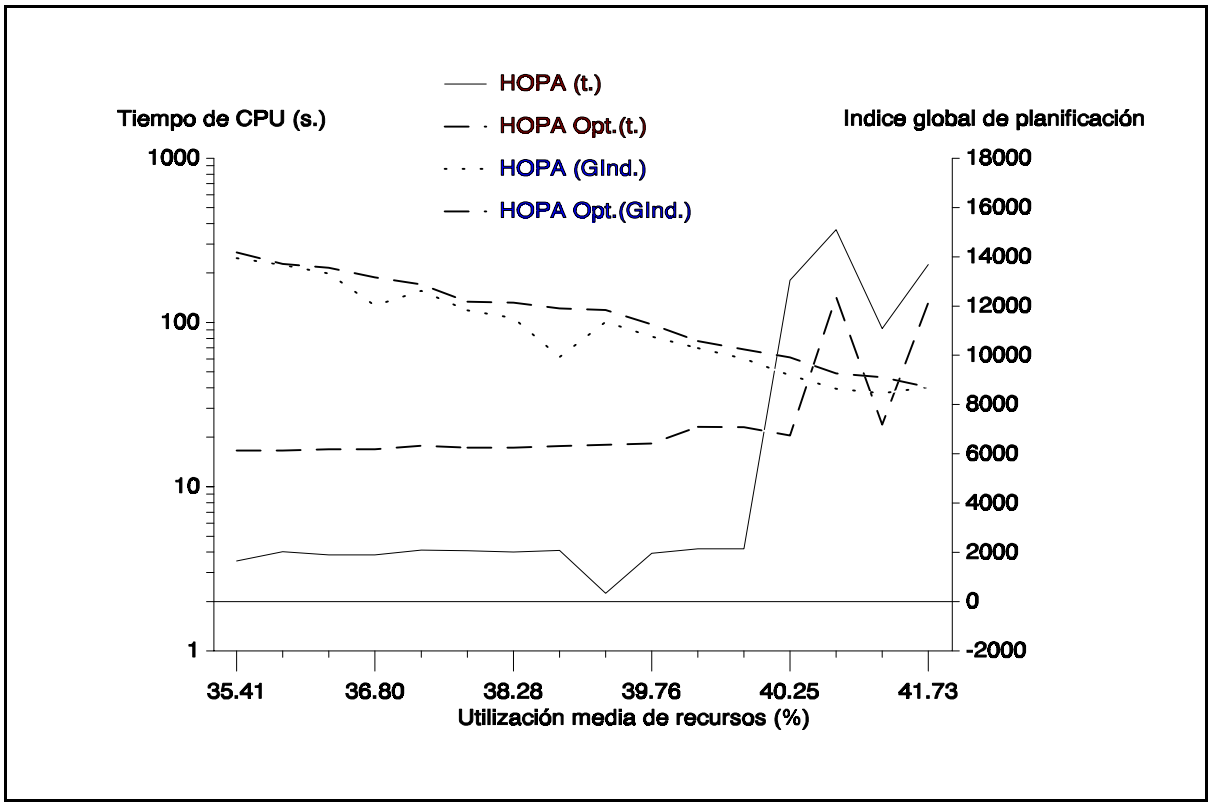

Figura 4-13. Algoritmo heurístico con y sin optimización.

consumidos con y sin optimización se debe a que la optimización permite obtener mejores asignaciones iniciales de prioridades, que constituyen mejores puntos de partida para la asignación de prioridades al sistema completo. Por esta razón, para utilizaciones cercanas al límite de planificabilidad, cuando se hace realmente difícil encontrar soluciones que hagan al sistema planificable, mejores asignaciones iniciales de prioridades conducen a menores tiempos para la asignación de prioridades al sistema completo.

La Figura 4-14 muestra los resultados obtenidos por el algoritmo de *templado simulado extendido* con y sin la optimización de soluciones. La gráfica representa los índices globales de planificación de las soluciones obtenidas y los tiempos de CPU que se han empleado en obtenerlas. De igual manera que para el algoritmo *HOPA extendido*, la optimización da lugar a mejores soluciones, y también se emplean menores tiempos en obtenerlas cuando nos acercamos al límite de planificabilidad del sistema. Más aún, cerca del límite de planificabilidad la optimización permite alcanzar soluciones planificables para algunos casos en los que el resultado sin optimizar era no planificable.

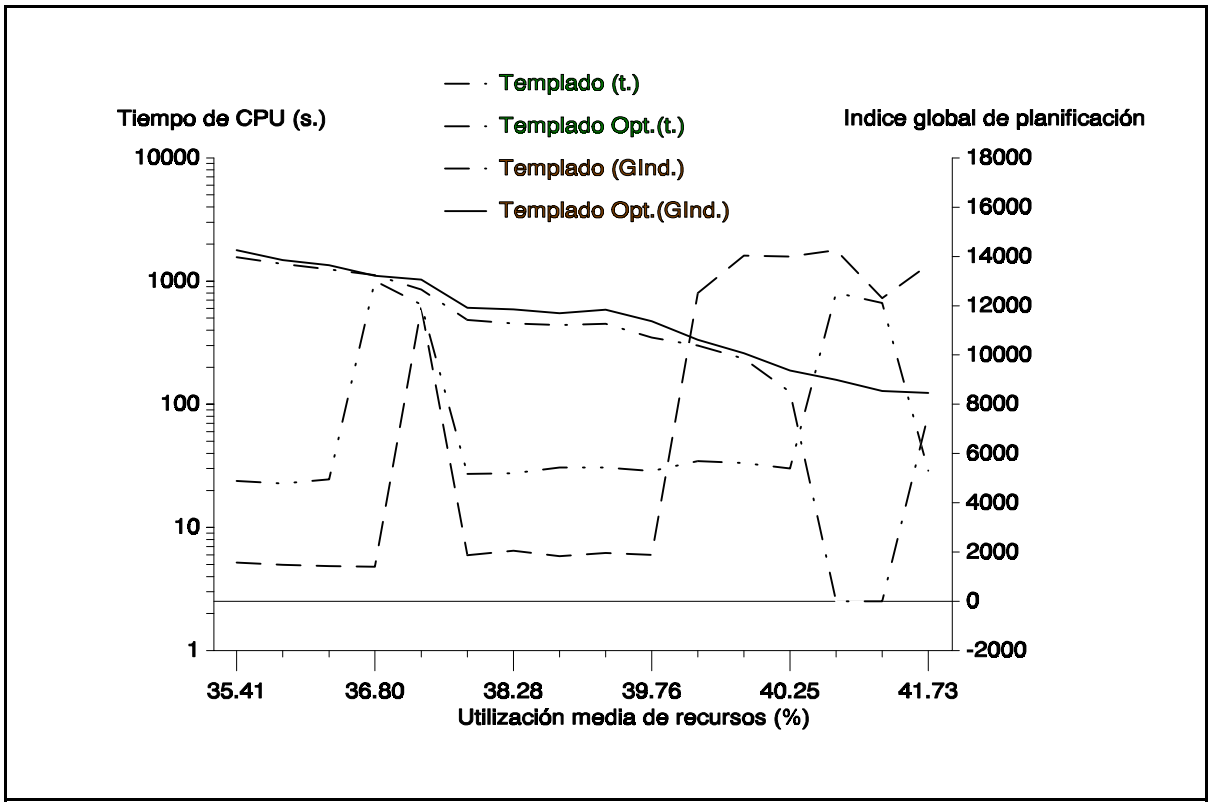

Figura 4-14. Algoritmo de *templado simulado* con y sin optimización.

Con los resultados obtenidos de la aplicación de las herramientas de planificación al ejemplo y representados en las gráficas anteriores (Figura 4-13 y Figura 4-14), realizamos una comparación entre las técnicas de planificación teniendo en cuenta los mejores resultados alcanzados por las mismas (utilizando la optimización de las soluciones alcanzadas). La Figura 4-15 muestra una gráfica con esta comparación en la que se representan los índices globales de planificación de las mejores soluciones obtenidas y los tiempos de CPU que se han empleado en obtenerlas. Considerando los datos representados en esta gráfica podemos hacer los siguientes comentarios sobre la comparación de las dos técnicas de planificación:

- La extensión de los algoritmos de planificación permite resolver el problema de la asignación de prioridades en el *modelo general* de sistema distribuido.

- Ambos algoritmos son capaces de realizar en la misma medida la planificación de sistemas distribuidos de tiempo real estricto representables por el *modelo general* definido, es decir, los dos algoritmos alcanzan aproximadamente el mismo límite de planificabilidad (utilización de los recursos para la que el sistema deja de ser

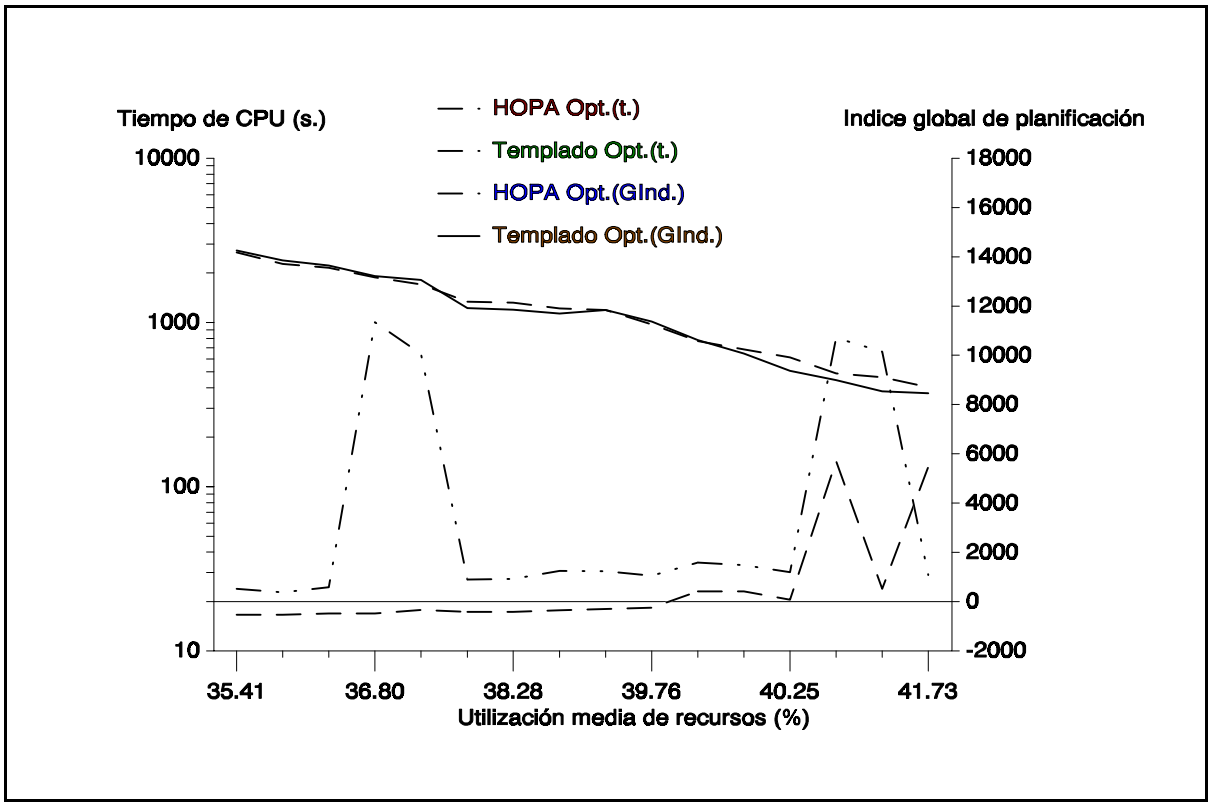

Figura 4-15. Comparación entre los algoritmos de planificación extendidos.

planificable).

- Los índices de planificación de las soluciones alcanzadas por los dos algoritmos son muy similares y no se puede decir que uno sea claramente mejor que el otro en el sentido de la calidad de las soluciones alcanzadas.

- En cualquier caso, el algoritmo heurístico sigue siendo bastante más rápido que el algoritmo de *templado simulado*. En promedio, el tiempo de CPU consumido por el algoritmo heurístico extendido es un orden de magnitud menor que el empleado por el *templado simulado extendido*.

De los resultados que se obtienen de la aplicación de los algoritmos a diferentes ejemplos podemos deducir que para bajas utilizaciones de los recursos, al igual que ocurría con los algoritmos aplicables al *modelo lineal*, también estos encuentran soluciones factibles muy rápidamente. En este caso no hay grandes diferencias en la calidad de las soluciones alcanzadas por ambos algoritmos (se consiguen soluciones con índices globales de

planificación muy similares). Del mismo modo, hay una diferencia de rendimiento considerable en favor del algoritmo heurístico cuando aumenta la utilización de los recursos y la relación entre los períodos y los plazos.

A partir de las pruebas realizadas también podemos concluir que la técnica de análisis desarrollada para el *modelo general* de sistema distribuido permite calcular realmente las respuestas de peor caso para las tareas y los mensajes, y además, se puede utilizar de una manera efectiva en el desarrollo de los algoritmos de asignación de prioridades.

En lo referente a la evaluación de las ventajas que conlleva la introducción de las configuraciones en el *modelo equivalente* para el análisis, vamos a comprobar mediante el sistema del ejemplo cómo efectivamente hay una reducción del pesimismo en el análisis. Para ello, aplicaremos las técnicas de planificación a los ejemplos generados para la obtención de las gráficas anteriores (Figura 4-13, Figura 4-14 y Figura 4-15) considerando las acciones con *or de salida* como *and de salida*, o sea, eliminando las configuraciones del análisis. La herramienta de planificación se irá aplicando sobre los ejemplos (aumentando progresivamente las utilizaciones de los recursos) hasta que se alcance el límite de planificabilidad.

En la Figura 4-16 se muestran los resultados obtenidos de la aplicación del análisis considerando las acciones con *or de salida* como *and de salida* comparados con los que se obtuvieron mediante el uso de configuraciones. La gráfica representa los índices globales de planificación de las soluciones obtenidas y las utilizaciones medias de los recursos para los distintos ejemplos considerados y para las mejores soluciones obtenidas (con cualquiera de los dos algoritmos de planificación utilizados). Como se podía esperar, la consideración de las configuraciones da lugar a utilizaciones más bajas de los recursos (de acuerdo con el cálculo de utilizaciones definido en la página 4-27), ya que se construye una situación de peor caso más cercana al peor caso real. Además, se consiguen mejores soluciones al problema de la asignación de prioridades (esta mejoría se acentúa en parte porque la consideración de configuraciones lleva implícita la obtención de índices globales de planificación mayores). Si tomamos como referencia las utilizaciones calculadas para los ejemplos haciendo uso de las configuraciones, la última solución planificable se alcanza para las siguientes utilizaciones medias: 41.73% cuando se utilizan configuraciones y 37.79% cuando se utilizan *and de salida*. Tomando como referencia las utilizaciones calculadas para los ejemplos haciendo uso

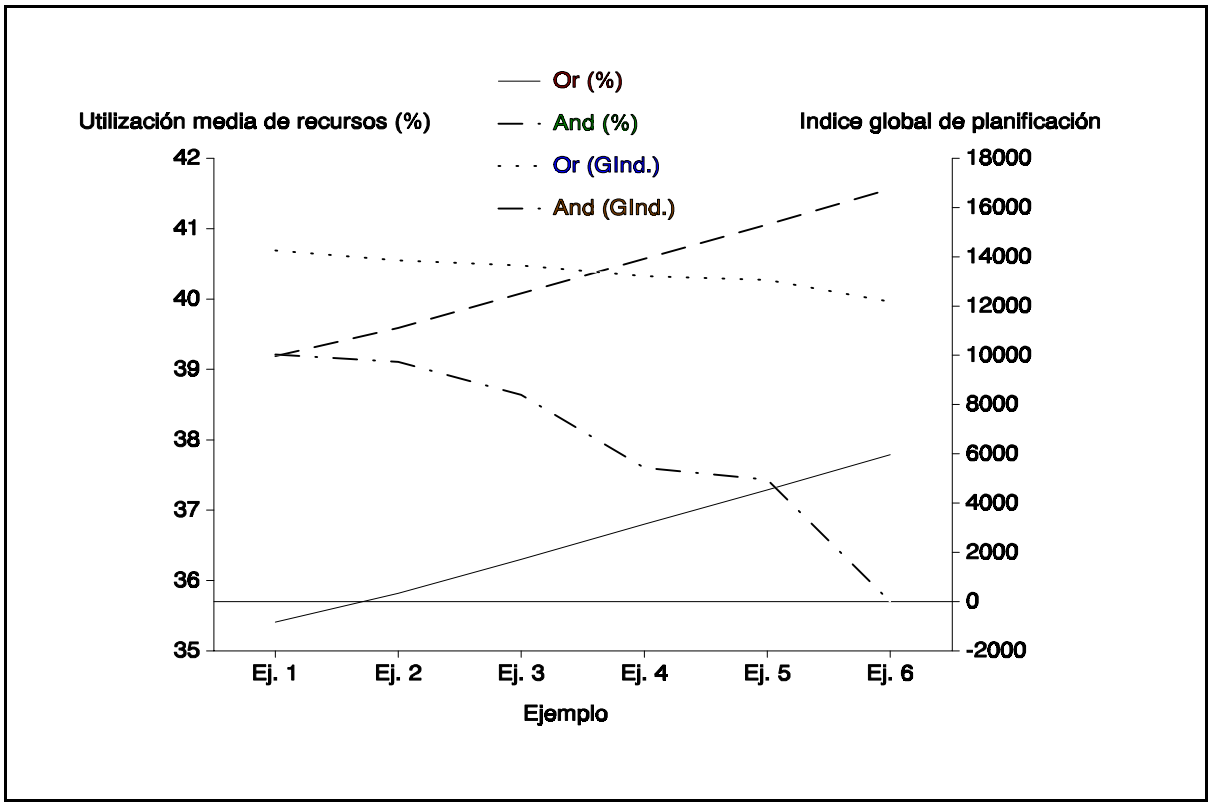

Figura 4-16. Resultados de la introducción de las configuraciones.

de las *and de salida*, las medidas de la utilización serían 46.48% y 41.56 respectivamente. Por lo tanto, la reducción del pesimismo en el análisis a través de la introducción de las configuraciones en el *modelo equivalente* permite aumentar considerablemente el límite de utilización de los recursos, incrementando así la posibilidad de realizar la planificación para un mayor número de casos.

La existencia de acciones con *and de salida* o de acciones con *or de salida* que requieren un tratamiento como *and de salida* hace que el algoritmo heurístico extendido no alcance soluciones planificables para algunos casos determinados en los que algoritmo de *templado simulado extendido* sí las alcanza. Esto se debe a que las acciones con *and de salida* dan lugar a la aparición de varias posibilidades en la asignación de plazos locales a la misma acción (para las acciones anteriores a la *and de salida*), y esta situación tiene que convivir con la necesidad de forzar la asignación de estos plazos locales debida a la existencia de acciones duplicadas y de prioridades preasignadas.

Por consiguiente, a partir de las múltiples pruebas realizadas con los dos algoritmos de planificación desarrollados para el *modelo general* de sistema distribuido de tiempo real estricto y de los resultados que se han conseguido, podemos hacer la siguiente consideración para el uso de los métodos desarrollados: a la hora de afrontar la asignación de prioridades para un sistema distribuido que responda al *modelo general*, en primer lugar se intentará aplicar el algoritmo heurístico de planificación (obtiene una respuesta más rápida), y si éste no consigue alcanzar una solución planificable para el sistema, entonces se aplicará el algoritmo de *templado simulado*.

5. *Implementación de redes de comunicación de tiempo real estricto*

5.1. Introducción

La mayoría de las veces en los sistemas multiprocesadores y distribuidos, los procesos o tareas necesitan cooperar entre sí para sincronizarse o intercambiarse cualquier tipo de información, ya sea en el mismo nudo, o en nudos distintos. Por lo tanto, si tratamos de desarrollar aplicaciones de tiempo real estricto, basadas en sistemas multiprocesadores y distribuidos, necesitamos utilizar redes capaces de garantizar una comunicación de tiempo real estricto.

Muchas interfases convencionales para intercomunicación de procesos o tareas, son inadecuadas para su utilización en aplicaciones de tiempo real, debido a que no pueden implementar un método de paso de datos eficiente y determinista, principalmente a causa de la ausencia de prioridades.

En las extensiones de tiempo real para sistemas operativos portables POSIX [POS93], se define una interfase para la comunicación entre procesos de tiempo real, que utiliza el mecanismo de paso de mensajes (basado en prioridades) para la intercomunicación de procesos o tareas. Aunque esta interfase, en la que se definen las *colas de mensajes*, está inicialmente concebida para las comunicaciones locales en sistemas no dirtribuidos, es la que vamos a tomar como referencia para implementar la comunicación entre tareas y desarrollar nuestra implementación de redes de comunicación de tiempo real estricto. Esta implementación se llevará a cabo mediante una sencilla extensión que permite la configuración de las diferentes colas de mensajes en los distintos nudos procesadores.

Como lenguaje de programación para desarrollar nuestra implementación elegimos el Ada, debido entre otras cosas a la facilidad de mantenimiento y cambio a nuevas versiones,

y sobre todo, a los mecanismos de que dispone para la programación concurrente, la manipulación de datos, la sincronización sobre los mismos, y además para la creación y utilización de paquetes Ada estándar.

Aunque ya ha sido aprobado el Ada95 [ADA95], debido a que no disponemos aún de una versión de este compilador, el desarrollo lo realizaremos en Ada83 utilizando un compilador con algunas extensiones que no son Ada estándar, pero que son fácilmente acomodables a los nuevos servicios del Ada95. Además comentaremos aquellas partes de la implementación de las redes de comunicación que se puedan mejorar con esta nueva versión del Ada.

Si tenemos en cuenta que la implementación de las redes de comunicación que vamos a presentar está fundamentada en Ada y POSIX, podemos considerar la implementación de las colas de mensajes como una aplicación portable, aún cuando en el desarrollo nos centremos en una plataforma concreta, y en unas redes concretas.

La plataforma que constituye nuestro sistema distribuido está formada por tarjetas procesadoras basadas en el microprocesador MC68010 de Motorola sobre un bus VME, y por PC's con microprocesadores 80386/80486 de Intel. Las tarjetas para el bus VME disponen de puertas serie RS-232 y RS-485, y el PC solamente tiene una puerta serie RS-232. Así pues, las redes de comunicación de tiempo real que van a ser objeto de nuestra implementación son: las líneas serie RS-232 y RS-485 (dedicadas para comunicaciones punto a punto), y el bus VME (utilizado como red multipunto).

Los compiladores de Ada que se utilizan son: Ada cruzado de Alsys (para las tarjetas VME, desarrollando el software sobre estaciones Hewlett-Packard Serie 300), y Ada de Alsys para PC con sistema operativo LynxOS.

Seguidamente describiremos las colas de mensajes y su implementación para las redes mencionadas. El modelado de esta implementación (desde el punto de vista del análisis de tiempo real) junto con algunos ejemplos de su utilización se muestran en el capítulo 7 de esta Tesis.

5.2. Colas de mensajes

Las colas de mensajes constituyen un mecanismo de comunicación entre procesos mediante el paso de mensajes. Tomando como referencia el mecanismo de paso de mensajes que utiliza el POSIX [POS93], al que ya hemos hecho referencia en la introducción, podemos describir las colas de mensajes como un conjunto de operaciones que actúan sobre una estructura de datos, y nos permiten el intercambio de mensajes entre distintos procesos o tareas, de acuerdo con unas reglas preestablecidas capaces de satisfacer requerimientos de tiempo real.

Una cola de mensajes es una cola de prioridad, en la que los procesos depositan y extraen los mensajes con los que se comunican, con independencia del nudo en el que se estén ejecutando. Por lo tanto, las colas de mensajes deben disponer del software necesario, para realizar las comunicaciones entre los distintos nudos, a través de las redes del sistema distribuido. La manipulación de mensajes de la misma prioridad en una cola se realiza siguiendo una política FIFO.

Dado que las colas de mensajes deben satisfacer las necesidades de comunicación de un sistema multiprocesador distribuido, se puede hacer una clasificación de éstas en:

Colas de mensajes locales, si comunican únicamente procesos localizados en el mismo nudo, por lo que no necesitan utilizar las redes de interconexión entre los distintos nudos.

Colas de mensajes globales, si comunican procesos que se encuentran en nudos distintos, aunque también se pueden utilizar localmente. Presentan algunas restricciones en las operaciones que se pueden realizar sobre ellas.

Por consiguiente, es necesario que la estructura de datos de las colas de mensajes disponga de parámetros de configuración que permitan declarar aquellas colas que sean globales, y los caminos de acceso a esas colas, es decir, las redes por las que se conectan.

Dentro de lo que constituye propiamente la estructura de datos de las colas de mensajes, y antes de describir las operaciones que se pueden realizar sobre éstas, vamos a definir los siguientes términos:

Nombre de la cola. Es el identificador de la cola, a través del cual las tareas van a solicitar la creación o la apertura de una cola de mensajes. Lleva implícita la información del nudo en el que se localiza la cola.

Descriptor. Es la etiqueta identificativa con la que las tareas se conectan a una cola de mensajes para operar sobre ella, y como estamos en sistemas multiprocesadores distribuidos se pueden clasificar en:

Descriptores locales	Si identifican a una cola de mensajes del propio nudo.
Descriptores remotos	Si identifican a una cola de mensajes de otro nudo.

Modo de acceso. Determina el modo de acceso permitido sobre la cola, que puede ser:

Lectura	Sólo se permite extraer mensajes de la cola.
Escritura	Sólo se permite introducir mensajes en la cola.
Lectura-escritura	Se permite tanto extraer como introducir mensajes en la cola.

Prioridad. Es el nivel de prioridad que tiene un mensaje tanto en la cola de mensajes, como en la transmisión, dentro del rango de prioridades definido.

Mensaje. Es el elemento de comunicación que gestionan las colas de mensajes.

Atributos de una cola. Son un conjunto de parámetros que caracterizan a cada cola de mensajes, entre los que destacan:

número máximo de mensajes en la cola
longitud máxima de mensaje
número actual de mensajes en la cola
característica de global o local de la cola

Por otra parte, teniendo en cuenta el marco de sistema distribuido en el que se van a ejecutar las operaciones sobre las colas de mensajes, vamos a dar una clasificación de éstas de las forma siguiente:

Locales. Cuando la operación se realiza sobre una cola local o global localizada en el propio nudo, y por tanto no hace uso de las redes de tiempo real.

Remotas. Si la operación se realiza sobre una cola global que se encuentra en otro nudo y utiliza por tanto las redes de tiempo real. Estas operaciones van a tener dos componentes: una que se realiza en el propio nudo en el que se ordena la operación, y otra que se lleva a cabo en el nudo en el que se localiza la cola de mensajes.

Antes de describir las operaciones que se pueden realizar sobre las colas de mensajes, vamos a comentar algunas diferencias con la interfase POSIX tomada como referencia, y que se deben fundamentalmente al carácter distribuido de nuestra implementación (la interfase POSIX se concibe para el intercambio de información entre procesos o tareas dentro del mismo procesador). No vamos a considerar la operación de establecimiento de atributos en cualquier momento, ya que la información que contienen es más propia de establecerse únicamente en el instante de creación de la cola. Tampoco vamos a considerar la operación de notificación de mensaje recibido debido a que supone una sobrecarga en la implementación, y las necesidades de recepción quedan perfectamente cubiertas con las operaciones que se suministran a tal efecto. En cuanto a los atributos, no se define como opción el efecto de bloqueo de las operaciones de envío y recepción de mensaje; en su lugar dividimos la recepción en dos operaciones, una bloqueante y otra no bloqueante, y consideramos que el envío nunca es bloqueante.

Para terminar, pasamos a explicar en líneas generales las operaciones que se pueden realizar sobre las colas de mensajes en la implementación que vamos a desarrollar, comenzando con una tabla que contiene una breve descripción de las mismas:

Operación	Descripción
Abre	Creación de una cola de mensajes (con apertura del descriptor) o apertura de un descriptor sobre una cola de mensajes abierta.
Cierra	Cierre de un descriptor.
Destruye	Destrucción de una cola de mensajes.
Envía	Introducción de un mensaje en una cola.
Recibe	Extracción bloqueante de mensajes de una cola (sólo retorna con el mensaje).
Recibe si hay	Extracción no bloqueante de mensajes de una cola (retorna tanto con mensaje como sin él).
Coge atributos	Consulta de los atributos de una cola de mensajes determinada en cualquier instante.

1.- ***Abre.***

Esta operación puede actuar de dos maneras diferentes; por un lado, se encarga de la creación de las colas, y por otro, de la apertura de descriptores. Existe un parámetro mediante el que podemos seleccionar cual de las dos operaciones es la que queremos realizar.

La creación de la cola, identificada por su nombre, implica la reserva del espacio de memoria donde se van a depositar los mensajes, por consiguiente se necesitan como parámetros los atributos de la cola (el número máximo de mensajes que puede almacenar la cola y la longitud máxima de estos mensajes). Además, como resultado de la operación se devuelve un descriptor.

La operación de apertura de un descriptor se realiza sobre una cola que ya esté creada.

En cualquier caso se devuelve un descriptor que apunta hacia la cola que se indica y también se establece el tipo de acceso permitido para éste sobre la cola.

La creación de colas está restringida al propio nudo, por lo que no se permite una operación remota de creación de cola.

Los parámetros que utiliza esta operación son:

Entrada: *nombre de la cola*, *acceso*, *crear/no crear* y *atributos* (*número máximo de mensajes* y *longitud máxima de mensaje*).

Salida: *descriptor*.

Se pueden producir errores al realizar esta operación en los casos en los que:

- Se quiera crear la cola y ésta ya haya sido creada.
- Se quiera crear la cola y estando declarada como global, pertenezca a otro nudo.
- El número de mensajes con el que se quiere crear la cola es mayor que el máximo estipulado para todas las colas.
- La longitud de mensaje con la que se quiere crear la cola es mayor que la máxima estipulada para todas las colas.
- No hay espacio en la estructura de datos para crear una nueva cola.
- No hay espacio en la estructura de datos para abrir un nuevo descriptor.
- Hay problemas de comunicación, si la operación de apertura de un descriptor es remota.

2.- ***Cierra*.**

Esta operación cierra un descriptor, es decir, desconecta la tarea propietaria del descriptor y la cola identificada por éste.

Además, ejecuta la operación de destrucción de la cola en el caso de que sea el último descriptor abierto, y la cola esté en *proceso de destrucción* (ver la operación *destruye*).

El único parámetro que utiliza esta operación es el *descriptor* (parámetro de entrada).

Se pueden producir errores al realizar la operación en los casos en los que:

- El descriptor que se quiera cerrar no haya sido abierto.
- Haya problemas de comunicación, si la operación de cierre es remota.

3.- ***Destruye*.**

Esta operación destruye una cola, o sea, libera la memoria que tenía reservada y se da de baja en la estructura de datos. En el caso de que la cola tenga descriptores abiertos, la destrucción no se lleva a cabo inmediatamente, sino que se cierra el descriptor y se coloca en *proceso de destrucción*. En esta situación, la cola no va a aceptar en lo sucesivo las operaciones *envía*, *abre* y *destruye*. Además, espera a no tener descriptores abiertos para proceder a su destrucción, que es ejecutada por la operación *cierra* sobre el último descriptor.

Al igual que la operación *abre* para la creación de colas, también esta operación está restringida al propio nudo; por lo tanto, no se permite la operación remota.

El único parámetro que utiliza esta operación es el *nombre de la cola* (parámetro de entrada).

Se pueden producir errores al realizar esta operación en los casos en los que:

- La cola que se pretende destruir no exista en el nudo en el que se ejecute la operación.
- La cola esté ya en *proceso de destrucción*.

4.- *Envía.*

Esta operación se encarga de introducir un mensaje de una longitud determinada en la cola a la que apunta el descriptor que se utiliza, y a una prioridad determinada. En la misma prioridad, el mensaje se encola según una política FIFO.

Además se encarga de despertar a la tarea de mayor prioridad, que se hubiera suspendido al ejecutar un *recibe* sobre la cola vacía.

Los parámetros que utiliza esta operación son: *descriptor*, *mensaje*, *longitud* y *prioridad* (todos parámetros de entrada).

Se pueden producir errores al realizar esta operación en los casos en los que:

- El descriptor no haya sido abierto.
- La cola esté en *proceso de destrucción*.
- La longitud del mensaje que se envía sea mayor que la admitida por la cola.
- La cola esté llena y no admita más mensajes.

- El descriptor no tenga *acceso* de escritura.
- Haya problemas de comunicación, si la operación de envío de un mensaje es remota.

5.- *Recibe*.

Esta operación extrae el mensaje de más alta prioridad de la cola a la que apunta el descriptor que se utiliza. Junto con el mensaje se devuelve su longitud y la prioridad a la que había sido enviado.

Esta es una operación bloqueante, es decir, en el caso de que no haya mensajes en la cola, la tarea que la realiza se suspende hasta que alguna otra tarea ejecute una operación *envía* sobre esa cola.

Los parámetros que utiliza esta operación son:

Entrada: *descriptor*.

Salida: *mensaje*, *longitud* y *prioridad*.

Se pueden producir errores al realizar esta operación en los casos en los que:

- El descriptor no esté activo.
- La cola esté en *proceso de destrucción*, y además esté vacía.
- El descriptor no tenga *acceso* de lectura.
- Haya problemas de comunicación, si la operación de recepción de un mensaje es remota.

6.- *Recibe si hay*.

Esta operación es análoga a *recibe*, con la diferencia de que no es bloqueante, o sea, si no hay mensajes en la cola, no se devuelve el mensaje y se avisa mediante un parámetro.

Los parámetros que utiliza esta operación son:

Entrada: *descriptor*.

Salida: *mensaje*, *longitud*, *prioridad* y *hay/no hay*.

Los errores que se pueden producir al realizar esta operación son los mismos que en *recibe*.

7.- *Coge atributos*.

Esta operación devuelve los atributos de la cola de mensajes a la que apunta el descriptor que se utiliza.

Los parámetros que utiliza esta operación son:

Entrada: *descriptor*.

Salida: *atributos*.

Se pueden producir errores al realizar esta operación en los casos en los que:

- El descriptor no esté activo.
- La cola esté en *proceso de destrucción*, y además esté vacía.
- Haya problemas de comunicación, si la operación es remota.

5.3. Implementación de las colas de mensajes

En el apartado anterior, hemos definido de una manera general las colas de mensajes como un mecanismo de comunicación entre tareas en un sistema distribuido. Ahora vamos a llevar a la práctica la implementación de dichas colas, utilizando el Ada83 como lenguaje de desarrollo. Aunque en esta implementación se utilicen extensiones de un compilador concreto, que no son Ada estándar, para resolver determinados aspectos como la suspensión de tareas, con la reciente aprobación del Ada95 [ADA95] la implementación no solamente se convierte en completamente general y portable, sino que también se puede ver mejorada en el sentido de conseguir un software más eficiente y elegante.

Cuando se trata de implementar las colas de mensajes para un sistema distribuido se pueden observar dos partes claramente diferenciadas; por un lado, la parte correspondiente a lo que son propiamente las colas de mensajes (su estructura y operaciones), y por otro, las redes de comunicación entre nudos, utilizadas por éstas para realizar las operaciones remotas. Además como estamos efectuando una implementación de tiempo real, debemos desarrollar un software determinista, tanto para las colas de mensajes, como para las redes que interconectan los diferentes nudos del sistema distribuido.

En este apartado vamos a describir la estructura general de la implementación de las colas de mensajes y de las comunicaciones, a través de la definición de los paquetes Ada correspondientes, y también vamos a describir la estructura particular de las colas de

mensajes, junto con su implementación en algunos de estos paquetes, siempre teniendo en cuenta la definición de cola de mensajes realizada en el apartado anterior.

Antes de entrar en la descripción exhaustiva de la implementación, detallaremos algunos términos importantes de la misma:

Nudo actual. Con él denotamos el propio nudo, es decir, el nudo en el que nos situamos y desde el que nos referimos al resto del sistema distribuido.

Nudo remoto. Es cualquier nudo que no sea el actual (el resto de los nudos del sistema distribuido).

Espacio de memoria. Es el espacio reservado en la memoria para que sea utilizado por las colas de mensajes para almacenar la información correspondiente a cada uno de los mensajes que contienen. Definimos una unidad de memoria que llamamos *segmento*, y establecemos los mecanismos adecuados de reserva y liberación de segmentos de memoria.

Espacio de mensajes. Es un espacio que reservamos en la memoria para el manejo de los mensajes utilizados en las comunicaciones en operaciones remotas. La unidad en este espacio es el *mensaje* y, como en el espacio de memoria, también establecemos los mecanismos adecuados de reserva y liberación de mensajes. Este *mensaje* debe dar soporte a los protocolos tanto de comunicaciones, como de codificación de operaciones remotas, y también al mensaje propio de las colas de mensajes. Con ello evitamos copias sucesivas de los mensajes generados en las operaciones remotas.

Colas de espera. Las colas de espera aparecen debido al efecto bloqueante de la operación *recibe*. Tenemos dos colas de espera: una *cola de espera local*, donde las tareas en ejecución de una operación *recibe* local esperan según su prioridad, a que la cola de mensajes deje de estar vacía, y una *cola de espera remota*, donde se encolan las operaciones *recibe* remotas de acuerdo con una política FIFO (no se utilizan prioridades debido a que los sistemas pueden no

ser homogéneos). Las operaciones remotas en espera tienen preferencia sobre las locales para minimizar el bloqueo que estas primeras pueden sufrir, y agilizar así uso de las colas de mensajes globales. Además, cada cola de mensajes dispone de una cola de espera de cada tipo.

Por último, es necesario resaltar la necesidad de utilizar mecanismos de suspensión de tareas para aquellas operaciones que necesitan una espera. Aquí entran todas las operaciones remotas (en espera de la respuesta) y la operación *recibe* en el caso de que la cola sobre la que se realice no tenga mensajes.

5.3.1. Estructura general

La implementación de las colas de mensajes como mecanismo de comunicación entre tareas en un sistema distribuido está estructurada según los paquetes Ada cuya jerarquía se muestra en la Figura 5-1. En esta figura se diferencian los paquetes Ada que corresponden a la parte estructural de las colas de mensajes, los que corresponden a la parte de comunicaciones entre nudos, y otros que son de uso global. También se destacan cuáles de estos paquetes constituyen la interfase hacia el usuario. Como se puede observar existen algunos paquetes Ada que contienen aspectos de la implementación que requieren una utilización global desde los distintos puntos de la implementación. Así por ejemplo, como se verá más adelante, en el paquete *memoria* se agrupa la memoria que usan las colas de mensajes y la que utilizan los mensajes para las comunicaciones entre los distintos nudos, y en el paquete *protocolo* se agrupa el protocolo de comunicaciones, en cuanto a división del mensaje de comunicación en paquetes de comunicación, y el protocolo de codificación de operaciones remotas.

La capa software de implementación de las comunicaciones, relativa a las colas de mensajes, se corresponde con los paquetes *tipos_de_colas*, *configuracion*, *memoria*, *colas_de_mensajes* y *control_de_colas*. Los demás paquetes soportan el software de comunicaciones entendido como simple intercambio de mensajes entre los diferentes nudos, salvo cuando se implemente el servidor esporádico para comunicaciones como mecanismo de control de este intercambio de mensajes (ver el capítulo 6).

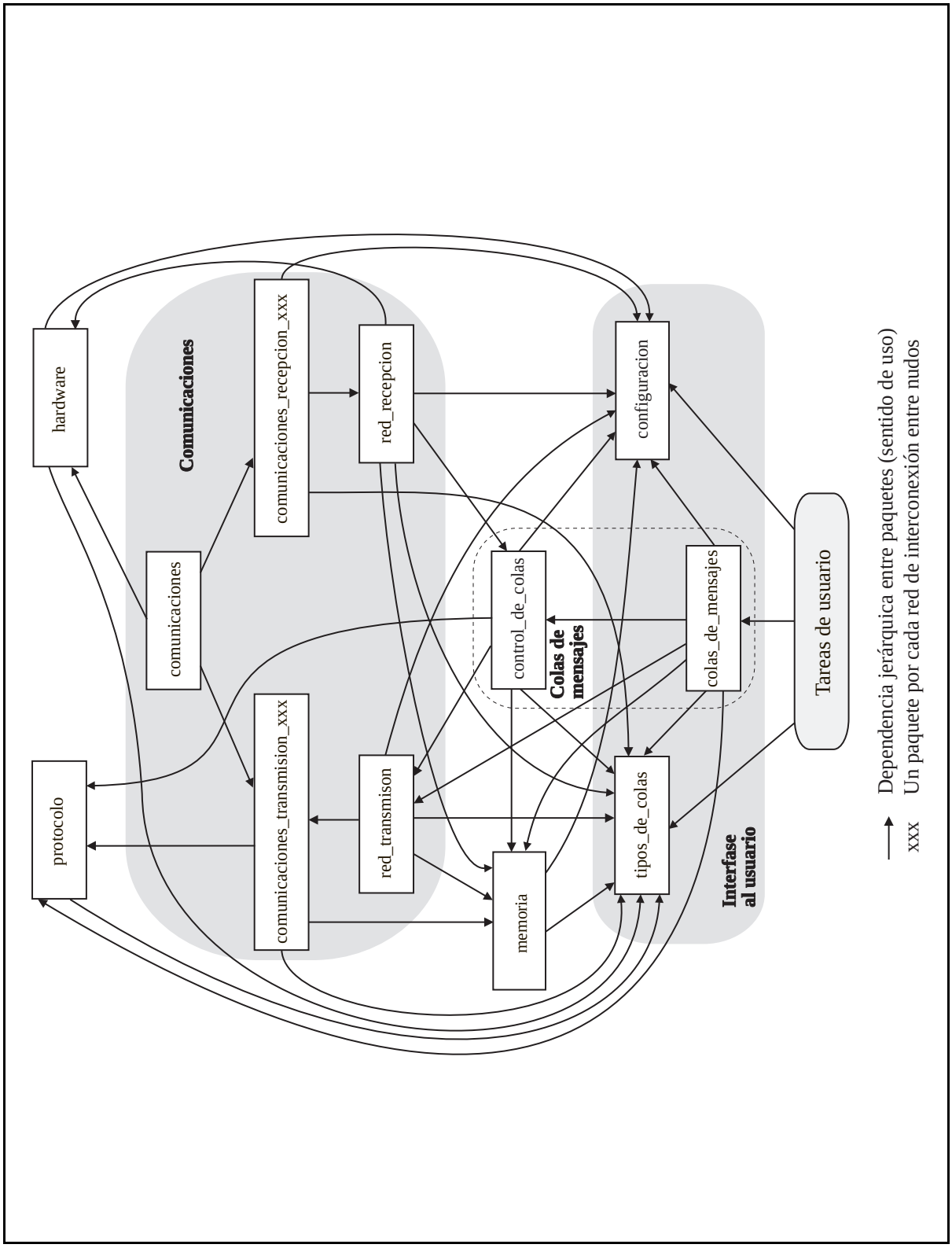

Figura 5-1. Estructura general de las redes de comunicación.

A continuación procedemos a dar una breve descripción de los paquetes Ada que componen la implementación de las colas de mensajes, y que han sido mostrados en la Figura 5-1:

tipos_de_colas	Definición de constantes específicas de las colas de mensajes, de los tipos de datos de la interfase con los programas de usuario y de las excepciones que se pueden producir en el uso de las colas de mensajes.
configuracion	Constantes, tipos, variables, procedimientos y funciones de establecimiento y consulta de la configuración del sistema, tales como, establecimiento del nudo actual, y declaración de la localización y accesibilidad de las colas globales.
memoria	Estructura de datos de soporte de la memoria utilizada por las colas de mensajes y estructura de datos de soporte de los mensajes de operaciones remotas (con el fin de evitar sucesivas copias y así optimizar tiempos). Además operaciones de manejo de estas estructuras de datos.
colas_de_mensajes	Operaciones de manipulación de las colas de mensajes y definición del tipo *descriptor*. Constituye la interfase hacia los programas de usuario, junto con los paquetes *tipos_de_colas* y *configuracion*.

control_de_colas	Estructura de datos de las colas de mensajes y tareas de control de ejecución de las operaciones que se realizan sobre ellas, tanto locales, como remotas, teniendo en cuenta los accesos a los recursos caracterizados por la exclusión mutua. Función de error de operaciones remotas y procedimiento de acceso a operaciones remotas.
protocolo	Constantes de codificación del protocolo utilizado en las operaciones remotas.
red_transmision	Procedimiento utilizado en las operaciones remotas para encaminar el mensaje de orden de operación remota, o de respuesta a una operación remota, hacia la red adecuada.
red_recepcion	Estructura de una cola de prioridad de operaciones remotas a ejecutar, o de respuestas a operaciones remotas solicitadas. Tarea de control de la cola y tarea de ejecución de las operaciones o respuestas encoladas.
comunicaciones_transmision_232	Cola de prioridad para la transmisión de los mensajes partidos en paquetes de comunicación, canalizados por *red_transmisión* a través de una línea serie RS-232. Tarea de control de esta cola.

comunicaciones_transmision_485	Cola de prioridad para la transmisión de los mensajes partidos en paquetes de comunicación, canalizados por *red_transmisión* a través de una línea serie RS-485. Tarea de control de esta cola.
comunicaciones_transmision_bus	Cola de prioridad para la transmisión de los mensajes partidos en paquetes de comunicación, canalizados por *red_transmisión* a través del bus VME. Tarea de control de esta cola.
comunicaciones_recepcion_232	Procedimiento de recepción de paquetes de comunicación y recomposición de mensajes enviados a través de una línea serie RS-232.
comunicaciones_recepcion_485	Procedimiento de recepción de paquetes de comunicación y recomposición de mensajes enviados a través de una línea serie RS-485.
comunicaciones_recepcion_bus	Procedimiento de recepción de paquetes de comunicación y recomposición de mensajes enviados a través del bus VME.
comunicaciones	Tareas de atención de interrupción de los niveles utilizados en las redes de comunicación implementadas.
hardware	Estructuras de datos que implementan las utilidades de comunicaciones más dependientes de la máquina.

Esta lista se ve completada por los paquetes Ada correspondientes a la implementación del servidor esporádico de comunicaciones, que se describirán en el capítulo 6. Las especificaciones de todos los paquetes Ada relativas a la implementación realizada se muestran en el apartado A.1 del apéndice A.

5.3.2. Estructura de datos

Vamos a definir la estructura de datos de las colas de mensajes a través de la descripción de las constantes, tipos y variables que la componen. La estructura de datos se encuentra distribuida por los paquetes Ada ya definidos, y alterna elementos visibles declarados en las interfases de los paquetes, con elementos no visibles declarados en los cuerpos de los mismos. En la descripción vamos a comenzar por toda la estructura que debe ser visible para los programas de usuario y después describiremos el resto con independencia de su visibilidad.

Antes de comenzar con la descripción de la estructura de datos, cabe destacar que el tratamiento de los errores que se puedan producir en el uso de las colas de mensajes se realiza mediante la propagación de excepciones; por eso, se considerarán como un punto más a tratar en esta descripción.

Como componentes de la estructura de datos visibles para los programas de usuario podemos considerar aquellos que representan parámetros de las colas, o tipos que contienen definiciones de la interfase entre las colas de mensajes y los programas de usuario. Esta estructura está compuesta por los siguientes datos:

1.- Constantes.

- En el paquete *tipos_de_colas*:

  max_numero_de_colas : **constant** positive;

  Número máximo de colas de mensajes por nudo.

  max_numero_de_mensajes : **constant** positive;

  Número máximo de mensajes por cola de mensajes.

  max_numero_de_descriptores : **constant** positive;

  Número máximo de descriptores por nudo, incluidos locales y remotos.

max_numero_de_descriptores_locales : **constant** positive;

Número máximo de descriptores locales por nudo.

max_longitud_de_mensaje : **constant** positive;

Longitud máxima que puede tener un mensaje de los que componen las colas de mensajes.

long_nombre_de_cola : **constant** positive;

Número de caracteres que componen el nombre de la cola.

- En el paquete *configuracion*:

max_elementos_configurables : **constant** positive;

Número máximo de colas de mensajes globales en el sistema distribuido.

2.- Tipos.

- En el paquete *tipos_de_colas*:

type *byte* **is** dependiente de la implementación;

Elemento del que están formados los mensajes.

type *tipo_nombre* **is new** string (1..*long_nombre_de_cola*);

Define el nombre de la cola.

type *tipo_prioridad* **is new** positive **range** dependiente de la implementación;

Define el rango de prioridades que se asignan a los mensajes.

type *tipo_acceso* **is** (*lectura*, *escritura*, *lectura_escritura*);

Define el modo de acceso de las tareas a las colas de mensajes.

type *tipo_mensaje* **is array** (1..*max_longitud_de_mensaje*) **of** *byte*;

Mensaje utilizado por las colas de mensajes.

type *tipo_atributos* **is**
record

numero_mensajes : positive;
Número máximo de mensajes en la cola.

longitud_mensaje : positive;
Longitud máxima de un mensaje de la cola.

mensajes_en_cola : integer;
Número actual de mensajes encolados

global : boolean;
Indica si la cola es o no global.

end record;

Definición de los atributos característicos de las colas de mensajes.

```
type tipo_error is
record
        envia       : integer;
        cierra      : integer;
        interno     : integer;
end record;
```

Definición del tipo de error que se puede producir en las operaciones remotas *envía* y *cierra*, al no ser bloqueantes y no poder propagar una excepción. Como error *interno* entendemos aquel que se produce en la recepción de mensajes y en la ejecución de operaciones remotas. Los campos del registro inicialmente valen cero, y van aumentando en una unidad con los errores producidos.

- En el paquete *configuración*:

  type *tipo_nudo* **is** dependiente de la implementación;

  Identificador de nudo dentro del sistema distribuido.

  type *tipo_red* **is** dependiente de la implementación;

  Identificador de red dentro del sistema distribuido.

- En el paquete *colas_de_mensajes*:

  type *tipo_descriptor* **is limited private**;

  Descriptor con el que las tareas operan sobre las colas de mensajes.

3.- Excepciones.

- En el paquete *tipos_de_colas*:

  cola_ya_creada

  Ocurre cuando se intenta crear una cola que ya está creada.

  cola_perteneciente_a_otro_nudo

  Si la cola pertenece a otro nudo no se pueden realizar las operaciones de creación y destrucción de la cola.

  cola_en_proceso_de_destruccion

  Sucede cuando hay una notificación de destrucción sobre la cola y se intenta realizar una de las operaciones *abre*, *destruye* o *envía*. También elevan esta excepción las operaciones *recibe* y *recibe si hay*, en el caso de que además la cola esté vacía.

numero_de_mensajes_fuera_de_rango

Cuando se crea una cola el número de mensajes debe estar dentro del rango permitido.

longitud_de_mensaje_fuera_de_rango

Cuando se crea una cola o se envía un mensaje la longitud del mensaje debe estar dentro del rango permitido.

no_hay_colas_libres

Indica que no se pueden crear más colas por haberse alcanzado el máximo permitido en el nudo.

no_hay_descriptores_libres

Indica que no se pueden abrir más descriptores, debido a que se ha agotado el número de locales o remotos de que dispone cada nudo.

descriptor_no_activo

El descriptor que se utiliza en una operación determinada no está activo, es decir no ha sido abierto.

cola_no_existente

La cola que se quiere destruir no existe.

insuficiente_espacio_en_la_cola

Ocurre cuando se intenta enviar un mensaje y no hay espacio disponible en la cola, por haberse alcanzado el número máximo de mensajes con que se creó.

acceso_no_permitido

El acceso establecido para ese descriptor no permite que se realice la operación de envío o recepción que se solicita.

problemas_de_comunicacion

Hay algún problema en alguna operación remota debido al uso de las redes de comunicación.

- En el paquete *configuracion*:

lista_completa

Se produce en una operación de configuración del nudo cuando se sobrepasa el número de elementos configurables.

La parte de la estructura de datos de las colas de mensajes que no es visible para los programas de usuario comprende toda la estructuración y el soporte físico de las colas de mensajes, y de la memoria que éstas y los mensajes de comunicaciones utilizan. La implementación de esta estructura se muestra en el apartado A.2 del apéndice A.

Para la definición de la estructura de datos se ha tenido en cuenta que la implementación de las redes de comunicación debe responder a requerimientos de tiempo real estricto, y por lo tanto, las operaciones que se ejecutan sobre ellas deben ser absolutamente deterministas y modelables desde un punto de vista temporal. Así, se puede observar que en la estructura de datos las listas no se implementan con punteros sino con índices, y la gestión de la memoria que se utiliza, tanto en las colas de mensajes, como en las comunicaciones, está totalmente controlada.

Los detalles más importantes de esta parte de la estructura de datos se pueden condensar en los puntos siguientes:

1.- Espacio de memoria y espacio de mensajes.

El espacio de memoria se define como un conjunto de segmentos de memoria de una longitud determinada. En el momento de la creación de una cola de mensajes se reserva el espacio de memoria (cantidad de segmentos) necesario para almacenar el número de mensajes que admite la cola. La manipulación de este conjunto de segmentos se controla mediante una lista en la que se anotan los que están libres.

El espacio de mensajes se define como un conjunto de mensajes de comunicación que se utilizan en las operaciones remotas para el intercambio de información entre los distintos nudos. Aunque la descripción exhaustiva de las partes de que se componen los mensajes de comunicaciones se realizará más adelante (protocolo de comunicaciones en este mismo capítulo, y modelado de las comunicaciones en el capítulo 7), vamos a dar a continuación una breve descripción de las partes de que se componen estos mensajes del espacio de mensajes:

Cabeza. Espacio en el que reside la información necesaria para la partición del mensaje en paquetes de comunicación, y su posterior reconstrucción a partir de estos paquetes.

Protocolo. Espacio que se reserva para la información relativa a la operación remota para la que se utiliza el mensaje de comunicación (tipo de operación, parámetros de la operación, nudos origen y destino, etc). Esta información va codificada.

Mensaje. Es el espacio reservado para los mensajes propios de las colas, o para la codificación del nombre de una cola de mensajes en el caso de una operación *abre* remota.

Longitud del último paquete. Información correspondiente a la longitud del último paquete. Se obtiene de manera sencilla en la partición del mensaje en paquetes y se guarda con motivo de evitar su cálculo en el momento de la transmisión del último paquete (más costoso y ejecutado a la prioridad de las comunicaciones).

Al igual que el espacio de memoria la manipulación de los mensajes de comunicación también se controla mediante una lista donde se anotan los que están libres.

2.- Descriptores.

Definimos el descriptor, no sólo como un índice que apunta a una cola determinada, sino que también le asociamos toda la información relacionada con el acceso a la cola, con la conexión hacia o desde un nudo remoto, y con la tarea propietaria (en orden a realizar su suspensión o activación).

La información asociada al descriptor depende del nudo en el que se encuentre la cola a la que referencia. De acuerdo con esto, vamos a distinguir dos tipos de descriptores:

Descriptor local. Es el que utiliza una tarea para apuntar a una cola de mensajes local o global del nudo actual (la cola y la tarea que utiliza el descriptor están localizados en el mismo nudo). La información que contiene este descriptor es la siguiente:

Indice a nombre. Apunta a la cola de mensajes a la que identifica.

Indice a descriptor. Descriptor remoto de un nudo remoto cuando una tarea de ese nudo remoto ha abierto este descriptor local, que por supuesto apunta a una cola global.

Acceso. Modo de acceso permitido sobre la cola a la que apunta.

Parámetros de suspensión. Estado de suspensión referido a la operación *recibe* efectuada por una tarea en el nudo local, o por una tarea en un nudo remoto a través de la red. Además para una operación local, la información relativa al control de la suspensión y activación de tareas.

Descriptor remoto. Es el que utiliza una tarea para apuntar a una cola de mensajes global de un nudo remoto (la cola y la tarea que utiliza el descriptor están localizados en nudos diferentes). La información que contiene este descriptor es la siguiente:

Indice a descriptor. Descriptor local en el nudo remoto que identifica a la tarea global a la que se conecta el descriptor remoto.

Nudo. Identificación del nudo en el que se encuentra la cola global.

Red. Identificación de la red por la que se accede a la cola global.

Acceso. Modo de acceso permitido sobre la cola a la que apunta.

Parámetros de suspensión. Estado de suspensión referido a las operaciones remotas en espera de respuesta, e información relativa al control de la suspensión y activación de tareas.

Los descriptores pertenecientes a cada uno de los dos tipos se gestionan mediante sendas listas en las que se encuentran anotados los libres (no están siendo utilizados). Un descriptor remoto (utilizado por una tarea en el nudo local para acceder a una cola global en un nudo remoto) se traduce a un descriptor local en el nudo remoto.

3.- Colas de mensajes.

Bajo este concepto se engloba toda la información relativa a la estructura de las colas de mensajes y al control de los mensajes contenidos en las mismas. La información que soporta esta estructura es la siguiente:

Cola en destrucción. Indica si la cola está en proceso de destrucción.

Nombre de la cola.

Atributos. Atributos definidos para las colas de mensajes.

Número de descriptores abiertos. Número de descriptores que apuntan a la cola.

Número y lista de segmentos de memoria. Segmentos de memoria reservados en el momento de la creación de la cola.

Cola de prioridad. Estructura de la cola de prioridad en la que se almacenan los mensajes. Está formada por un número de colas igual al de prioridades del sistema. Los índices a cada una de estas colas se encuentran en *lista de prioridades*. *Lista de mensajes* contiene elementos con información sobre la localización de los mensajes que puede almacenar la cola en los segmentos de memoria que se han reservado.

Lista de mensajes libres. Lista de los mensajes de la cola que no se están usando.

También las colas de mensajes se definen como un conjunto limitado de elementos con la estructura definida y su uso se controla mediante una lista con los elementos que están libres.

4.- Colas de espera.

Definición de los elementos de cola de espera que contienen la información necesaria para realizar una espera a una operación *recibe*. Para la cola de espera local, la única información necesaria es el *descriptor* (local), ya que a través de él se puede acceder a toda la información necesaria para la activación de nuevo de la tarea suspendida. La cola de espera remota necesita además la *red* y el *mensaje de comunicaciones*, a través de los que se canaliza la operación remota.

La cola de espera local se define como una cola de prioridad, de una forma similar a la utilizada en las colas de mensajes, donde la prioridad es la definida para las tareas del nudo en el que se implementa.

La cola de espera remota se define como una lista enlazada que se gestiona como una cola FIFO.

5.3.3. Implementación de las operaciones

En líneas generales vamos a considerar aquellas operaciones de que disponen los programas de usuario para manipular las colas de mensajes, obtener los errores producidos en esta manipulación, y configurar adecuadamente los nudos (en cuanto a la localización de las colas globales y acceso a través de una red de comunicaciones). Sin embargo, dada la complejidad que subyace detrás de las operaciones sobre las colas de mensajes debido a su carácter distribuido, y que los accesos a algunos datos requieren exclusión mutua, vamos a considerar también algunas operaciones más, relacionadas sobre todo, con la ejecución de operaciones remotas en el nudo remoto, y con la gestión de la memoria y de los mensajes de comunicación.

Así pues, vamos a dar una clasificación de estas operaciones atendiendo a la funcionalidad de los procedimientos, funciones y tareas que las implementan:

1.- Gestión de la memoria.

La gestión de la memoria se realiza a través de los procedimientos *reserva_segmentos(…)* y *libera_segmentos(…)* del paquete *memoria*, cuyas especificaciones se encuentran en el apartado A.1 de apéndice A. Los datos sobre los que operan estos procedimientos no necesitan ser protegidos de accesos concurrentes, debido a que van a ser utilizados únicamente por las operaciones de las colas de mensajes *abre* (en su opción de crear la cola), *cierra* (cuando debe destruir la cola) y *destruye*, que ya garantizan la exclusión mutua.

En el intento de reserva de segmentos se puede producir la excepción *no_hay_espacio_de_memoria* cuando no quedan segmentos suficientes.

2.- Gestión de los mensajes de comunicaciones.

La gestión de los mensajes de comunicaciones se realiza a través de los procedimientos *reserva_espacio_mensaje(…)* y *libera_espacio_mensaje(…)* del paquete *memoria*. Los datos sobre los que operan estos procedimientos necesitan ser protegidos para garantizar la exclusión mutua en el acceso, ya que van a ser manipulados por diferentes tareas.

La manera en que se garantiza en Ada83 la exclusión mutua en el acceso a los datos es utilizar una tarea con un punto de entrada por cada operación sobre los datos, y una sentencia **select** para gestionar las llamadas a los puntos de entrada. Así, para el control de las operaciones sobre el espacio de mensajes disponemos de la tarea *maneja_recursos*, cuya especificación y pseudocódigo se encuentran en el apartado A.3 del apéndice A.

En el intento de reserva de mensaje de comunicaciones se puede producir la excepción *no_hay_espacio_de_mensajes*, cuando no quedan mensajes libres.

3.- Configuración.

La configuración de un nudo consiste en la declaración de las colas globales, de los nudos en los que están localizadas, de las redes mediante las que se accede a ellas, y finalmente del nudo actual. Para realizar estas operaciones, el usuario tiene a su disposición en el paquete *configuracion* los siguientes procedimientos: *configura_nudo(…)* (establece la localización de las colas globales y las redes a través de las que se accede a ellas), y *establece_nudo_actual(…)* (establece el identificador del nudo actual).

La consulta de la configuración necesaria para realizar algunas de las operaciones sobre las colas se hace mediante los procedimientos *obten_nudo(…)* (nudo en el que está localizada una cola global determinada), *obten_red(…)* (consulta de la red por la que se accede a una cola global determinada), y la función *obten_nudo_actual* (devuelve el identificador del nudo actual).

Las especificaciones de estas operaciones se encuentran en el apartado A.1 del apéndice A, dentro del paquete *configuracion*.

4.- Control de recursos.

En este punto vamos a considerar el control de los recursos disponibles, relacionados con las colas de mensajes y con los descriptores. La utilización de una forma coherente de estos datos requiere que las operaciones que se realicen sobre ellos, los manejen de forma mutuamente exclusiva.

Por otro lado, podemos hacer una distinción entre los recursos de ámbito local, constituidos por las colas de mensajes (cuyos datos se manipulan siempre de forma local) y los descriptores locales, y los recursos de ámbito remoto, que son únicamente los descriptores remotos. La manipulación de los recursos de ámbito local no necesita ser mutuamente excluyente con la de los de ámbito remoto.

De acuerdo con lo expuesto, necesitamos controlar separadamente la exclusión mutua de los accesos a los recursos locales y a los remotos, para lo cual, definimos las siguientes tareas:

Sincroniza. Garantiza exclusión mutua en el acceso a los recursos locales. Dado que esto significa la manipulación de todos los datos referidos a las colas de mensajes, esta tarea pasa por ser realmente el centro de control de la ejecución de todas las operaciones sobre las colas de mensajes, tanto locales, como remotas.

Recursos_remotos. Garantiza exclusión mutua en las operaciones de apertura y cierre de descriptores remotos.

Ambas tareas pertenecen al paquete *control_de_colas* y sus especificaciones y pseudocódigos se muestran respectivamente en los apartados A.1 y A.3 del apéndice A.

5.- Operaciones locales.

Como operaciones locales consideramos las que ordena el usuario sobre una cola de mensajes del nudo actual. Estas operaciones se realizan mediante los procedimientos *abre(…), cierra(…), destruye(…), envia(…), recibe(…), recibe_si_hay(…)* y *coge_atributos(…),* del paquete *colas_de_mensajes*.

En todos estos procedimientos, después de la detección de que se trata de una operación local y del chequeo de algunas excepciones que se puedan producir, se transfiere el control a la tarea *control_de_colas.sincroniza*.

Las especificaciones y pseudocódigos de todas estas operaciones se encuentran en los apartados A.1 y A.3 del apéndice A.

6.- Operaciones remotas en el nudo actual.

Son las partes de las operaciones remotas sobre las colas de mensajes que se ejecutan en el nudo actual. Estas operaciones se realizan mediante los procedimientos *abre(…)* (no es válida la opción crear), *cierra(…)*, *envia(…)*, *recibe(…)*, *recibe_si_hay(…)* y *coge_atributos(…)*, del paquete *colas_de_-mensajes*.

Al igual que en el caso de las operaciones locales, después de la detección de que se trata de una operación remota y del chequeo de algunas excepciones que se puedan producir, se compone el mensaje apropiado, se transmite por la red, y se suspende la tarea que los ha llamado, salvo en el caso de las operaciones *cierra* y *envía* que no esperan respuesta. Cuando la tarea es despertada porque se ha recibido la respuesta a la operación remota, la interpretan y actúan en consecuencia.

7.- Operaciones remotas en el nudo remoto.

La operación remota en el nudo remoto es la ejecución efectiva de la operación remota, es decir, la actuación real sobre la cola de mensajes global a que se refiere la operación.

Las operaciones remotas en el nudo remoto se inician en la red a partir de la recepción de un mensaje completo (describiremos el mecanismo en el apartado de comunicaciones). La red canaliza los mensajes recibidos a través del procedimiento *recibe(…)* del paquete *control_de_colas*. Estos mensajes de comunicaciones pertenecen al espacio de mensajes descrito en este capítulo (ver página 5-21).

Cuando se interpreta el protocolo que llevan asociado estos mensajes (ver página 5-32) y se trata de una solicitud para realizar una operación remota se llama al punto de entrada *ejecuta_remoto(…)* de la tarea *control_de_co-las.sincroniza*, que es la que se encarga de la ejecución de la operación remota a través de los procedimientos *abre_remoto(…)*, *cierra_remoto(…)*, *envia_re-*

moto(…), recibe_remoto(…), recibe_si_hay_remoto(…) y *coge_atributos_remoto(…)* localizados en el cuerpo del paquete *control_de_colas*.

Si al interpretar el protocolo no es una solicitud de operación remota, puede tratarse de una respuesta, en cuyo caso despertará a la tarea que estaba suspendida en espera, o de un error de las operaciones remotas *cierra* o *envia*, que será oportunamente anotado. Si se produce la excepción *error_de_comunicaciones* como resultado del uso de la red de transmisión, se anota un error *interno* en la variable de error.

Las especificaciones y pseudocódigos de todas estas operaciones se encuentran en los apartados A.1 y A.3 del apéndice A.

8.- Chequeo de errores.

Consulta de los errores producidos en las operaciones remotas *cierra* y *envia* (en el nudo actual), y en la recepción y ejecución de las operaciones remotas (en el nudo remoto). Se implementa mediante la función *obten_error* del paquete *colas_de_mensajes*, que actúa como interfase al usuario, y la función *errores_remotos* del paquete *control_de_colas* para el acceso a la información de error. Cada vez que se realiza una consulta de los errores producidos se reinicializan las variables de error. El pseudocódigo de estas funciones se muestra en el apartado A.3 del apéndice A.

Las operaciones implementadas cumplen las especificaciones definidas para las colas de mensajes, que unidas a una implementación adecuada de las redes de comunicación, configuran en su conjunto una plataforma de comunicaciones para sistemas distribuidos con requerimientos de tiempo real estricto.

El flujo de control de las operaciones que se realizan sobre las colas de mensajes, ya ha sido descrito en líneas generales en este apartado, pero se muestra de una forma más concreta en el pseudocódigo de los procedimientos, funciones y tareas que las implementan (apéndice A), y se pondrá además de manifiesto más adelante, en el capítulo dedicado al modelado de las redes de comunicación.

5.3.4. Aspectos dependientes de la implementación

Para definir lo que entendemos como aspectos dependientes de la implementación vamos a dar una clasificación de éstos en dos puntos:

Parámetros de configuración. Son constantes o tipos, cuyos valores o rangos quedan indeterminados y dependen de la aplicación concreta. Estos parámetros no afectan a la definición de la estructura de datos ni a la de las operaciones que se realizan sobre ellos, y se determinan en el desarrollo de una aplicación determinada.

Limitaciones del lenguaje de programación. Son aquellos puntos de la implementación en los que se recurre a extensiones del lenguaje o a otro tipo de soluciones que se apartan del estándar del lenguaje. Estos aspectos afectan considerablemente a la portabilidad de la implementación y pueden dar lugar a importantes cambios estructurales de la misma.

El primer punto está bien resuelto en la implementación realizada, gracias a la facilidad que presenta el Ada para la definición estructurada de la información que se requiere en este tipo de implementaciones, y la manipulación de la misma.

Con respecto al segundo punto, aunque la implementación realizada de las colas de mensajes se puede considerar general en cuanto a la definición de la estructura de datos, y de las especificaciones de los paquetes Ada tanto de interfase hacia el usuario, como de los otros paquetes de la implementación, existen algunos puntos concretos que no están resueltos en el Ada83 y por lo tanto, se ha tenido que recurrir a las extensiones del compilador utilizado.

Este es el caso de la suspensión de tareas, en el que se recurre a la extensión de prioridades dinámicas del compilador utilizado. La suspensión de una tarea se realiza cambiando su prioridad a la *prioridad de suspensión*, que se elige como la prioridad más baja del sistema y se reserva para la suspensión, estableciendo su estado de suspensión y haciéndole ejecutar un lazo infinito en el que chequea este estado. La activación de la tarea se realiza estableciendo la prioridad que tenía y cambiando su estado a no suspendida.

Afortunadamente, en el anexo de sistemas de tiempo real del manual de referencia del Ada95 [ADA95], encontramos la solución al problema de la suspensión de tareas. Así, es posible mantener el sistema de suspensión utilizado en esta implementación, basado en prioridades dinámicas (paquete *Ada.Dynamic_Prioriries*), o utilizar el paquete *Ada.Asynchronous_Task_Control*, en el que hay procedimientos para suspender a una tarea, hacer que continúe, o bien verificar si está suspendida. En cualquier caso, la solución elegida es completamente estándar.

Más aún, en Ada95 podemos cambiar las tareas que controlan la exclusión mutua en el acceso a recursos por objetos protegidos, que definan los datos y las operaciones protegidas que se realicen sobre ellos, y además, elegir la política de planificación más adecuada. Podemos conseguir así una implementación mucho más eficiente (la ejecución de un procedimiento protegido no implica un cambio de contexto, la tarea sí), y elegante (los objetos protegidos hacen más natural la definición estructurada de los datos).

5.4. Implementación de las comunicaciones de tiempo real

Se han definido e implementado las colas de mensajes como mecanismo de comunicación entre tareas en un sistema distribuido con requerimientos de tiempo real estricto. Para implementar el sistema de comunicaciones completo es necesario dotar a las colas de mensajes de redes de comunicación adecuadas al sistema distribuido que se pretende desarrollar. En la implementación de las redes de comunicación hay que tener en cuenta que deben cumplir requisitos de tiempo real, y por lo tanto, aparece una cierta dificultad en el uso de redes convencionales. Para nuestra implementación elegimos una transmisión con planificación por prioridades y expulsora (por paquetes de comunicación).

Con la plataforma de desarrollo que hemos elegido (tarjetas procesadoras sobre bus VME con líneas serie RS-232 y RS-485), vamos a definir una implementación de las comunicaciones con carácter general para líneas punto a punto y para el bus VME, aunque algunos paquetes Ada de la misma, serán válidos para cualquier otra red de comunicaciones.

Tal como dijimos al describir la estructura general de la implementación de las redes de comunicación, el punto de conexión de las comunicaciones y de las colas de mensajes

definidas es la realización de las operaciones remotas. Estas operaciones se canalizan hacia o desde los paquetes Ada en los que se implementan las comunicaciones, a través de los paquetes *red_transmision* y *red_recepcion*.

Además, las operaciones remotas se codifican de acuerdo con un protocolo definido, al que se añade una componente de red, para realizar la partición de mensajes en paquetes de comunicación. En la implementación realizada, los protocolos utilizados son conocidos por las diferentes etapas de la comunicación con objeto de dinamizar ésta, y disminuir su coste temporal al evitar sucesivas copias de los mensajes entre las etapas. Este dinamismo en la manipulación de los mensajes se ve afianzado por la definición del espacio de mensajes en el paquete *memoria* del que ya hemos hablado (ver página 5-21).

La Figura 5-2 muestra la estructura de la implementación realizada de las comunicaciones, junto con el flujo de los mensajes de comunicación a través de esta estructura. Se destacan también las colas de prioridad que se utilizan para almacenar los paquetes de comunicación hasta su transmisión, y los mensajes de comunicación después de su recepción.

A continuación, vamos a realizar la descripción de las comunicaciones comenzando con el protocolo de comunicaciones, continuando después con la gestión de las redes, y finalizando con la implementación de las comunicaciones en líneas punto a punto y en el bus VME.

5.4.1. Protocolo de comunicaciones

Cada mensaje de comunicaciones está formado por el protocolo de codificación de las operaciones remotas (*protocolo de mensaje*) más el mensaje propio de las colas de mensajes (que puede contener un mensaje del usuario o información adicional de alguna operación concreta). Además, los mensajes de comunicaciones se parten en paquetes de comunicaciones de longitud fija para ser transmitidos, y a cada paquete se le añade una *cabecera de paquete* con la información necesaria tanto para la reconstrucción del mensaje, como para la reutilización de los mensajes del espacio de mensajes (el mismo mensaje que se reserva para la solicitud de ejecución de una operación se utiliza después en su respuesta sin necesidad de liberarlo). Por lo tanto, para definir el protocolo de comunicaciones vamos a diferenciar la

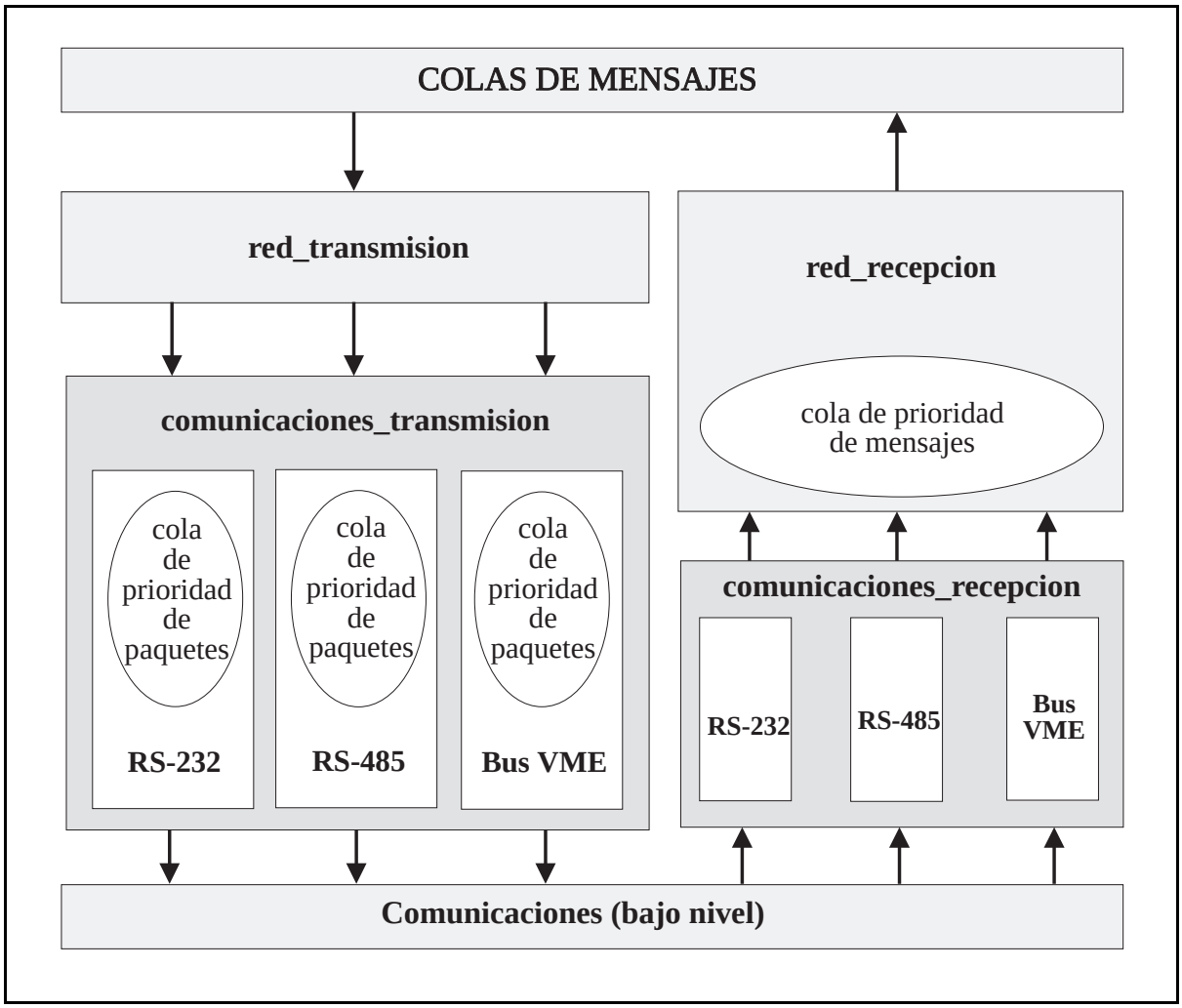

Figura 5-2. Implementación de las comunicaciones.

parte correspondiente al *protocolo de mensaje*, y la correspondiente a la *cabecera de paquete*. Ambas se ven reflejadas en el tipo *tipo_espacio_de_mensajes* del paquete *memoria* (apartado A.1 del apéndice A). La Figura 5-3 muestra la estructura del mensaje de comunicaciones y su partición en paquetes de comunicación.

El *protocolo de mensaje* debe reflejar de manera exacta las características particulares de cada operación remota. Desde el punto de vista de las comunicaciones se puede considerar que las operaciones remotas se realizan en las dos fases siguientes:

Solicitud de ejecución. Cuando se solicita la ejecución de cualquier operación remota sobre una cola de mensajes, se genera un mensaje de comunicaciones dirigido hacia el nudo remoto en el que se va a ejecutar la operación.

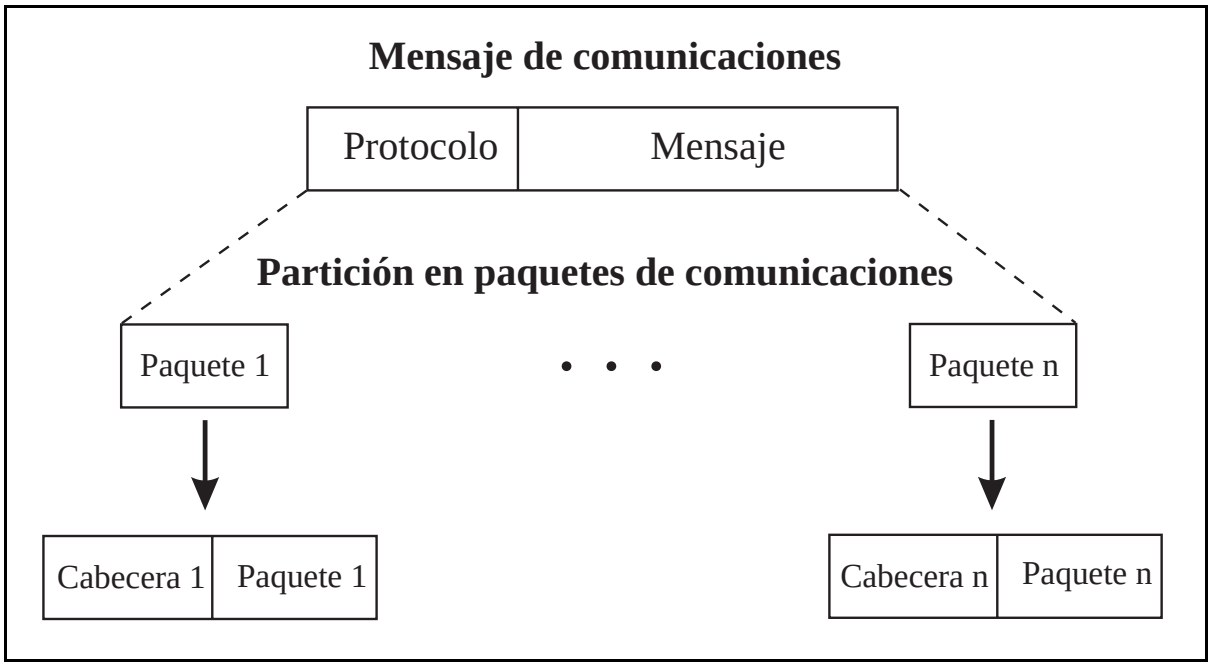

Figura 5-3. Partición de los mensajes de comunicaciones.

Respuesta o error. Después de la ejecución de una operación remota sobre una cola de mensajes en el nudo remoto, se genera un mensaje de comunicaciones dirigido hacia el nudo actual que contiene la respuesta a la operación o el error según proceda en cada caso.

Así pues, para describir el *protocolo de mensaje*, hemos de considerar cada una de las operaciones remotas, teniendo en cuenta la información que se debe enviar codificada en sus fases de solicitud de realización y de respuesta o error. La información que se necesita codificar en cada caso es la que se muestra a continuación (la primera fila pertenece a la solicitud de ejecución, la segunda a la respuesta con éxito o al error, y *XXX* indica no utilizado):

1.- *Abre remoto*.

Solicitud de ejecución	Operación *abre*	Longitud (long. nombre cola)	Prioridad	Descriptor (remoto)
Respuesta / Error	Operación *abre*	Descriptor (local)	Prioridad	Descriptor (remoto)

A esta operación se le asocia un mensaje con el nombre codificado de la cola de mensajes global que se quiere abrir. La longitud se corresponde con la de ese mensaje. El descriptor local corresponde al nudo remoto (obtenido de la apertura de la cola), y el descriptor remoto al nudo actual (descriptor que identifica en el nudo actual a una cola un nudo remoto). La prioridad es la utilizada en las colas de mensajes y en ambos casos es la máxima.

2.- *Cierra remoto*.

Solicitud de ejecución	Operación *cierra*	XXX	Prioridad	Descriptor (local)
Error	Operación *cierra*	XXX	Prioridad	XXX

El descriptor local es el que se quiere cerrar en el nudo remoto. La prioridad en ambos casos es la máxima.

3.- *Envía remoto*.

Solicitud de ejecución	Operación *envía*	Longitud	Prioridad	Descriptor (local)
Error	Operación *envía*	XXX	Prioridad	Descriptor (remoto)

La prioridad de la solicitud de ejecución es la del mensaje que se quiere enviar, ya que de esta forma, por un lado se mantiene la coherencia del mecanismo de prioridades utilizado por las colas de mensajes por un lado, y por otro se da entidad al mecanismo de planificación por prioridades y expulsor por paquetes de las comunicaciones. La prioridad en caso de error es la máxima (en otro caso la operación no genera respuesta). La longitud es la del mensaje que se envía a la cola. El descriptor local corresponde al nudo remoto, y el descriptor remoto al nudo actual.

4.- *Recibe remoto*.

Solicitud de ejecución	Operación *recibe*	XXX	Prioridad	Descriptor (local)
Respuesta / Error	Operación *recibe*	Longitud	Prioridad	Descriptor (remoto)

La prioridad de la respuesta con el mensaje a recibir es la correspondiente al propio mensaje. Así, la transmisión real del mensaje de la cola de mensajes se realiza a la prioridad del propio mensaje, manteniendo la coherencia del mecanismo de prioridades utilizado por las colas de mensajes. La prioridad de la solicitud de ejecución o del error en la respuesta es la máxima. La longitud es la del mensaje que se recibe y los descriptores son como en los casos anteriores.

5.- *Recibe si hay remoto*.

Solicitud de ejecución	Operación *recibe si hay*	XXX	Prioridad	Descriptor (local)
Respuesta / Error / No hay	Operación *recibe si hay*	Longitud	Prioridad	Descriptor (remoto)

Esta operación se codifica como la anterior, pero se añade la opción de *no hay* mensaje como respuesta no errónea.

6.- *Coge atributos remoto*.

Solicitud de ejecución	Operación *coge atributos*	XXX	Prioridad	Descriptor (local)
Respuesta / Error	Operación *coge atributos*	Longitud	Prioridad	Descriptor (remoto)

La prioridad que se utiliza es la máxima en ambos casos, la longitud se refiere a la del mensaje con la codificación de los atributos y los descriptores son como en los casos anteriores.

Todas las operaciones salvo las que conllevan la transmisión de un mensaje de las colas se realizan a la prioridad máxima para evitar inversiones de prioridad. Una solicitud de recepción de un mensaje que tiene muy alta prioridad podría tener que esperar la transmisión de otros de prioridad menor. En todas las operaciones la prioridad que se utiliza se corresponde con la que se va a asignar a los paquetes de comunicación que se van a transmitir. Además, la información de error codificada en las respuestas se toma como punto de partida para propagar las excepciones oportunas.

En cuanto a la *cabecera de paquete* cabe decir que se manda con cada paquete de comunicaciones, y se va modificando para poder controlar la partición de paquetes en la transmisión del mensaje y su posterior recomposición en la recepción. Pero además, lleva información sobre la identidad de los mensajes reservados en el espacio de mensajes con el fin de poder utilizarlos tanto en la orden de operación remota, como en la respuesta, y así, evitar operaciones innecesarias (de reserva y liberación de mensajes) mucho más costosas en tiempo de CPU que una simple consulta de identificación.

La información que debe llevar la *cabecera de paquete* es la que se muestra a continuación:

Nudo destino	*Nudo origen*	*Identificador de mensaje*	*Número de paquete*	*Número total de paquetes*

Nudo destino. Nudo al que va dirigido el paquete de comunicaciones.

Nudo origen. Nudo de partida del paquete de comunicaciones.

Identificador de mensaje. Identificación del mensaje de comunicaciones reservado en el espacio de mensajes del nudo de partida cuando se solicita la ejecución de una operación sobre una cola de mensajes. Este mismo mensaje del espacio de mensajes será utilizado para la recomposición del mensaje en la respuesta a la operación solicitada.

Número de paquete. Número de orden del paquete de comunicaciones que se transmite, dentro del total de paquetes en que se divide el mensaje.

Número total de paquetes. Número total de paquetes de comunicación de que se compone el mensaje del que forma parte el paquete que se transmite.

En la partición del mensaje hay que tener en cuenta que el *protocolo de mensaje* forma parte del primer paquete de comunicaciones que se transmite (ver Figura 5-3 en la página 5-34). La definición del protocolo utilizado en la implementación que hemos realizado se encuentra en el paquete *protocolo*, cuya especificación se muestra en el apartado A.1 del apéndice A.

5.4.2. Gestión de redes

Por gestión de redes entendemos la partición de los mensajes que se quieren transmitir, su canalización hacia la red de comunicaciones adecuada, y el almacenamiento y posterior tramitación de los que se reciben por las diferentes redes de comunicación.

Para describir la implementación realizada de la gestión de las redes vamos a considerar los paquetes *red_transmision* y *red_recepcion* en los que ésta se realiza.

La transmisión de los mensajes generados a partir de las órdenes de operación remota, o de sus respuestas, se realiza mediante el procedimiento *transmite(…)*, localizado en el paquete *red_transmision*. Este procedimiento se encarga por un lado de la partición del mensaje en paquetes de comunicaciones, estableciendo de forma adecuada sus cabeceras, y por otro, del posterior encolado de los paquetes en la cola de transmisión de la red oportuna, con objeto de que sea finalmente transmitido. Este procedimiento propaga la excepción *no_hay_espacio_en_la_cola*, que se produce al manejar la correspondiente, elevada por las comunicaciones de transmisión.

La gestión de red en cuanto a la recepción satisface dos importantes requisitos: por un lado, se encarga del encolado de todos los mensajes recibidos con independencia de la red por la que lleguen, y por otro, de la tramitación de estos mensajes hacia el control de las colas de mensajes.

La tramitación de los mensajes la realiza la tarea *ordena_recibe* del cuerpo del paquete *red_recepcion*. Esta tarea va sacando los mensajes recibidos de la cola donde se van depositando, y los va procesando haciendo uso del procedimiento *control_de_colas.recibe*, que es el encargado de la ejecución de las operaciones remotas y de sus respuestas.

Para el encolado de los mensajes de comunicaciones en su recepción se implementa una cola de prioridad, del estilo de las colas de mensajes, y con la misma definición de prioridades. Con esta implementación se mantiene la coherencia de las prioridades utilizadas por las colas de mensajes también en la gestión de los mensajes recibidos. Las operaciones de encolar y desencolar mensaje son controladas mediante la tarea *control_recibe* del cuerpo del paquete *red_recepcion*. Esta tarea es necesaria para asegurar la exclusión mutua en las operaciones sobre la cola, que es utilizada tanto por las tareas de comunicación de las diferentes redes (*encola*, a través del procedimiento de interfase del mismo nombre), como por la tarea *ordena_recibe* (*desencola*).

Las especificaciones de los paquetes *red_transmision* y *red_recepcion*, así como los pseudocódigos de los procedimientos y tareas mencionados se encuentran respectivamente en los apartados A.1 y A.4 del apéndice A.

La implementación realizada resuelve de una manera general la gestión de las redes, aunque por supuesto, el software del paquete *red_transmision* depende de la arquitectura del sistema distribuido en el que se quiera utilizar. La implementación de las colas de prioridad y de su control, al igual que ocurre en las colas de mensajes, se puede mejorar sensiblemente si se define como un objeto protegido del Ada95.

5.4.3. Comunicaciones mediante líneas punto a punto

En este apartado, vamos a describir la implementación de las comunicaciones de tiempo real, para líneas serie RS-232 y RS-485 utilizadas como líneas punto a punto. Toda la implementación está hecha en Ada, salvo un módulo de la parte correspondiente a la rutina de atención de interrupción que ha sido realizada en ensamblador.

En la implementación de las comunicaciones se utiliza un mecanismo de planificación basado en mensajes con prioridad que se dividen en paquetes de longitud fija. Cada mensaje lleva su prioridad dentro del rango de la red, y todos los paquetes en que se dividen llevan esa misma prioridad. El planificador de red tiene en cada nudo una cola de prioridad de paquetes que van a ser transmitidos. Un paquete individual no es expulsable, es decir, una vez que ha comenzado su transmisión ningún otro mensaje puede interrumpirla, pero los paquetes de alta prioridad siempre se transmiten primero que los de baja prioridad a partir de la cola del planificador. La Figura 5-4 muestra la interfase del planificador que se utiliza en el marco de unas comunicaciones definidas mediante la estructura de niveles OSI. En esta figura, el nivel de transporte se corresponde con lo que hemos denominado gestión de redes, y los niveles de red y de enlace (junto con el físico que no aparece en la figura) son los que vamos a describir en este apartado.

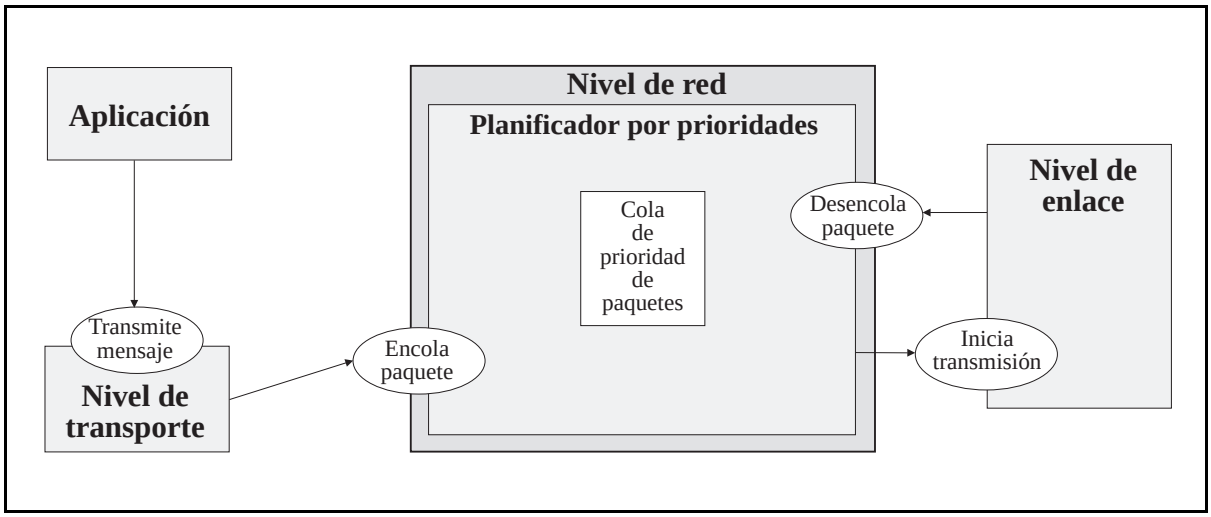

Figura 5-4. Planificador de mensajes por prioridades.

Los puntos que consideramos importantes en la definición del software de comunicaciones para líneas punto a punto son los siguientes:

1.- Protocolo.

El protocolo que se utiliza para la transmisión de paquetes de comunicación es el envío de un caracter de comienzo de mensaje, *STX*, y un caracter de final de mensaje, *ETX*, definidos ambos en el paquete *protocolo*.

2.- Transmisión.

La transmisión se implementa mediante los paquetes *comunicaciones_transmision_232/485*, en cuya interfase aparecen los procedimientos *transmite* y *ordena_transmision*, y la excepción *no_hay_espacio_en_la_cola*.

El procedimiento *transmite* se encarga de introducir el mensaje de comunicaciones ya partido en paquetes en la cola de transmisión, con los valores adecuados de las *cabeceras de paquete*. *Ordena_transmision* es el procedimiento a través del cual la tarea de interrupción inicia la transmisión de un nuevo paquete de comunicaciones.

La cola de transmisión se implementa como una cola de prioridad, con las prioridades definidas para las colas de mensajes. La gestión de esta cola la realiza la tarea *sincroniza_transmite*, que ejecuta las operaciones de encolado de paquetes (punto de entrada *transmite*) y desencolado para la transmisión (punto de entrada *ordena_transmision*), de forma que se garantiza su exclusión mutua. La operación de desencolado realiza la copia del paquete en el *buffer de transmisión*.

La transmisión de un paquete de comunicaciones la inicia *sincroniza_transmite.transmite* (en el caso de que la transmisión estuviese parada), y *sincroniza_transmite.ordena_transmision* (si la transmisión está activa). En el caso de las líneas serie implementadas, iniciar la transmisión de un paquete significa enviar el primer caracter, es decir, el *STX*.

3.- Recepción.

La recepción se implementa mediante el procedimiento *recibe* localizado en los paquetes *comunicaciones_recepcion_232/485*. Este procedimiento recoge el paquete de comunicaciones del *buffer de recepción* y se lo incorpora al mensaje adecuado. En el caso de que sea el último paquete lo introduce en la cola de *red_recepcion*, haciendo uso del procedimiento *encola*.

La llamada al procedimiento *recibe* la realiza la tarea de interrupción cuando se recibe un paquete completo.

4.- Rutina y tarea de atención de interrupción.

La transmisión y recepción de caracteres en las líneas serie utilizadas se realiza mediante interrupciones. Por lo tanto, es necesario definir un procedimiento de atención de interrupción que recoja los caracteres recibidos y envíe los que se quieren transmitir.

El Ada83 permite la utilización de interrupciones, pero solamente en los puntos de entrada de las tareas, por lo que no se puede realizar un procedimiento de atención de interrupción. Tampoco las extensiones del compilador utilizado permiten este tipo de procedimientos, pero sí que se implementa la llamada a tareas desde procedimientos de interrupción realizados en ensamblador.

Utilizar una tarea para atender una interrupción producida por cada caracter recibido o transmitido es una opción muy costosa en tiempo de procesador, debido al rendezvuos que se efectúa.

Así pues, la implementación que se lleva acabo se basa en el desarrollo en lenguaje emsamblador del procedimiento de atención de interrupción (que utiliza el nivel 5). Este procedimiento se encarga de las recepciones y transmisiones de caracteres por la línea serie.

Para la transmisión se envía cada caracter del *buffer de transmisión*, y se produce una nueva interrupción para el envío del siguiente caracter. En el momento en que se han mandado todos los caracteres del buffer, se llama a la tarea de atención de interrupción, *interrupcion_nivel_5* del paquete *comunicaciones*, para que desencole un nuevo paquete de comunicaciones a través del procedimiento *orden_transmision* del paquete Ada de las comunicaciones de transmisión.

En la recepción, cada caracter que llega se almacena en el buffer de recepción, hasta que se detecta el final de paquete de comunicaciones, en cuyo caso se llama a la tarea de atención de interrupción (la misma que para la transmisión), para que lo canalice por la red adecuada a través del procedimiento *recibe* del paquete Ada de las comunicaciones de recepción.

La tarea de atención de interrupción mencionada es la encargada de ordenar la transmisión de un nuevo paquete de comunicaciones, o de canalizar el paquete que se ha recibido.

Consideramos que en la transmisión de caracteres no se producen errores, aunque sería posible realizar una implementación más compleja de este nivel de la comunicación capaz de realizar una transmisión libre de errores.

Las especificaciones de los paquetes Ada, y los pseudocódigos de los procedimientos y tareas mencionados se encuentran en los apartados A.1 y A.4 del apéndice A.

La implementación que hemos realizado de las comunicaciones de tiempo real mediante líneas punto a punto se puede mejorar sensiblemente si se programa en Ada95. En primer lugar, se puede mejorar la implementación de las colas de prioridad y de su control, si se definen como objetos protegidos, al igual que ocurre en las colas de mensajes y en la de la red de recepción. Pero además, permite hacer en Ada el procedimiento protegido de atención de interrupción por caracter recibido o transmitido, con lo que se elimina también la tarea de interrupción. Ada95 soporta este tipo de procedimientos, como se puede ver en el anexo de programación de sistemas, del manual de referencia [ADA95].

5.4.4. Comunicaciones a través del bus VME

En este apartado, vamos a describir la implementación de las comunicaciones de tiempo real en el bus VME a través de memoria compartida. Al igual que para las líneas punto a punto, toda la implementación está hecha en Ada, salvo la parte correspondiente a la rutina de atención de interrupción, que ha sido realizada en ensamblador. También aquí se utiliza un mecanismo de planificación por prioridades y expulsor por paquetes de longitud fija, y es igualmente válida la estructura de comunicaciones representada en la Figura 5-4 (ver página 5-40).

Los puntos que vamos a reseñar en la definición del software de comunicaciones para el bus VME son los siguientes:

1.- Cola de control del bus.

Puesto que la comunicación entre los distintos nudos conectados al bus se hace a través de memoria compartida, tenemos que establecer un mecanismo de control del bus para preservar la coherencia de los datos.

El mecanismo que creamos para controlar el bus es la *cola de control del bus*, cuya estructura de datos se define en el paquete *hardware* (contiene las estructuras de datos de la implementación directamente relacionadas con el hardware para comunicaciones). Los parámetros de control de esta cola se exponen a continuación:

Estado de transmisión. Indica si se está transmitiendo, porque aún quedan paquetes de comunicación en alguna cola de transmisión, o si está parada la transmisión.

Disparo de transmisión. Indica si la interrupción ha sido disparada para transmitir un paquete de comunicaciones (simple transferencia de control del bus, en cuyo caso no se ejecuta la recepción de paquete).

Lista de procesadores. Lista cuyo número de elementos es igual al número de procesadores conectados al bus. Cada elemento contiene la siguiente información (relevante para el control del bus en este caso):

Prioridad. Mayor prioridad entre los paquetes de comunicación encolados para ser transmitidos en ese procesador.

Cola vacía. Indica si la cola de transmisión está vacía en ese nudo.

Además, para proteger la exclusión mutua en el uso de la *cola de control del bus*, se define un *semáforo* y una *función de reserva de semáforo*, que se realiza en ensamblador mediante la instrucción *TAS* (test and set) multiprocesadora.

El *buffer* de memoria compartida que se utiliza como medio para el intercambio de paquetes de comunicación, no necesita ser protegido por un semáforo, puesto que con la creación de la *cola de control del bus* se consigue que solamente el nudo en el que se encuentre encolado el paquete de mayor prioridad, tenga capacidad de transmisión.

2.- Transmisión.

La transmisión se implementa mediante el paquete *comunicaciones_transmision_bus*, en cuya interfase aparecen los procedimientos *transmite* y *ordena_transmision*, y la excepción *no_hay_espacio_en_la_cola*, y cuyas

finalidades son las mismas que los homónimos definidos para las líneas punto a punto.

El proceso de transmisión también se basa en una cola de prioridad, y la gestión de esta cola la realiza la tarea *sincroniza_transmite*, cuyos principios de funcionamiento también coinciden con los de la implementada en las líneas punto a punto, en cuanto al encolado y desencolado de paquetes de comunicación para la transmisión, y a la copia del paquete en el *buffer*.

Sin embargo, las operaciones de control que son ejecutadas por los mismos puntos de entrada que el encolado y desencolado de paquetes de comunicación son sensiblemente diferentes en el bus que en las líneas punto a punto.

La transmisión de un paquete de comunicaciones la inicia *sincroniza_transmite.transmite* o *sincroniza_transmite.orden_transmision*, dependiendo de que la transmisión esté parada o activa. En cualquier caso, una vez hecha la copia del paquete en el *buffer*, se transfiere el control del bus al nudo al que se destina el mensaje.

Además estos puntos de entrada tienen capacidad para transferir el control a otro nudo, sin transmisión de paquete de comunicaciones, o parar la transmisión, en el caso de que se les haya llamado como resultado de la recepción de un paquete.

3.- Recepción

La recepción se implementa mediante el procedimiento *recibe* localizado en el paquete *comunicaciones_recepcion_bus*. Este procedimiento recoge el paquete de comunicaciones del *buffer* de memoria compartida y lo incorpora al mensaje adecuado. Si se trata del último paquete, lo introduce en la cola de *red_recepcion* mediante el procedimiento *encola*. La llamada al procedimiento *recibe*, la realiza la tarea de interrupción cuando *disparo_de_-transmisión* es falso, es decir, cuando no se trata de una simple transferencia de control del bus.

4.- Rutina y tarea de atención de interrupción.

La transferencia del control de un nudo a otro, ya sea simple, o para la transmisión de un paquete de comunicaciones, se realiza provocando una interrupción en el nudo al que se le quiere transferir el control.

Del mismo modo que en las líneas punto a punto, es necesario recurrir al ensamblador para realizar el procedimiento de atención de interrupción (de nivel 7), que en este caso tan sólo tiene que realizar la llamada a la tarea de interrupción, *interrupcion_nivel_7* del paquete *comunicaciones*.

Esta tarea es la encargada de recibir el mensaje que se ha enviado en el caso de que la interrupción no haya sido solamente un *disparo de transmisión*. En cualquier caso, haya o no recepción de paquete de comunicaciones, se llama al procedimiento *orden_transmision* del paquete Ada de las comunicaciones de transmisión, para consultar la *cola de control del bus* y ver quién debe tener el control, o detener la transmisión (todas las colas de transmisión están vacías).

Las especificaciones de los paquetes Ada, y los pseudocódigos de los procedimientos y tareas mencionados se encuentran en los apartados A.1 y A.4 del apéndice A.

En la misma línea que para las comunicaciones punto a punto, la implementación que hemos realizado de las comunicaciones de tiempo real a través del bus VME se puede mejorar sensiblemente si se programa en Ada95. Podemos definir la cola de prioridad como un objeto protegido, y la tarea de interrupción, como un procedimiento de interrupción, lo que supone la eliminación del rendezvous, y por lo tanto, un ahorro de tiempo.

6. *Servidor esporádico para comunicaciones*

6.1. Introducción

El concepto de *servidor esporádico* surge de la necesidad de incorporar las respuestas a eventos aperiódicos al modelo de Liu y Layland dentro del desarrollo de la teoría RMA. Tradicionalmente los eventos aperiódicos se manejaban mediante tareas de muestreo periódico (*polling*), es decir, tareas periódicas que realizaban un chequeo para ver si se había producido alguno de los eventos que manejaban. Sin embargo, cuando un evento aperiódico llegaba inmediatamente después de que la tarea de *polling* hubiera hecho su chequeo, el evento debía esperar un período completo de esta tarea antes de ser procesado.

La necesidad de mantener la naturaleza periódica de las tareas que manejan los eventos aperiódicos, y a la vez, de procesarlos de una forma inmediata, llevó al desarrollo de los *servidores aperiódicos* [LEH87]. Además, estos servidores permiten que el sistema pueda ser analizado.

Por todos estos factores, aparece el *servidor esporádico* [SPR89B] como un mecanismo para la planificación de actividades aperiódicas en sistemas de tiempo crítico; éste proporciona una respuesta rápida a eventos aperiódicos y limita el efecto aperiódico.

El *algoritmo del servidor esporádico* controla la ejecución de las tareas que responden a eventos aperiódicos, permitiendo la ejecución de estas tareas aperiódicas como si se tratase de tareas periódicas con tiempos de respuesta cortos, y con un impacto predecible (e igual al de una tarea periódica equivalente) sobre las tareas que responden a eventos de menor prioridad. El algoritmo del servidor esporádico está caracterizado por tres parámetros: *prioridad*, *capacidad de ejecución* y *período de relleno*. En este algoritmo se reserva una cierta cantidad de *capacidad de ejecución* para procesar eventos aperiódicos a un nivel de *prioridad* determinado, que una vez consumida, se recupera cuando expire el *período de*

relleno. Cuando llega un evento aperiódico, éste activa la ejecución de una tarea aperiódica (el instante de tiempo de la llegada se guarda para futuras referencias). El tiempo de ejecución consumido por la tarea se va restando de la capacidad del servidor hasta que ésta se termina, y en ese momento, se suspende la ejecución de la tarea aperiódica. Cada porción de capacidad de ejecución consumida se rellena, es decir, se añadirá más tarde a la capacidad disponible. El instante de relleno es igual al instante en el que se activó la porción de capacidad consumida (normalmente el instante de llegada del evento) más el período de relleno. En algunas implementaciones la tarea aperiódica con capacidad de ejecución nula puede continuar su ejecución a una *prioridad reducida de background*.

Con la utilización de la política del *servidor esporádico*, los efectos del procesado de actividades aperiódicas usando un servidor esporádico no pueden ser peores que el efecto equivalente de una tarea periódica con un período igual al período de relleno, y un tiempo de ejecución igual a la capacidad inicial de ejecución del servidor esporádico. Por lo tanto, este mecanismo garantiza el control del impacto producido por las tareas aperiódicas sobre la planificabilidad del resto de las tareas del sistema.

Como vimos en el capítulo 2, en los sistemas distribuidos el efecto del *jitter* influye de manera negativa en la planificabilidad, ya que contribuye al aumento de los tiempos de respuesta de peor caso, y como consecuencia hace que disminuyan las posibilidades de utilización de los recursos. Sin embargo, aunque cualitativamente conocemos el impacto de este efecto negativo, no disponemos de una estimación cuantitativa de este impacto. Por consiguiente, es necesario realizar un estudio del impacto real del *jitter* en la planificabilidad de los sistemas distribuidos de tiempo real estricto. Veremos cómo este efecto negativo sobre la planificabilidad es realmente muy alto para estos sistemas, y las ventajas que reporta su posible eliminación.

De la estimación del impacto del *jitter* sobre la planificabilidad de los sistemas distribuidos, surge la necesidad de buscar técnicas eliminen este efecto indeseable. Nosotros propondremos la utilización del *servidor esporádico* para eliminar el *jitter*. Para ello, introduciremos el *servidor esporádico* tanto en la planificación de las tareas, como en la planificación de los mensajes de comunicación. El principal problema que se plantea es la

aplicación del *servidor esporádico* a la planificación en las redes de comunicación, ya que su aplicación a las tareas es sobradamente conocida.

En este capítulo vamos a estudiar en primer lugar el efecto del *jitter* sobre la planificabilidad de los sistemas distribuidos de tiempo real estricto; más tarde vamos a describir el modelo del *servidor esporádico para comunicaciones*, su implementación (tanto para redes de comunicación punto a punto, como para redes multipunto), y finalmente su aplicación a las redes de comunicación de tiempo real desarrolladas en el capítulo 5.

6.2. Estudio de los efectos del *jitter* sobre la planificabilidad

En los sistemas multiprocesadores y distribuidos la activación retrasada o *jitter* aparece como consecuencia de que las respuestas a los eventos del sistema son ejecutadas por recursos diferentes. El impacto que tiene el *jitter* sobre la planificabilidad en los sistemas multiprocesadores y distribuidos es realmente importante cuando éstos deben responder a requerimientos de tiempo real estricto. La Figura 6-1 muestra una gráfica en la que se representa los promedios de las utilizaciones máximas de los recursos (procesadores y redes) frente a la relación plazos/períodos (D/T) de las diferentes respuestas a los eventos que llegan al sistema para dos casos: con *jitter* y sin *jitter*. La gráfica se basa en los resultados obtenidos de la aplicación de las herramientas de asignación de prioridades y de generación automática de ejemplos al sistema de ejemplo con 50 tareas periódicas distribuidas en 8 procesadores, y 43 mensajes periódicos enviados a través de 3 redes de comunicación diferentes (ver el apartado 3.4 del capítulo 3). En esta gráfica podemos observar por ejemplo, que la máxima utilización promedio para $D/T=7$ es aproximadamente un 50% si el *jitter* está presente, y casi un 100% si se ha podido evitar. Los resultados obtenidos para otros ejemplos apuntan en esta misma dirección. A partir de esta misma gráfica podemos ver que la diferencia entre las dos líneas representadas (con y sin *jitter*) es mayor para plazos de principio-a-fin más largos. Los plazos largos son muy comunes en los sistemas distribuidos debido a que una respuesta se ejecuta en diferentes recursos (procesadores y redes), con plazos locales similares al período en cada recurso, de manera que el plazo global suele ser aproximadamente igual al período multiplicado por el número de recursos utilizados en la respuesta. De todo ello, podemos deducir que realmente la eliminación del *jitter* en este tipo de sistemas de tiempo real supone un aumento considerable de su planificabilidad.

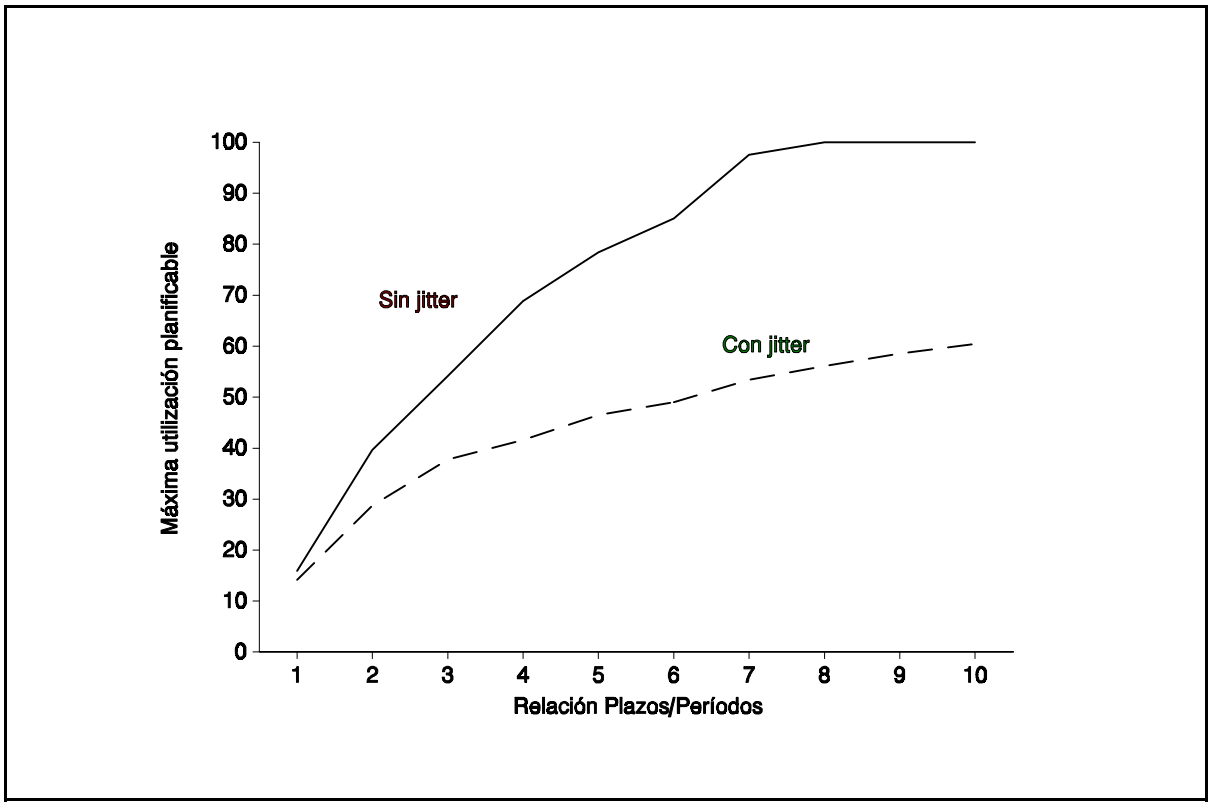

Figura 6-1. Efecto del jitter en la máxima utilización planificable.

Este efecto negativo en la planificabilidad se puede evitar calculando el tiempo de respuesta de peor caso de una tarea que dispara a un mensaje o a otra tarea, y creando una tarea agente de alta prioridad que libere al mensaje o a la tarea con *jitter* solamente después de que haya transcurrido ese tiempo de respuesta de peor caso de la tarea original. Esta técnica recibe el nombre de *protocolo de control de fase* [SUN95]. Sin embargo, desde el punto de vista de la ingeniería software, esta técnica no es mejor que la aproximación tradicional de ejecutivos cíclicos en la que el diseño del software está fuertemente marcado por los requerimientos temporales. Generalmente, intentaremos mantener siempre la independencia entre la planificación (y el análisis temporal) y las fases de las tareas.

El uso del *servidor esporádico* hace posible la eliminación del problema del *jitter* en el tiempo de respuesta, porque es capaz de impedir que cualquier acción (tarea o mensaje) se ejecute más de una vez durante el intervalo de tiempo correspondiente a su período. Esto se consigue asignando un servidor esporádico a la tarea que sufre de activación retrasada, con una capacidad inicial igual al tiempo de ejecución de peor caso de la tarea, y un período de

relleno igual al período de la tarea. Además, esta técnica preserva el principio de la ingeniería software de separación de intereses entre los requerimientos lógicos y los temporales.

Aunque el *servidor esporádico* ha sido diseñado como un algoritmo de planificación para procesadores, vamos a ver cómo el concepto se puede adaptar fácilmente a la planificación del tráfico de mensajes en redes de comunicación de tiempo real. Si tomamos como referencia el *servidor esporádico* empleado por las tareas, podemos pensar en desarrollar un mecanismo de control para la transmisión de mensajes por las redes de comunicación de tiempo real utilizadas en los sistemas distribuidos, con la finalidad de eliminar el *jitter* producido en dichas redes, y contribuir así al aumento de la planificabilidad del sistema.

A este mecanismo de control de la transmisión en redes de comunicación de tiempo real basadas en prioridades, le vamos a llamar *servidor esporádico para comunicaciones*. Al igual que el desarrollado para los procesadores, se basa en la asignación de una *capacidad de transmisión* determinada para cada *prioridad*, que una vez consumida se recupera cuando transcurre el *período de relleno*. La mayoría de las redes y de los protocolos de red disponibles no utilizan prioridades, y muchas tienen tiempos de transmisión de peor caso que no se pueden predecir. Afortunadamente, es posible obtener un comportamiento temporal predecible con algunas de las redes utilizadas en comunicaciones de tiempo real estricto, como TDMA [TIN94F], *token rings* [AGR91] [STR88], el bus CAN [TIN94E], etc. Se pueden implementar protocolos con prioridades para los mensajes en el sentido de conseguir un incremento de la planificación de los mensajes. Por ejemplo, el bus CAN utiliza el identificador de mensaje como controlador del acceso al bus y también como prioridad del mensaje. Es en este tipo de redes donde planteamos la introducción del *servidor esporádico para comunicaciones* que vamos a describir a continuación.

6.3. Modelo del servidor esporádico para comunicaciones

En el contexto de los sistemas multiprocesadores y distribuidos de tiempo real estricto en los que se utilizan redes cuyas comunicaciones se basan en prioridades (tal como la implementación de las colas de mensajes descritas en el capítulo 5), definimos el *servidor esporádico para comunicaciones (SEC)* como un mecanismo de control temporal de la

transmisión de información para el aumento de la planificabilidad de este tipo de sistemas. En este mecanismo, a cada prioridad le asignamos una serie de parámetros que sirven para determinar y establecer en todo instante de tiempo la capacidad de transmisión a esa prioridad.

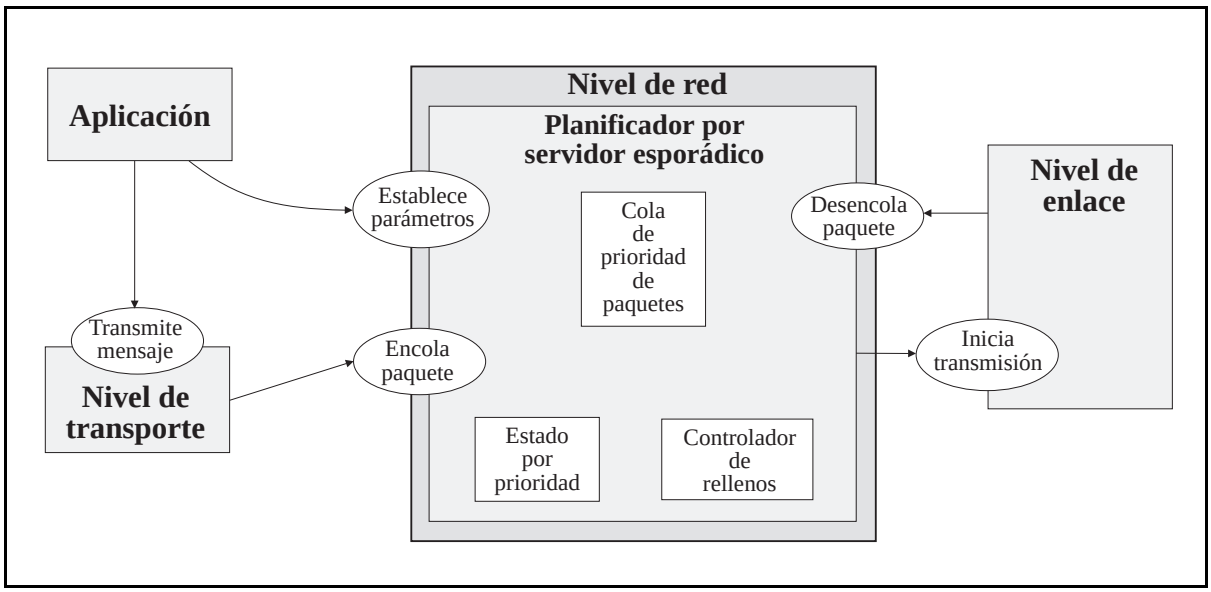

Figura 6-2. Planificador de mensajes mediante servidor esporádico.

Tomando como referencia el contexto de planificación de mensajes por prioridades mostrado en la Figura 5-4 del capítulo anterior, es fácil adaptar el concepto de *servidor esporádico* (Figura 6-2). Para una secuencia de mensajes dada, los parámetros del servidor esporádico son: la capacidad inicial (número de paquetes que se pueden transmitir a prioridad normal), y el período de relleno. El planificador de la red puede transmitir al menos el número de paquetes que se hayan reservado con prioridad normal. Por cada uno de esos paquetes transmitidos, la capacidad de transmitir otro paquete se recupera un período de relleno después de que el paquete fuera encolado.

En el planificador por *servidor esporádico* para tareas, se asocia un *servidor esporádico* a cada tarea. Sin embargo, los mensajes no constituyen un concepto como las tareas, las cuales pueden formar parte de múltiples respuestas a secuencias de eventos particulares; los mensajes llegan al subsistema de comunicaciones sin ninguna relación aparente con otros mensajes previos. Como consecuencia, tenemos que definir un *servidor esporádico* para que actúe sobre cualquier mensaje de un nivel de prioridad determinado. Esto quiere decir que es necesario un número de prioridades suficiente para permitir que cada

secuencia de mensajes tenga diferente nivel de prioridad, y poder distinguir así unas secuencias de otras.

La Figura 6-2 muestra la estructura general de un planificador de mensajes conceptual basado en el *SEC*. El *SEC* tiene la misma interfase que el planificador de mensajes por prioridades convencional (Figura 5-4 del capítulo anterior) más una operación que establece los parámetros iniciales de los servidores correspondientes a cada prioridad. Estos parámetros son los siguientes:

Capacidad inicial. Es el número de paquetes de comunicación que se pueden transmitir a prioridad normal sin restricciones (más tarde definiremos el estado de transmisión).

Período de relleno. Tiempo que debe transcurrir desde el instante en el que fue encolado el paquete de comunicaciones hasta el instante en el que el servidor puede recuperar la capacidad consumida por la transmisión de ese paquete.

El planificador con *SEC* define internamente un objeto que es una cola de prioridad en la que se pueden almacenar los paquetes de comunicación, y que se puede manipular a través de las operaciones *introduce* y *extrae*. Esta cola cumple las funciones de cola de transmisión y las operaciones deben implementar el control de la misma mediante los mecanismos temporales adecuados. Antes de describir cómo actúan las operaciones, vamos a definir los parámetros que corresponden al entorno de la implementación.

Para la definición de la cola de prioridad del *SEC* se necesitan dos tipos de datos del entorno de la implementación:

Tipo de prioridad. El tipo de prioridad que se utiliza en el *SEC* para definir la cola debe ser el mismo que se defina en las comunicaciones que se estén implementando.

Tipo de paquete. Es el tipo que identifica a los paquetes de comunicación en la implementación que se esté realizando.

Cada nivel de prioridad, y por lo tanto, cada servidor dentro del *SEC*, lleva asociados una serie de parámetros de control y de estado que se definen de la forma siguiente:

Capacidad actual. En cualquier instante de tiempo es el número de paquetes de comunicación que se pueden transmitir a la prioridad asociada. Inicialmente la capacidad actual es igual a la capacidad inicial.

Instante de activación. Este parámetro se establece como el instante actual en dos situaciones que son: bien cuando se encola un paquete de comunicaciones en un servidor determinado a la prioridad asociada, la capacidad actual es mayor que cero y no hay otros paquetes encolados; o bien, cuando se ejecuta un relleno (esta operación se describirá más adelante).

Capacidad usada. Es el número de paquetes consumidos (a la prioridad asociada) desde el último instante de activación establecido.

Estado. Define el modo de extracción de paquetes de comunicación y se establece en función de la capacidad actual y del número de paquetes encolados con la prioridad correspondiente. El estado del *SEC* para una prioridad determinada puede ser uno de los tres siguientes:

Inactivo	Cuando no hay paquetes en la cola con esa prioridad.
Normal	Cuando hay paquetes en la cola con esa prioridad y la capacidad actual es mayor que cero.
Background	Si hay paquetes en la cola con esa prioridad, pero la capacidad actual es cero.

El mecanismo que controla el tiempo en el *SEC* se define como un objeto que vamos a llamar *cola de rellenos* y que consiste en una lista cuyos elementos, llamados *rellenos*, están ordenados por tiempos de manera creciente (primero el tiempo más pequeño). La información que contienen los rellenos es la siguiente:

Tiempo de relleno. Es el instante de tiempo en el que expira un relleno, es decir, el instante en el que se puede recuperar una determinada capacidad usada aumentando la capacidad actual del *SEC* en la cantidad establecida por capacidad de relleno a la prioridad de relleno.

Capacidad de relleno. Es la capacidad que debe ser rellenada (en número de paquetes). Esta capacidad fue consumida a la prioridad de relleno en el momento en el que se añade el relleno a la cola de rellenos.

Prioridad de relleno. Es el nivel de prioridad del *SEC* que se debe rellenar.

Las operaciones que se pueden efectuar sobre el objeto cola de rellenos son las que se detallan a continuación:

Planificar un relleno. Esta operación consiste en incorporar un relleno a la cola de rellenos. Un relleno se planifica estableciendo para una prioridad determinada (prioridad de relleno) la capacidad usada (capacidad de relleno), y el instante de tiempo (tiempo de relleno) en el que se puede recuperar esa capacidad usada, que es el instante actual más el período de relleno. En la llamada a esta operación se deben suministrar los parámetros: capacidad de relleno, tiempo de relleno y prioridad de relleno.

Ejecutar rellenos. La ejecución de rellenos consiste en extraer de la cola de rellenos todos los rellenos cuyos tiempos de relleno hayan expirado, es decir, cuyos tiempos de relleno sean anteriores o iguales al tiempo actual. Por cada relleno ejecutado, se debe añadir la capacidad de relleno a la capacidad actual del servidor para la prioridad de relleno, y se establece el instante de activación con la hora actual. Si la capacidad actual fuera cero y el estado asociado a esa prioridad estuviera en background, habría que cambiar el estado a normal.

Para terminar con la definición del modelo del *SEC* describimos sus operaciones básicas para inicialización y manipulación de la cola de prioridad:

Establece parámetros. Se establecen la capacidad inicial y el período de relleno para la prioridad requerida.

Introduce. Incorpora un paquete de comunicaciones a la cola del *SEC* en la prioridad correspondiente siguiendo un orden FIFO. Si el estado del servidor asociado a esa prioridad fuera inactivo, se cambiaria a normal o background (para capacidad mayor que cero o igual a cero respectivamente). Si el estado cambia a normal, hay que establecer de nuevo el instante de activación.

Extrae. Ejecuta los rellenos pendientes y después extrae un paquete de comunicaciones de la cola del *SEC*. Esta operación normalmente se invoca desde el controlador apropiado del nivel de enlace para seleccionar el paquete que debe ser transmitido. Se busca en la cola el paquete de comunicaciones de mayor prioridad que tenga asociado un servidor en estado normal. Si no hay paquetes en estado normal, se extrae el paquete de mayor prioridad en estado background. La extracción de un paquete en estado normal implica el consumo de una unidad de la capacidad actual, y si esta capacidad llega a ser cero, o si el servidor pasa a estado inactivo (porque no hay más paquetes encolados a esa prioridad), se realiza la operación de planificación de un relleno. La capacidad de relleno es igual a la capacidad usada, y el tiempo de relleno se calcula como el instante de activación más el período de relleno. La extracción de un paquete en estado background no consume capacidad (hay que recordar que en este caso ya es cero), y solamente se puede realizar si no hay paquetes en estado normal encolados en el *SEC*.

En líneas generales podemos decir que en cada operación se debe mantener la coherencia en cuanto a los valores actuales de las capacidades y de los estados del *SEC* para las diferentes prioridades, lo que se consigue ejecutando los rellenos oportunos. También es interesante poner de manifiesto que se garantiza la extracción, y por tanto, la transmisión de paquetes a cualquier prioridad que aún disponga de capacidad, y además que la extracción en estado background no consume capacidad (ya que en este estado la capacidad es nula).

6.4. Implementación del servidor esporádico para comunicaciones

En este apartado vamos a desarrollar la implementación del servidor esporádico para comunicaciones de acuerdo con el modelo definido en el apartado anterior. Además, continuando con la implementación de redes de comunicación de tiempo real estricto cuya descripción se hizo en el capítulo 5, vamos a ampliar ahora dicha implementación incorporando el *SEC* a la transmisión de paquetes. Naturalmente el lenguaje de desarrollo tanto para el *SEC*, como para su incorporación a las redes de comunicación va a seguir siendo Ada.

La definición del *SEC* como una cola de paquetes de comunicación y no sólo como el control de esta cola, le da una total independencia del entorno a la hora de realizar su implementación. Esto nos permite servirnos de las utilidades que posee el Ada para la creación de unidades genéricas, y así, definir la implementación del *SEC* como un paquete Ada genérico.

Con el fin de satisfacer las necesidades de control de la transmisión de paquetes de comunicación en distintos tipos de redes, vamos a realizar dos implementaciones del *SEC* que de una forma general abarcan los casos de redes dedicadas punto a punto, y de redes multipunto. Ambas implementaciones cumplen las funciones del *SEC* en los términos en que éste ha sido definido, pero atendiendo a las características particulares de cada tipo de red difieren en algunos aspectos. Así pues, tendremos dos paquetes Ada genéricos correspondientes a las dos implementaciones del *SEC*.

La Figura 6-3 muestra las dependencias entre los paquetes Ada que forman parte tanto de la implementación del *SEC*, como de su aplicación a las diferentes redes de comunicación, y también muestra la localización de estos paquetes Ada en el entorno correspondiente al nivel de red en el que van a ser implementados. A continuación se realiza una breve descripción de los paquetes Ada a los que hemos hecho referencia:

SS_comunicaciones	Paquete genérico que implementa el *SEC* de aplicación a redes punto a punto. Internamente se

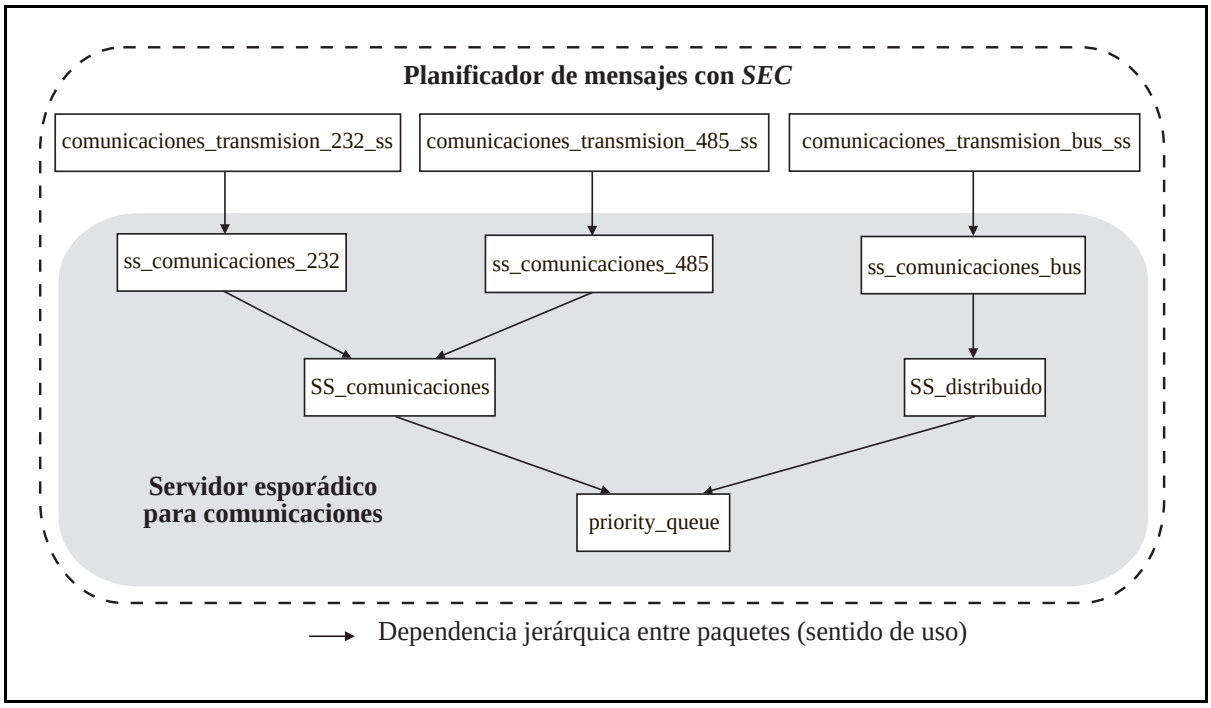

Figura 6-3. Estructura de paquetes Ada del planificador con *SEC*.

define el paquete *cola_rellenos* (instanciación de *priority_queue*).

SS_distribuido — Paquete genérico que implementa el *SEC* de aplicación a redes multipunto. Internamente se define el paquete *cola_rellenos* (instanciación de *priority_queue*).

priority_queue — Paquete genérico que implementa una cola cuyos elementos se ordenan de acuerdo con la prioridad y la función de orden definidas.

ss_comunicaciones_232 — Paquete que implementa el *SEC* para una línea serie RS-232. Instanciación de *SS_comunicaciones*.

ss_comunicaciones_485	Paquete que implementa el *SEC* para una línea serie RS-485. Instanciación de *SS_comunicaciones*.
ss_comunicaciones_bus	Paquete que implementa el *SEC* para el bus VME. Instanciación de *SS_distribuido*.
comunicaciones_transmisión_232_ss	Localización de la tarea de transmisión de paquetes a través de una línea serie RS-232 utilizando el *SEC*.
comunicaciones_transmisión_485_ss	Igual que el anterior para una línea serie RS-485.
comunicaciones_transmisión_bus_ss	Igual que los anteriores para el bus VME.

Las especificaciones de los paquetes Ada relatados se muestran en el apartado A.1 del apéndice A.

Seguidamente vamos a describir las dos implementaciones realizadas para el *SEC* y su aplicación a las redes utilizadas, fijándonos en todos aquellos aspectos relevantes de la interfase, de la estructura de datos y de las operaciones de control de éstos, y haciendo especial hincapié en las analogías y diferencias entre ambas implementaciones.

6.4.1. Implementación en redes punto a punto

La implementación del *SEC* para redes punto a punto se lleva a cabo, como ya hemos dicho, en el paquete Ada genérico *SS_comunicaciones*, en cuya especificación (o interfase) podemos distinguir los siguientes elementos:

1.- Parámetros genéricos.

max_elementos_cola_servidor : **in** positive;

Número máximo de elementos de la cola de prioridad que implementa el *SEC*.

max_elementos_cola_rellenos : **in** positive;

Número máximo de elementos reservados para la cola de rellenos.

type *tipo_prioridad* **is range** <>;

Tipo de prioridad que se utiliza para definir la cola de prioridad del *SEC*.

type *tipo_paquete* **is private**;

Tipo que define el elemento de la cola de prioridad del *SEC*. Normalmente será un puntero o un índice al paquete de comunicaciones definido en la implementación.

2.- Procedimientos y funciones de la interfase.

Si bien las descripciones de estos procedimientos y funciones las daremos más adelante, en este punto vamos a clasificarlos de la forma siguiente:

Interfase hacia los programas de usuario:

procedure *establece_parametros* (…);

Interfase hacia los programas de comunicaciones:

procedure *introduce* (…);

procedure *extrae* (…);

function *cola_vacia* **return** boolean;

function *espacio_disponible* **return** integer;

3.- Excepciones.

no_hay_espacio_en_la_cola_servidor : **exception**;

Sucede cuando la cola de prioridad del *SEC* está llena y se intenta introducir un paquete, aunque se proporcionan los instrumentos suficientes como para que nunca se produzca esta excepción.

no_hay_espacio_en_la_cola_rellenos : **exception**;

Ocurre si se intenta planificar un relleno y la cola de rellenos está completa.

Siguiendo con la definición de la implementación, vamos a considerar ahora la estructura de datos del *SEC* atendiendo a los dos objetos que se definen internamente, que son:

La cola de prioridad de paquetes. En la definición de esta cola se considera tanto la propia estructura de la cola de paquetes, como la estructura correspondiente a los servidores asociados a cada uno de los niveles de prioridad con los parámetros de control y de estado correspondientes.

La cola de rellenos. La estructura de datos correspondiente a la cola de rellenos configura una cola de prioridad ordenada por tiempos en la que los elementos contienen la información correspondiente a los rellenos definidos.

En la estructura de datos que conforma la cola de prioridad del *SEC* podemos considerar los siguientes tipos y variables:

1.- Tipos.

type *tipo_estado_cola* **is** (*inactivo*, *normal*, *background*);

Define el estado de cada prioridad para la extracción de paquetes.

type *tipo_elemento_cola_servidor* **is**
record
paquete : *tipo_paquete*;
siguiente : positive;
end record;

Define el elemento de la cola de prioridad del *SEC* (de tipo *tipo_paquete*) para ser utilizado en una lista enlazada por índices.

type *tipo_lista* **is**
record
cabeza : positive;
Apunta al primer elemento de la lista.
cola : positive;
Apunta al último elemento de la lista.
numero_de_elementos : integer := 0;
Número actual de elementos en la lista.

end record;

Tipo que se define para crear una lista enlazada de elementos de la cola del servidor por cada prioridad.

type *tipo_cabeza_cola_servidor* **is**
record
estado : *tipo_estado_cola* := *inactivo*;
Estado del *SEC* para una prioridad determinada.
lista : *tipo_lista*;
Lista de elementos encolados a una prioridad determinada.
C_actual : integer := 0;
Capacidad actual.
C_usada : integer;
Capacidad usada.
T_activacion : calendar.time;
Instante de activación.
periodo_relleno : duration;
Período de relleno.
end record;

Definición del tipo que contiene los parámetros del *SEC* para cada prioridad.

type *tipo_cola_servidor* **is array** (*tipo_prioridad*) **of**
tipo_cabeza_cola_servidor;

Define la cola de prioridad del *SEC*.

2.- Variables.

elementos_cola_servidor : **array** (1..*max_elementos_cola_servidor*) **of**
tipo_elemento_cola_servidor;

Espacio reservado para contener los elementos de la cola de prioridad del *SEC*.

cola_servidor : *tipo_cola_servidor*;

Creación de la cola de prioridad del *SEC* que se había definido.

numero_de_elementos_cola_servidor : integer := 0;

Número de elementos en la cola del *SEC* en el instante actual (en el instante inicial ninguno).

libres_cola_servidor : *tipo_lista* :=
(*cabeza* => 1,
cola => *max_elementos_cola_servidor*,
numero_de_elementos => *max_elementos_cola_servidor*);

Lista enlazada de elementos de la cola del *SEC* que están libres (inicialmente todos).

La estructura de datos que constituye la cola de rellenos del *SEC* está compuesta por un tipo de dato que define el relleno (con los parámetros que le caracterizan), y por un paquete que define una lista ordenada en función de los tiempos de relleno. Estos son los siguientes:

1.- Tipo.

type *tipo_elemento_cola_rellenos* **is**
record
    capacidad : integer;
    prioridad : *tipo_prioridad*;
end record;

Definición del elemento de la cola de rellenos. Solamente se necesitan la capacidad y la prioridad porque con la definición que vamos a hacer de la cola de rellenos, el tiempo de relleno está implícito en la posición que ocupa el elemento en la cola.

2.- Paquete cola_rellenos.

Paquete Ada que implementa la cola de rellenos, en la que los elementos se encolan ordenados según el tiempo de relleno en orden creciente, es decir, primero el menor.

Para construir esta cola utilizamos el paquete genérico *priority_queue*, procedente de otro desarrollo en Ada que implementa una cola en la que los elementos se almacenan de acuerdo con una prioridad establecida, y que presenta los siguientes parámetros genéricos:

qsize	Número máximo de elementos en la cola.
element	Tipo de elemento de la cola.
priority_t	Tipo de prioridad usado para ordenar los elementos de la cola.
">"	Función usada para ordenar las prioridades.
compare	Función de comparación de los elementos de la cola. La utiliza *dequeue_middle* para localizar el elemento que se quiere extraer.

Seguidamente se muestran las operaciones que se pueden realizar sobre la cola:

empty y *full*	Funciones de consulta de cola vacía o llena.
enqueue	Inserta un elemento en la cola por su prioridad.
dequeue	Extrae el elemento de mayor prioridad de la cola (no se impone un orden particular para elementos con la misma prioridad).
dequeue_middle	Extrae un elemento concreto de la cola.
read_first	Devuelve el elemento de mayor prioridad de la cola sin extraerlo.

La instanciación del paquete genérico *priority_queue* para crear la cola de rellenos se efectúa en los términos siguientes:

```
package cola_rellenos is new priority_queue
        (qsize        => max_elementos_cola_rellenos,
         element      => tipo_elemento_cola_rellenos,
         priority_t   => calendar.time,
         ">"          => calendar."<",
         compare      => "=" );
```

Una vez que hemos definido la interfase y la estructura de datos del *SEC*, vamos a describir mediante pseudocódigo los procedimientos y funciones que implementan las operaciones de la interfase y de control interno de los objetos definidos. Comenzamos por las operaciones de control definidas internamente, y que son utilizadas por las operaciones de la interfase del *SEC* hacia los programas de comunicaciones. Podemos distinguir los siguientes procedimientos:

1.- Procedure *cambia_prioridad_a_normal (…)*

```
procedure cambia_prioridad_a_normal
          (prioridad : in tipo_prioridad) is

begin
   Establece el cambio de estado a normal para prioridad:
      if estado /= normal then
         C_usada := 0;
         estado := normal;
      end if;
end cambia_prioridad_a_normal;
```

El cambio del estado a normal para una prioridad determinada implica la actualización del estado y la puesta a cero de la capacidad usada.

2.- Procedure *cambia_prioridad_a_background (…)*

```
procedure cambia_prioridad_a_background
          (prioridad : in tipo_prioridad) is

begin
   Establece el cambio de estado a background para prioridad:
      estado := background;
end cambia_prioridad_a_background;
```

El cambio del estado a background para una prioridad determinada implica únicamente la actualización del estado.

3.- Procedure *ejecuta_rellenos*

```
procedure ejecuta_rellenos is

begin
   loop
      exit when cola de rellenos vacía;
      Lee el primer relleno de la cola de rellenos:
         establece su tiempo de relleno;
      Establece hora actual;
      exit when tiempo de relleno > hora actual;
      Desencola el relleno de la cola de rellenos;
      Para la prioridad del relleno:
         Establece T_activacion con la hora actual;
         if estado = background then
            cambia_prioridad_a_normal(prioridad del relleno);
         end if;
      Aumenta la capacidad actual a la prioridad del relleno en una
         cantidad igual a la capacidad del relleno;
   end loop;
end ejecuta_rellenos;
```

Este procedimiento realiza la operación de ejecución de rellenos, para lo cual realiza un lazo en el que va ejecutando los rellenos hasta que se vacía la cola o no se puede ejecutar el primer relleno de la cola de rellenos. Cada vez que se comprueba si se puede ejecutar un relleno, se establece la hora actual con el fin de realizar la comparación de tiempos de la forma más exacta posible. Es necesario cambiar el estado a normal para la prioridad del relleno

en el caso de que estuviera en background. Con la ejecución de cada relleno se establece el instante de activación para la prioridad correspondiente.

4.- Procedure *planifica_un_relleno (…)*

```
procedure planifica_un_relleno
          (T_relleno : in calendar.time;
           capacidad : in integer;
           prioridad : in tipo_prioridad) is

begin
   Chequeo de excepciones:
      raise no_hay_espacio_en_la_cola_rellenos;
   Establece parámetros del relleno:
      capacidad;
      prioridad;
   Encola el relleno en la cola de rellenos según T_relleno;
end planifica_un_relleno;
```

Este es el procedimiento que ejecuta la operación de planificación de un relleno, en la que después de comprobar que la cola de rellenos no está completa, establece los parámetros del relleno y lo encola.

En cuanto a las operaciones que constituyen la interfase del *SEC*, vamos a comentar aquellos aspectos que tienen un interés especial para su utilización. Distinguimos los siguientes procedimientos y funciones:

1.- Procedure *establece_parametros (…)*

```
procedure establece_parametros
          (prioridad         : in tipo_prioridad;
           capacidad_inicial : in integer;
           periodo_relleno   : in duration) is

begin
   Establece parámetros del SEC para prioridad:
      C_actual := capacidad_inicial;
      periodo_relleno;
end establece_parametros;
```

Este es el procedimiento que constituye la interfase del *SEC* hacia los programas de usuario, y su misión es establecer la capacidad actual (a partir de la capacidad inicial, que es la máxima) y el período de relleno para una prioridad determinada.

2.- Procedure *introduce (…)*

```
procedure introduce (paquete   : in tipo_paquete;
                     prioridad : in tipo_prioridad) is

begin
   if cola_vacia then
      ejecuta_rellenos;
   end if;
   Chequeo de excepciones:
      raise no_hay_espacio_en_la_cola_servidor;
   Mete el paquete en la cola del SEC en su prioridad;
   Establece el nuevo estado para prioridad:
      if estado = inactivo then
         if C_actual = 0 then
            cambia_prioridad_a_background(prioridad);
         else
            Establece T_activacion con la hora actual;
            cambia_prioridad_a_normal(prioridad);
         end if;
      end if;
end introduce;
```

Procedimiento de interfase hacia los programas de comunicaciones que se encarga de introducir un paquete en la cola de prioridad del *SEC* a la prioridad correspondiente. La ejecución de rellenos es una operación que solamente es necesario realizar en la extracción de paquetes. Sin embargo, cuando la cola del *SEC* está vacía también se realiza en la introducción de paquetes para asignar el estado manteniendo la consistencia de los datos en todo momento. En el caso de que el estado de la cola para la prioridad del paquete fuera inactivo, se establecería el nuevo estado de acuerdo con la capacidad actual (si la capacidad fuese mayor que cero se establecería también el instante de activación con la hora actual).

Aunque existe una función de consulta del espacio disponible en la cola del *SEC*, se comprueba si la cola está llena para elevar la correspondiente excepción.

3.- Procedure *extrae (…)*

```
procedure extrae (paquete : out tipo_paquete) is

prioridad : tipo_prioridad;

begin
   ejecuta_rellenos;
   Extrae el paquete de la cola del SEC y establece prioridad;
   Actualiza parámetros para prioridad:
```

```
      if estado = normal then
         C_actual := C_actual-1;
         C_usada := C_usada+1;
         if (C_actual = 0) or
            (lista.numero_de_elementos = 0) then
            planifica_un_relleno
               (T_activacion+periodo_relleno,C_usada,prioridad);
         end if;
         if (C_actual = 0) and
            (lista.numero_de_elementos /= 0) then
            cambia_prioridad_a_background(prioridad);
         end if;
      end if;
      if lista.numero_de_elementos = 0 then
         estado := inactivo;
      end if;
end extrae;
```

Es el procedimiento de la interfase hacia los programas de comunicaciones que se encarga de sacar de la cola del *SEC* el paquete con mayor prioridad en estado normal, o en estado background si no hubiera ninguno en estado normal. Antes de localizar el paquete que se debe extraer hay que ejecutar los rellenos para trabajar con los valores actuales de estados y capacidades disponibles. Para finalizar es necesario establecer los nuevos valores del estado y de las capacidades para la prioridad del paquete que se ha desencolado, y también comprobar si hay que planificar un relleno (si se agota la capacidad actual o se acaban los paquetes de la cola del *SEC*).

4.- Function *cola_vacia*

```
function cola_vacia return boolean is

begin
   if numero_de_elementos_cola_servidor = 0 then
      return true;
   else
      return false;
   end if;
end cola_vacia;
```

Esta función toma el valor *true* si la cola está vacía. Los programas de comunicaciones podrán comprobar mediante esta función si queda algún paquete por extraer.

5.- Function *espacio_disponible*

```
function espacio_disponible return integer is

begin
   return (número de elementos disponibles en la cola del SEC);
end espacio_disponible;
```

A través de esta función los programas de comunicaciones pueden conocer el número de paquetes que se pueden almacenar aún en la cola del *SEC* (a cualquier prioridad). De esta forma, si se quieren encolar varios paquetes, se puede consultar el espacio disponible para comprobar si es suficiente.

Hasta este momento hemos descrito la implementación realizada del *SEC*; ahora vamos a poner de manifiesto las características particulares de la utilización del paquete Ada genérico en el que éste se implementa, aunque algunas ya se han comentado en el contexto de las operaciones relatadas. Estas características son las siguientes:

No se garantiza la exclusión mutua en la utilización que de la cola del *SEC* hacen las operaciones *introduce* y *extrae* (deben de ser los programas de comunicaciones que utilizan el *SEC* los que tienen que establecer esta garantía). Esto permite limitar la sobrecarga que puede suponer la utilización del *SEC* sin perjudicar su adaptación a los programas de comunicaciones, porque éstos ya realizan normalmente estas operaciones de manipulación de paquetes de manera mutuamente excluyente (el *SEC* viene a sustituir a la cola de transmisión). Además, la utilización de estas dos operaciones de manera mutuamente excluyente garantiza también la exclusión mutua de las operaciones que se ejecutan sobre la cola de rellenos.

Antes de intentar extraer un paquete se debe consultar si la cola del *SEC* está vacía.

Cuando se quieran introducir varios paquetes que componen un mensaje se debe consultar previamente el espacio disponible y actuar en consecuencia. Esto es debido a que los paquetes hay que introducirlos en la cola del *SEC* de uno en uno.

Los programas de usuario deben configurar los parámetros del *SEC* (capacidad inicial y período de relleno) al menos para aquellas prioridades que se vayan a utilizar. Esta configuración se realiza mediante el procedimiento *establece_parametros (…)* correspondiente a la interfase del *SEC*. En caso de que se utilice alguna prioridad no configurada, ésta operará en estado de background.

6.4.2. Implementación en redes multipunto

En líneas generales el *SEC* para redes multipunto debe responder a unas necesidades de control de la transmisión de paquetes de comunicación en una cola de prioridad distribuida a través de diferentes nudos. Esta cola distribuida se concibe como un conjunto de colas de prioridad (una por cada nudo de la red multipunto) para almacenar los paquetes a transmitir de cada nudo, más una estructura global de control de estás colas (accesible por todos los nudos). Esta estructura de control debe tener la información correspondiente a cuál de las colas es la que tiene el paquete de comunicaciones que se debe transmitir en primer lugar. El controlador de la transmisión en cada nudo transmite un paquete o transfiere el control al nudo adecuado en función de esta estructura global de control. La adaptación del *SEC* a este tipo de redes sin sacrificar la concepción genérica del mismo viene caracterizada por los siguientes puntos:

- El *SEC* se implementa como una cola de prioridad en cada nudo de la misma forma que para redes punto a punto.

- La estructura global de control de la cola distribuida debe cambiar en el sentido de incorporar los parámetros de control de la transmisión correspondientes al *SEC* (como el estado del servidor para cada prioridad). Esta estructura de control se mantiene en el ámbito de los programas de comunicaciones y accesible por todos los nudos de la red.

- Los programas de comunicaciones deben suministrar al *SEC* un procedimiento para la manipulación de la estructura de control.

- Se añade un nuevo procedimiento al *SEC* para que los programas de comunicaciones puedan consultar cual es la prioridad y el estado al que puede transmitir un nudo en un momento dado.

De la misma forma que para redes punto a punto, vamos a describir seguidamente la implementación del *SEC* para redes multipunto, que como ya dijimos, se lleva a cabo en el paquete Ada genérico *SS_distribuido*. Existen algunas diferencias entre las interfases de las dos implementaciones del *SEC*, que se pueden apreciar en la relación de los parámetros de la interfase para redes multipunto que damos a continuación:

1.- Parámetros genéricos.

max_elementos_cola_servidor : **in** positive;
max_elementos_cola_rellenos : **in** positive;
type *tipo_prioridad* **is range** <>;
type *tipo_paquete* **is private**;

Iguales que para redes punto a punto.

with procedure cambio_a_normal (prioridad : **in** tipo_prioridad);

Procedimiento que se suministra al *SEC* para que marque el cambio a normal en alguna prioridad cuando la operación de ejecución de rellenos no es activada por *introduce* o *extrae*.

2.- Procedimientos y funciones de la interfase.

La relación de procedimientos y funciones se ve incrementada por el procedimiento *estado*, cuya descripción realizaremos más adelante, y queda de la siguiente manera:

Interfase hacia los programas de usuario:

procedure *establece_parametros* (…);

Interfase hacia los programas de comunicaciones:

procedure *introduce* (…);
procedure *extrae* (…);
function *cola_vacia* **return** boolean;
function *espacio_disponible* **return** integer;
procedure *estado* (…);

3.- Excepciones.

no_hay_espacio_en_la_cola_servidor : **exception**;
no_hay_espacio_en_la_cola_rellenos : **exception**;

Iguales que para redes punto a punto.

La estructura de datos de la implementación del *SEC* para redes multipunto coincide totalmente con la de las redes punto a punto, ya que únicamente contiene la cola de prioridad y la cola de rellenos, y estas colas se pueden definir de manera exacta en ambas implementaciones.

Continuando con la descripción de la implementación, pasamos ahora a comentar los procedimientos y funciones que realizan las operaciones de control interno y de la interfase hacia los programas de comunicaciones, tal como hicimos para las líneas punto a punto. También vamos a comentar la tarea que se utiliza para el control de las operaciones sobre la cola de rellenos. Esta tarea se hace necesaria para mantener la consistencia de los datos del *SEC* en todo momento y en todos los nudos. Sin esta tarea se podrían transmitir paquetes a una prioridad y en un nudo que no corresponden, por no detectar a tiempo un cambio en el estado de algún servidor producido por expiración de algún relleno un nudo en el que momentáneamente no se están realizando operaciones sobre la cola de prioridad.

Aunque vamos a dar una relación completa de las operaciones, únicamente explicaremos aquellas que difieren de las implementadas para redes punto a punto. En primer lugar vamos a describir los procedimientos correspondientes a las operaciones de control interno, junto con la tarea de control de rellenos de la forma que se expone a continuación:

1.- Procedure *cambia_prioridad_a_normal (…)*

Igual que para redes punto a punto.

2.- Procedure *cambia_prioridad_a_background (…)*

Igual que para redes punto a punto.

3.- Task *control_rellenos*

```
task control_rellenos is
   pragma priority(24);
```

```
   entry planifica_un_relleno (T_rell    : in calendar.time;
                               capacidad : in integer;
                               prioridad : in tipo_prioridad);
   entry ejecuta_rellenos;
end control_rellenos;

task body control_rellenos is

durmiendo : duration := 86400.0;

begin
   loop
      select
         accept planifica_un_relleno
                (T_rell    : in calendar.time;
                 capacidad : in integer;
                 prioridad : in tipo_prioridad) do
            Chequeo de excepciones:
               raise no_hay_espacio_en_la_cola_rellenos;
            Establece parámetros del relleno:
               capacidad;
               prioridad;
            Encola el relleno en la cola de rellenos según
               T_rell;
            Lee el primer relleno y su tiempo de relleno;
            Establece hora actual;
            durmiendo := tiempo de relleno -hora actual;
         end planifica_un_relleno;
      or
         accept ejecuta_rellenos do
            loop
               durmiendo := 86400.0;
               exit when cola de rellenos vacía;
               Lee el primer relleno de la cola de rellenos:
                  establezco su tiempo de relleno;
               Establece hora actual;
               durmiendo := tiempo de relleno-hora actual;
               exit when tiempo de relleno > hora actual;
               Desencola el relleno de la cola de rellenos;
               Para la prioridad del relleno:
                  Establece T_activacion con la hora actual;
                  if estado = background then
                     cambia_prioridad_a_normal
                             (prioridad del relleno);
                  end if;
               Aumenta la capacidad actual a la prioridad del
                  relleno en una cantidad igual a la capacidad
                  del relleno;
            end loop;
         end ejecuta_rellenos;
      or
         delay durmiendo;
            loop
               durmiendo := 86400.0;
               exit when cola de rellenos vacía;
               Lee el primer relleno de la cola de rellenos:
                  establezco su tiempo de relleno;
               Establece hora actual;
               durmiendo := tiempo de relleno-hora actual;
               exit when tiempo de relleno > hora actual;
               Desencola el relleno de la cola de rellenos;
               Para la prioridad del relleno:
                  Establece T_activacion con la hora actual;
                  if estado = background then
                     cambia_prioridad_a_normal
```

```
                                (prioridad del relleno);
                            cambio_a_normal(prioridad del relleno);
                        end if;
                    Aumenta la capacidad actual a la prioridad del
                        relleno en una cantidad igual a la capacidad
                        del relleno;
                end loop;
            end select;
        end loop;
    end control_rellenos;
```

Esta tarea realiza las operaciones de planificación y ejecución de rellenos de una forma similar a los procedimientos *planifica_un_relleno* y *ejecuta_rellenos* en las redes punto a punto.

Debido al carácter distribuido de las colas de prioridad en una red multipunto, es necesario que la ejecución de rellenos tenga su propia línea de flujo de control de programa, es decir, que la ejecución de rellenos la realice una tarea. Esta tarea además tiene que admitir la planificación de rellenos para garantizar la correcta manipulación de la cola de rellenos. La ejecución de rellenos se puede efectuar de dos maneras: una forzada a través de un punto de entrada, y otra por expiración de un tiempo determinado mediante el *delay*.

A continuación vamos a describir las líneas de operación de esta tarea de acuerdo con los siguientes puntos:

Planifica_un_relleno. Realiza la operación de planificación de un relleno. Actualiza la variable *durmiendo* según el primer relleno de la cola y la hora actual.

Ejecuta_rellenos. Operación de ejecución de rellenos forzada por las operaciones *introduce* y *extrae*. También actualiza la variable *durmiendo* según el primer relleno de la cola y la hora actual.

Delay. Ejecuta el primer relleno en el instante que le corresponda, siempre que no haya sido ejecutado de manera forzada. Cuando se produce un cambio a prioridad normal hay que notificarlo mediante el procedimiento pasado como parámetro genérico *cambio_a_normal*. En este caso, el cambio de las capacidades y del estado de la cola para una prioridad determinada puede implicar un cambio en el control global de las colas. Igualmente actualiza la variable *durmiendo* según el primer relleno de la cola y la hora actual.

La duplicidad en la operación de ejecución de rellenos se justifica por la necesidad de que ésta se efectúe de manera instantánea cuando se realizan las operaciones *introduce* y *extrae*. Con la ejecución de cada relleno se establece el instante de activación para la prioridad correspondiente.

En cuanto a las operaciones que constituyen la interfase del *SEC*, distinguimos los siguientes procedimientos y funciones:

1.- Procedure *establece_parametros (…)*

Igual que para redes punto a punto.

2.- Procedure *introduce (…)*

Igual que para redes punto a punto cambiando la llamada a *ejecuta_rellenos* por *control_rellenos.ejecuta_rellenos.*

3.- Procedure *extrae (…)*

Igual que para redes punto a punto cambiando las llamadas a *ejecuta_rellenos* y *planifica_un_relleno* por *control_rellenos.ejecuta_rellenos* y *control_rellenos.planifica_un_relleno* respectivamente.

4.- Function *cola_vacia*

Igual que para redes punto a punto.

5.- Function *espacio_disponible*

Igual que para redes punto a punto.

6.- Procedure *estado (…)*

```
procedure estado (prioridad   : out tipo_prioridad;
                  modo_normal : out boolean) is

begin
   control_rellenos.ejecuta_rellenos;
   Establece la prioridad del paquete que se puede extraer y el modo
      en el que está la prioridad:
            normal     -->  modo_normal := true;
            background -->  modo_normal := false;
end estado;
```

Este procedimiento devuelve el estado y la prioridad del paquete que la operación *extrae* sacaría de la cola del *SEC*.

Para terminar, vamos a poner de manifiesto las características particulares de la utilización del paquete Ada genérico en el que se implementa el *SEC* para redes multipunto, algunas de las cuales coincidirán con las de la implementación anterior para redes punto a punto y no las comentaremos (ver página 6-23). Estas características son las siguientes:

No se garantiza la exclusión mutua en la utilización que de la cola del *SEC* hacen las operaciones *introduce* y *extrae*.

Antes de intentar extraer un paquete se debe consultar si la cola del *SEC* está vacía.

Después de extraer un paquete se debe realizar una consulta del estado para establecer la prioridad y el estado del nudo actual.

Cuando se quieran introducir varios paquetes que componen un mensaje se debe consultar previamente el espacio disponible.

Los programas de usuario deben configurar los parámetros del *SEC* al menos para aquellas prioridades que se vayan a utilizar.

6.5. Aplicación a las redes implementadas del servidor esporádico para comunicaciones

En este apartado vamos a incorporar las implementaciones del *SEC* que hemos realizado tanto para redes punto a punto, como para redes multipunto, al desarrollo de las redes de comunicación basadas en colas de mensajes cuya implementación se describe en el capítulo 5.

En primer lugar, tenemos que crear los paquetes Ada de utilidad correspondientes al *SEC* mediante la instanciación de los paquetes Ada genéricos en los que éste se implementa. Obtenemos los paquetes *ss_comunicaciones_232* y *ss_comunicaciones_485* de aplicación a las líneas punto a punto, y *ss_comunicaciones_bus* de aplicación al bus VME. Los términos en los que se efectúan las instanciaciones se muestran en el apartado A.1 del apéndice A. Como se puede observar, el tipo de prioridad que se utiliza es el que se define para las colas de mensajes. Como tipo de paquete, utilizamos el identificador del mensaje de comunicaciones del espacio de mensajes definido en el paquete *memoria*, es decir, el tipo *positive*. Para el bus VME se establece además el parámetro genérico *cambio_a_normal* mediante el procedimiento *marca_cambio_a_normal* del paquete *hardware*, cuyo pseudocódigo se encuentra en el apartado A.4 del apéndice A, y comentaremos más adelante.

El *SEC* viene a sustituir a la cola de prioridad que se utilizaba en las comunicaciones de transmisión por una cola de prioridad controlada en función de unos requisitos temporales. Por lo tanto, definimos un nuevo paquete Ada por cada red utilizada para implementar las comunicaciones de transmisión con *SEC* de una forma análoga a como lo hacíamos sin él. Teniendo en cuenta los dos tipos de redes que se implementan dividimos los nuevos paquetes Ada de la forma siguiente:

1.- *Comunicaciones_transmision_232_ss/485_ss*.

La interfase que presentan estos paquetes es la misma y con la misma finalidad, que la de los paquetes *comunicaciones_transmision_232/485* que implementan las comunicaciones de transmisión cuando no se utiliza el *SEC*.

También en el cuerpo de estos paquetes Ada encontramos la tarea *sincroniza_transmite* que ejecuta las operaciones de encolado y desencolado de paquetes de comunicación de igual forma que antes, pero haciendo uso ahora del *SEC*. El pseudocódigo de esta tarea se encuentra en el apartado A.4 del apéndice A.

La cola de transmisión desaparece al utilizar el *SEC*, pero el resto de la estructura de comunicaciones permanece inalterada; tan sólo se realizan algunos cambios en el modo de operación que pasamos a detallar.

En nuestra implementación, el modo de encolar un mensaje de comunicaciones partido en paquetes en la cola del *SEC* es ejecutar la operación

introduce con el identificador del mensaje; tantas veces como paquetes tenga el mensaje. Cada vez que se extrae un paquete recuperamos en realidad el identificador del mensaje de comunicaciones del espacio de mensajes, que ya contiene la información actual del paquete que se debe transmitir. Con el tipo de paquete elegido para la cola del *SEC*, las operaciones de manipulación de paquetes de comunicaciones no suponen una sobrecarga importante para las comunicaciones de transmisión. La manipulación de la información de los mensajes y de los paquetes de comunicación se realiza fuera del ámbito del *SEC*, y éste se constituye únicamente como un controlador de la transmisión.

Es importante poner de manifiesto que absolutamente todos los paquetes deben pasar por la cola del *SEC*. Aunque la cola estuviera vacía y el mensaje tuviera solamente un paquete, habría que introducirlo y después extraerlo para ser transmitido. En otro caso, se perdería la consistencia en el control de la transmisión por parte del *SEC* (se tendrían valores erróneos en las capacidades y en los estados de los servidores).

2.- *Comunicaciones_transmision_bus_ss*.

Este paquete Ada sustituye a *comunicaciones_transmision_bus* en el caso de que se utilice el *SEC*. Al igual que para líneas punto a punto no se altera la estructura de comunicaciones con excepción de la cola de transmisión de paquetes distribuida.

También para el bus VME se mantienen intactos tanto el mecanismo de transmisión de paquetes, como la interfase que se presenta hacia los demás paquetes Ada de las comunicaciones. El pseudocódigo de la tarea de control de la transmisión, *sincroniza_transmite* con las modificaciones relativas a la utilización del *SEC*, se muestra en el apartado A.4 del apéndice A.

La modificación más importante, que se produce al incorporar el *SEC* a la transmisión a través del bus VME, se encuentra en los factores que intervienen en la decisión del paquete de comunicaciones que se tiene que transmitir, a causa de que la cola de transmisión está distribuida. De acuerdo con esto, vamos a especificar finalmente estas modificaciones destacando los siguientes puntos:

- La *cola de control del bus* se modifica en el sentido de incorporar la información relativa al estado en el que se encuentra la prioridad a la que va a transmitir un procesador concreto, que aparece como consecuencia del estado en el que se encuentra la prioridad del paquete que puede ser extraído de la cola del *SEC*. El estado de la prioridad de transmisión refleja el estado normal o background de las prioridades del *SEC*, y va a servir para reproducir en el mecanismo de decisión de paquete a transmitir los mismos esquemas utilizados en el *SEC*.
- El procedimiento *marca_cambio_a_normal* del paquete *hardware* es el que utiliza el *SEC* para marcar un cambio al estado normal en alguna prioridad, producido por la ejecución temporizada de rellenos. Aunque se accede a un recurso que está protegido por un semáforo, no es necesario bloquear éste último para establecer la prioridad y el estado (se podría producir un bloqueo mutuo en la tarea de control de rellenos del *SEC*). Sin embargo, sí que es importante actualizar primero la prioridad y después el estado (cuando es necesario modificar ambos) para que no se produzca una inconsistencia en los datos.
- Después de extraer un paquete, hay que consultar el estado mediante el procedimiento *ss_comunicaciones_bus.estado*, para establecer los valores adecuados en la *cola de control del bus* en cuanto a prioridad y modo de transmisión en el procesador que realiza la operación.

7. Modelado de las redes de comunicación

7.1. Introducción

En los capítulos anteriores hemos descrito técnicas de análisis y planificación de sistemas multiprocesadores y distribuidos de tiempo real estricto. También hemos implementado algunas redes que se pueden utilizar en este tipo de sistemas haciendo uso de técnicas de paso de mensajes. Sin embargo, para poder analizar y planificar los sistemas que utilicen estas redes es necesario un modelado adecuado de las mismas, tal como el que se abordará en este capítulo.

El modelado de las redes de comunicación consiste en la definición de los modelos a los que se adaptan las operaciones que se implementan en las redes, en cuanto a la descripción de las acciones o tareas que forman parte de dichas operaciones, y en la evaluación de los tiempos de peor caso para cada acción.

En este capítulo vamos a realizar el modelado de las redes de comunicación basadas en las colas de mensajes que hemos implementado. En primer lugar, definiremos el modelado general de las colas de mensajes mediante las operaciones que se pueden realizar sobre ellas. Después pasaremos a desarrollar el modelado de las comunicaciones para las redes punto a punto y multipunto que hemos implementado. Tomando como referencia estos modelos, se podrá hacer ya una evaluación del software desarrollado para las redes de comunicación, considerando los tiempos de peor caso de las acciones detalladas en el modelo, y definiendo además todos los casos particulares a que puedan dar lugar estas acciones. En segundo lugar, vamos a realizar el modelado y la evaluación del servidor esporádico para comunicaciones en los mismos términos que las colas de mensajes. Por último, vamos a exponer algunos ejemplos en los que, por un lado, se aplicarán los modelos definidos a la evaluación de mismos, y por el otro, se pondrán de manifiesto tanto las características peculiares de uso de

las redes de comunicación implementadas, como las ventajas que supone la utilización del servidor esporádico en las redes de comunicación.

En el apéndice B se muestra una lista con la terminología definida en este capítulo, sus respectivos significados y una lista de correspondencias entre las acciones y las tareas que las realizan.

7.2. Modelado de las colas de mensajes

Las colas de mensajes constituyen un mecanismo de comunicación con una buena base para el desarrollo de redes de comunicación de tiempo real estricto, tal como vimos en el capítulo 5 de esta Tesis.

En la implementación realizada de las colas de mensajes distinguimos dos partes: por un lado, la que constituye propiamente las colas de mensajes, es decir, las operaciones que las caracterizan y que pueden ser utilizadas por los programas de usuario; y por otro lado, las comunicaciones que son características de cada red y a través de las cuales se realizan las operaciones remotas sobre las colas de mensajes. Así pues, en la definición del modelo vamos a tener en cuenta una primera diferenciación en cuanto a la parte dependiente y la no dependiente de la red de comunicaciones.

También se puede hacer una segunda diferenciación si tenemos en cuenta el tipo de operación que se realiza sobre la cola de mensajes, es decir, si la operación es local o remota. De acuerdo con ésta, definimos el modelado general de las operaciones sobre las colas de mensajes en los términos que se muestran en la Figura 7-1, en la que se pueden distinguir los siguientes tipos de operaciones:

Tipo I — *Operación local*. Operaciones locales que no implican la utilización de la red. Estas operaciones son las siguientes: *abre local* (en sus dos modalidades: abre y crea), *cierra local*, *destruye local*, *envía local* (si no hay operación *recibe remoto* en espera), *recibe local*, *recibe si hay local* y *coge atributos local*.

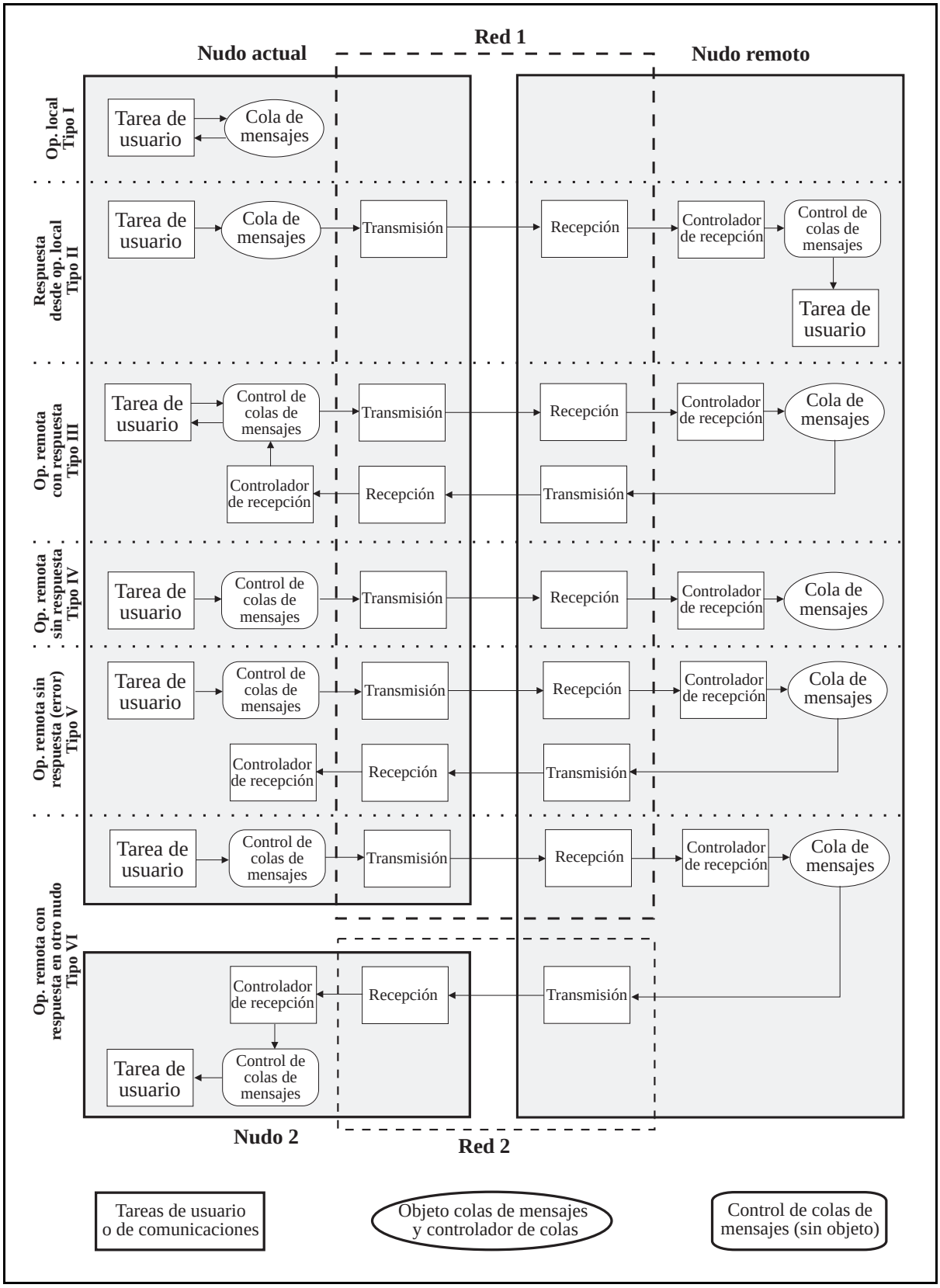

Figura 7-1. Modelado general de las operaciones sobre las colas de mensajes.

Tipo II *Respuesta desde operación local*. Operación *envía* local que genera la respuesta a una operación *recibe remoto* que estuviera esperando.

Tipo III *Operación remota con respuesta*. Operaciones remotas en las que la tarea que las solicita se suspende en el nudo actual en espera de la respuesta. A este grupo pertenecen las operaciones remotas: *abre remoto*, *recibe remoto* (si hay mensajes en la cola), *recibe si hay remoto* y *coge atributos remoto*.

Tipo IV *Operación remota sin respuesta*. Operaciones remotas bien que no requieren la espera de respuesta y se llevan a cabo correctamente, o bien que se suspenden en el nudo remoto en espera de respuesta. Estas son las operaciones *cierra remoto* y *envía remoto* (sin tareas en espera pertenecientes a un nudo distinto del remoto) cuando su ejecución se efectúa sin errores, y también la operación *recibe remoto* con suspensión en el nudo remoto.

Tipo V *Operación remota sin respuesta, con error*. Operaciones remotas que no requieren la espera de respuesta, pero se produce un error y se debe gestionar en el nudo que realiza la operación. Son las operaciones *cierra remoto* y *envía remoto* para el caso en el que se producen errores.

Tipo VI *Operación remota con respuesta en otro nudo*. Operación *envía remoto* que genera la respuesta a una operación *recibe remoto* solicitada desde un nudo distinto del remoto. La respuesta se canaliza hacia este nudo (desde el que partió la operación *recibe remoto*).

En la Figura 7-1 se muestran en líneas generales las secuencias de acciones que se realizan en cada tipo de operación de los seis definidos anteriormente y se ponen de

manifiesto también los recursos desde el punto de vista tanto del software (colas de mensajes), como del hardware (redes y nudos), que se ven implicados en cada operación.

Pasamos ahora a describir el modelado concreto en acciones para cada tipo de operación, teniendo en cuenta las características específicas de la implementación desarrollada de las redes de comunicación, y siguiendo la pauta marcada por la primera diferenciación comentada de las mismas (dependiente y no dependiente de la red).

7.2.1. Modelado de las operaciones sobre las colas de mensajes

En este apartado vamos a realizar el modelado de las operaciones sobre las colas de mensajes, teniendo en cuenta las acciones a que dan lugar, que se corresponden con las tareas procedimientos y funciones de los paquetes Ada de la implementación.

A continuación, vamos a llevar a cabo la descomposición de las operaciones en acciones, para cada tipo de operación de los seis que hemos definido, de la forma siguiente:

Tipo I.

Dentro de este tipo de operaciones podemos distinguir los casos que se muestran a continuación:

- Operaciones locales *abre*, *cierra*, *destruye*, *envía*, *recibe si hay* y *coge atributos*. Las acciones que se realizan en estas operaciones son:

Recurso	Acciones
Nudo actual	*Colas (llamada por tarea de usuario)* *Control* *Colas (retorno a tarea de usuario)*

donde,

- *Colas*. Es el código correspondiente a los procedimientos del paquete Ada *colas_de_mensajes* ejecutados por la tarea de usuario al realizar la operación.

- *Control*. Se corresponde con el código que posee la tarea *sincroniza* del paquete Ada *control_de_colas* para cada operación sobre las colas de mensajes.

- Operación *recibe*. Las acciones para esta operación son:

Recurso	Acciones
Nudo actual	*Colas (llamada por tarea de usuario)* *< Control -- Suspensión -- >* *Colas (retorno a tarea de usuario)*

donde, *colas* y *control* son las acciones definidas anteriormente. Esta operación se suspende y ejecuta *control* tantas veces como sea necesario hasta que consiga un mensaje.

Tipo II.

En este tipo de operación consideramos las siguientes acciones:

Recurso	Acciones
Nudo actual (Envía)	*Colas (llamada por tarea de usuario)* *Control - Red_tr* *Com_tr* *Red_tr - Control* *Colas (retorno a tarea de usuario)*
Red	*Transmisión*
	Recepción
Nudo remoto (Recibe)	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Colas* *Libera_men* *Colas (retorno a la tarea de usuario)*

donde:

- *Colas* y *Control*. Son las acciones definidas anteriormente.
- *Red_tr*. Es el código correspondiente al procedimiento *transmite* del paquete *red_transmision*. Este código puede ser ejecutado por *colas*, es

decir, por la tarea de usuario, o por *control* (en la generación de la respuesta a la operación).

- *Com_tr*. Es la acción de encolado de paquetes en la cola de transmisión. La lleva a cabo la tarea *sincroniza_transmite* de los paquetes Ada *comunicaciones_transmision_xxx* o *comunicaciones_transmision_xxx_ss*.
- *Transmisión*. Se corresponde con las acciones de transmisión de paquetes de comunicación a través de las redes, que se abordarán con mayor profundidad más adelante.
- *Recepción*. Son las acciones correspondientes a la recepción de paquetes de comunicación a través de las redes, y al igual que las anteriores, también se verán más adelante.
- *Ordena_rec*. Es la acción de ejecución de las operaciones remotas o de sus respuestas en el nudo remoto. Se corresponde con la tarea *ordena_recibe* del paquete Ada *red_recepcion*.
- *Desencola_rec*. Es la acción de sacar un mensaje de la cola de recepción, y la realiza la tarea *control_recibe* del paquete *red_recepcion*.
- *Libera_men*. Es la acción de liberación de un mensaje, controlada por la tarea *maneja_recursos* del paquete Ada *memoria*.

Tipo III.

Las operaciones que corresponden a este tipo se componen de las siguientes acciones:

Recurso	Acciones
Nudo actual	*Colas (llamada por tarea de usuario)* *(Abre_rem)* *Reserva_men* *Colas - Red_tr* *Com_tr* *Red_tr - Colas*
Red	*Transmisión*
	Recepción

Nudo remoto	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Control (ejecuta_remoto) - Red_tr* *Com_tr* *Red_tr - Control (ejecuta_remoto)* *Ordena_rec*
Red	*Transmisión*
	Recepción
Nudo actual	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Colas* *Libera_men* *Colas (retorno a tarea de usuario)*

donde todas las acciones que intervienen han sido definidas en los tipos anteriores excepto las siguientes:

- *Abre_rem*. Es la acción de apertura de un descriptor remoto y la realiza la tarea *recursos_remotos* del paquete Ada *control_de_colas*.
- *Reserva_men*. Es la acción de reserva de un mensaje, controlada por la tarea *maneja_recursos* del paquete Ada *memoria*.
- *Control (ejecuta_remoto)*. Control es la acción definida anteriormente, pero en ejecución de una operación remota en el nudo remoto.

Tipo IV.

Para este tipo de operación vamos a distinguir los casos que se detallan a continuación:

- Operaciones *cierra* y *envía*. Estas operaciones se componen de las siguientes acciones:

Recurso	**Acciones**
Nudo actual	*Colas (llamada por tarea de usuario)* *Reserva_men* *Colas - Red_tr* *Com_tr* *Colas - Red_tr* *Cierra_rem* *Colas (retorno a tarea de usuario)*
Red	*Transmisión*
	Recepción
Nudo remoto	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Control (ejecuta_remoto)* *Libera_men* *Control (ejecuta_remoto)* *Ordena_rec*

Todas las acciones que intervienen han sido definidas en los tipos anteriores salvo *Cierra_rem* que es la acción de cierre de un descriptor remoto, y la realiza la tarea *recursos_remotos* del paquete Ada *control_de_colas*.

- Operación *recibe* (si no hay mensajes y debe esperar en la cola de espera remota). Se queda esperando hasta que se produzca una operación *envía* sobre esa cola y ejecute su respuesta.

Recurso	**Acciones**
Nudo actual	*Colas (llamada por tarea de usuario)* *Reserva_men* *Colas - Red_tr* *Com_tr* *Red_tr - Colas (retorno a tarea de usuario)*
Red	*Transmisión*
	Recepción

Nudo remoto	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Control (ejecuta_remoto)* *Ordena_rec*

Las acciones que intervienen son las definidas en los casos anteriores.

Tipo V.

Las operaciones que corresponden a este tipo se componen de las acciones que se detallan a continuación:

Recurso	Acciones
Nudo actual	*Colas (llamada por tarea de usuario)* *Reserva_men* *Colas - Red_tr* *Com_tr* *Colas - Red_tr* *Cierra_rem* *Colas (retorno a tarea de usuario)*
Red	*Transmisión*
	Recepción
Nudo remoto	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Control (ejecuta_remoto) - Red_tr* *Com_tr* *Red_tr - Control (ejecuta_remoto)* *Ordena_rec*
Red	*Transmisión*
	Recepción
Nudo actual	*Ordena_rec* *Desencola_rec* *Ordena_rec*

Todas las acciones que intervienen han sido definidas en los tipos anteriores.

Tipo VI.

Este tipo de operación está formado por las siguientes acciones:

Recurso	Acciones
Nudo actual (Envía)	*Colas (llamada por tarea de usuario)* *Reserva_men* *Colas - Red_tr* *Com_tr* *Red_tr - Colas (retorno a tarea de usuario)*
Red 1	*Transmisión*
	Recepción
Nudo remoto	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Control (ejecuta_remoto) - Red_tr* *Com_tr* *Red_tr - Control (ejecuta_remoto)* *Libera_men (del recibe remoto en espera)* *Control (ejecuta_remoto)* *Ordena_rec*
Red 2	*Transmisión*
	Recepción
Nudo 2 (Recibe)	*Ordena_rec* *Desencola_rec* *Ordena_rec* *Colas* *Libera_men* *Colas (retorno a la tarea de usuario)*

Las acciones que intervienen son las definidas en los tipos anteriores. En este caso la operación *envía* en su ejecución en el nudo remoto encuentra una operación *recibe* remota en espera, y elabora su respuesta. Por esta razón existe una Red 2 y un Nudo 2 que son los relacionados con la operación *recibe*. La Red 2 puede ser perfectamente la propia Red 1, y el Nudo 2 puede ser el nudo actual o cualquier nudo remoto distinto del que genera la respuesta a la operación *recibe*.

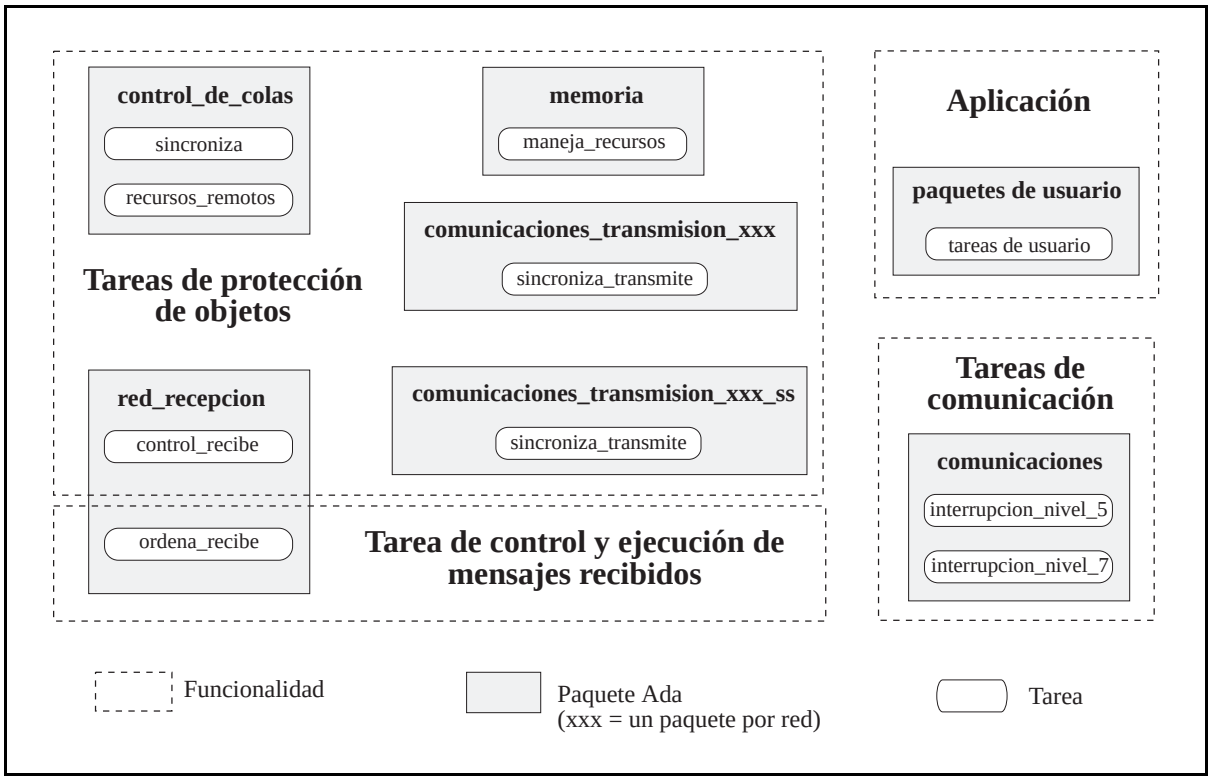

Figura 7-2. Tareas de la implementación en Ada de las redes de comunicación en el entorno de una aplicación.

Para terminar con el modelado de las colas de mensajes, vamos a dar una relación de las tareas que intervienen en la ejecución de las distintas acciones que hemos detallado para nuestra implementación, y también de las que intervienen en las acciones de comunicación que veremos en el próximo subapartado. Atendiendo a la funcionalidad de las tareas podemos hacer la siguiente clasificación (la Figura 7-2 muestra una visión global esta clasificación en el entorno de una aplicación):

1.- Tareas de comunicación.

Las tareas de comunicación son las que se encargan de la transmisión y recepción de paquetes en las redes de comunicación implementadas. Estas tareas, que se encuentran en el paquete *comunicaciones*, son activadas por procedimientos de interrupción y se detallan a continuación:

interrupcion_nivel_5 (Prioridad = Máxima+Nivel_Hardware)

interrupcion_nivel_7 (Prioridad = Máxima+Nivel_Hardware)

2.- Tarea de control y ejecución de mensajes recibidos.

Esta tarea se encarga de gestionar los mensajes que se reciben por las diferentes redes y que se encuentran en la cola de recepción. La tarea es *ordena_recibe* (Prioridad = Máxima-1) del paquete Ada *red_recepcion*.

3.- Tareas de protección de objetos.

Aquí se agrupan las tareas cuya única finalidad es la de garantizar la exclusión mutua en el acceso a determinados objetos o recursos, y que pueden ser sustituidas por objetos protegidos de Ada95. Estas tareas son las siguientes:

- En el paquete Ada *memoria*:

  maneja_recursos (Prioridad = Máxima). Protección de los mensajes de comunicaciones.

- En el paquete Ada *control_de_colas*:

  sincroniza (Prioridad = Máxima). Protección de las operaciones sobre las colas de mensajes.

  recursos_remotos (Prioridad = Máxima). Protección de los descriptores remotos de las colas de mensajes.

- En el paquete Ada *red_recepcion*:

  control_recibe (Prioridad = Máxima). Protección de la cola que contiene los mensajes recibidos a través de las redes de comunicación.

- En los paquetes Ada de comunicaciones de transmisión disponemos de dos tareas por cada red implementada para la protección de la cola de transmisión: una que utiliza servidor esporádico para comunicaciones, y otras que no lo utiliza. Estas tareas son las siguientes:

  En *comunicaciones_transmision_xxx*:

  sincroniza_transmite (Prioridad = Máxima).

  En *comunicaciones_transmision_xxx_ss*:

  sincroniza_transmite (Prioridad = Máxima).

7.2.2. Modelado de las comunicaciones

Del mismo modo que para las colas de mensajes vamos a modelar ahora las comunicaciones para las redes implementadas teniendo en cuenta las acciones que intervienen en las operaciones de transmisión y recepción que se llevan a cabo. Antes de comenzar con el modelado de las comunicaciones, definimos algunos aspectos de la nomenclatura y la simbología que vamos a utilizar en la descripción de las secuencias de acciones o de tramos de acción que forman parte del modelado. Estas secuencias se van a representar mediante sucesiones de cajas, en las que cada caja la interpretamos de la forma que se expone a continuación:

Acción	Acción o tramo de acción.
< *Acción* >	Acción o tramo de acción localizados en un bucle.
-- *Suspensión* --	Tiempo que permanece una cadena de acciones suspendida.
Acción	Acción o tramo de acción ejecutables opcionalmente.
Acción	Acción o tramo de acción modelados con anterioridad.
Acción 1 *Acción 2*	Ejecución de una de las dos acciones.
Acción	Continuidad en las acciones (la cadena de acciones ocupa más de una línea).

Así pues, vamos a modelar las comunicaciones mediante la extracción de las acciones que se llevan a cabo en las operaciones de comunicación para las redes implementadas. Consideramos las siguientes operaciones en la comunicación:

1.- Transmisión en líneas serie RS-232 y RS-485.

Por cada caracter transmitido únicamente sucede una acción:

Interrupción nivel 5

donde, *interrupción nivel 5* se corresponde con la ejecución del procedimiento de interrupción para cada caracter transmitido.

Por cada paquete de comunicaciones transmitido, salvo el primero (en el caso de que la transmisión estuviese parada), se produce la secuencia de acciones siguiente:

Interrupción nivel 5	*Tarea interrupción 5*	*Ordena_tr*

Tarea interrupción 5	*Interrupción nivel 5*

donde:

- *Interrupción nivel 5*. Es la acción del procedimiento de interrupción cuando busca un nuevo paquete de comunicaciones para transmitir.
- *Tarea interrupción 5*. Es la acción que realiza la tarea de interrupción *interrupcion_nivel_5* del paquete Ada *comunicaciones* cuando es llamada para la transmisión de un nuevo paquete de comunicaciones.
- *Ordena_tr*. Es la acción de extracción de un paquete de comunicaciones de la cola de transmisión efectuada por la tarea *sincroniza_transmite* de los paquetes Ada *comunicaciones_transmision_xxx* y *comunicaciones_transmision_xxx_ss*.

2.- Transmisión en el bus VME.

En el bus VME no hay acción por caracter transmitido, pero por cada paquete de comunicaciones transmitido (salvo el primero si la transmisión estuviese parada), o para el cambio de control de la transmisión en un nudo, ocurre la siguiente secuencia de acciones:

Interrupción nivel 7	*Tarea interrupción 7*	*Ordena_ tr*

Tarea interrupción 7	*Interrupción nivel 7*

donde:

- *Interrupción nivel 7*. Es la acción del procedimiento de interrupción para la transmisión de un nuevo paquete de comunicaciones o el cambio en el control de la transmisión.
- *Tarea interrupción 7*. Es la acción que realiza la tarea de interrupción *interrupcion_nivel_7* del paquete Ada *comunicaciones* cuando es llamada para la transmisión de un nuevo paquete de comunicaciones o la transferencia del control de transmisión.
- *Ordena_tr*. Son las acciones tanto de extracción de un paquete de comunicaciones de la cola de transmisión, como de orden de cambio del control de transmisión, efectuadas por la tarea *sincroniza_transmite* de los paquetes Ada *comunicaciones_transmision_xxx* y *comunicaciones_transmision_xxx_ss*.

3.- Recepción en líneas serie RS-232 y RS-485.

Al igual que para la transmisión, por cada caracter recibido únicamente se produce la acción:

Interrupción nivel 5

donde, *interrupción nivel 5* se corresponde con la ejecución del procedimiento de interrupción para cada caracter recibido.

Por cada paquete de comunicaciones recibido sucede la secuencia de acciones siguiente:

Interrupción nivel 5	*Tarea interrupción 5 - Procedimiento recibe*

Encola_rec	*Tarea interrupción 5*	*Interrupción nivel 5*

donde:

- *Interrupción nivel 5*. Es la acción del procedimiento de interrupción cuando se recibe un paquete de comunicaciones completo.
- *Tarea interrupción 5 - Procedimiento recibe*. Es la acción que realiza la tarea de interrupción *interrupcion_nivel_5* del paquete Ada *comunicaciones* cuando es llamada a causa de la recepción de un paquete de

comunicaciones completo. Esta tarea ejecuta el procedimiento *recibe* de los paquetes Ada *comunicaciones_recepcion_xxx*.

- *Encola_rec*. Es la acción de encolado de un mensaje de comunicaciones (cuando está completo) en la cola de recepción. Esta acción la realiza la tarea *control_recibe* del paquete Ada *red_recepcion*.

4.- Recepción en el bus VME.

De la misma manera que ocurría en la transmisión, tampoco aquí hay acción por caracter transmitido, pero por cada paquete de comunicaciones recibido se produce la siguiente secuencia de acciones:

Interrupción nivel 7	*Tarea interrupción 7 - Procedimiento recibe*

Encola_rec	*Tarea interrupción 7*	*Interrupción nivel 7*

donde:

- *Interrupción nivel 7*. Es la acción del procedimiento de interrupción cuando se recibe un paquete de comunicaciones.
- *Tarea interrupción 7 - Procedimiento recibe*. Es la acción que realiza la tarea de interrupción *interrupcion_nivel_7* del paquete Ada *comunicaciones* cuando es llamada a causa de la recepción de un paquete de comunicaciones completo. Esta tarea, al igual que en las líneas serie, ejecuta el procedimiento *recibe* de los paquetes Ada *comunicaciones_recepcion_xxx*.
- *Encola_rec*. Es la misma acción que hemos descrito en el punto anterior.

El modelado de las comunicaciones no concluye con la extracción de las acciones que tienen lugar en las operaciones de comunicación; también hay que modelar los mensajes que circulan por las redes de comunicación. Así pues, para comenzar vamos a dar una relación con las características fundamentales asociadas a los mensajes de comunicación para las redes implementadas. Estas características son las siguientes:

- Unicamente las operaciones remotas dan lugar a la transmisión de mensajes por las redes de comunicación. Estas operaciones van codificadas en un protocolo de longitud l_{pro} caracteres (5 en nuestra implementación).

- Cada mensaje de comunicaciones está formado por el protocolo de codificación de las operaciones remotas más el mensaje propio de las colas de mensajes de longitud l_m (que puede contener un mensaje del usuario o información adicional de alguna operación concreta).

Protocolo (l_{pro})	*Mensaje* (l_m)

- Los mensajes de comunicaciones se parten en paquetes de longitud fija l_p, y cada paquete lleva una cabecera de identificación con el protocolo de comunicación de longitud l_{cab} caracteres (5 para nuestra implementación).

Cabecera (l_{cab})	*Paquete* (l_p)

- En las líneas serie RS-232 y RS-485 hay que considerar también dos caracteres más por cada paquete de comunicaciones que son *STX* y *ETX* (caracteres de comienzo y final de paquete respectivamente).

Otro aspecto que es importante poner de manifiesto es la longitud que tienen los mensajes de comunicación para las distintas operaciones sobre las colas de mensajes, ya sea en la orden de operación, o ya en su respuesta. Las longitudes de los mensajes de comunicación son las siguientes (--> orden de operación, <-- respuesta):

1.- Abre.

Tarea -->	*Protocolo* (l_{pro})	*Nombre de cola* (l_{nom})
Tarea <--	*Protocolo* (l_{pro})	

donde, l_{nom} es la longitud del mensaje correspondiente a la codificación del nombre de la cola de mensajes para su transmisión.

2.- Cierra.

Tarea -->	*Protocolo (l_{pro})*	
Error <--	*Protocolo (l_{pro})*	

3.- Envía.

Tarea -->	*Protocolo (l_{pro})*	*Mensaje (l_m)*
Error <--	*Protocolo (l_{pro})*	

4.- Recibe.

Tarea -->	*Protocolo (l_{pro})*	
Tarea <--	*Protocolo (l_{pro})*	*Mensaje (l_m)*

5.- Recibe si hay.

Tarea -->	*Protocolo (l_{pro})*	
Tarea <--	*Protocolo (l_{pro})*	*Si no hay mensaje.*
Tarea <--	*Protocolo (l_{pro})*	*Mensaje (l_m)*

6.- Coge atributos.

Tarea -->	*Protocolo (l_{pro})*	
Tarea <--	*Protocolo (l_{pro})*	*Atributos (l_{atrib})*

donde, l_{atrib} es la longitud del mensaje correspondiente a la codificación de los atributos de las colas de mensajes para su transmisión.

En consonancia con las longitudes de los mensajes de comunicaciones que acabamos de ver, el número de paquetes en los que se dividen éstos se calcula de la siguiente forma:

$$N_{paquetes} = \left\lceil \frac{l_{pro} + l_m}{l_p} \right\rceil$$

donde, l_p es la longitud del paquete de comunicaciones, l_m y l_{pro} son las longitudes del mensaje y del protocolo respectivamente (ver página 7-18), y $\lceil\ \rceil$ es el operador techo (redondeo al entero superior).

Para concluir este apartado, hacemos un breve comentario acerca de los tiempos de transmisión de mensajes por las redes implementadas. Para el bus VME podemos considerar que la transmisión de un paquete no consume tiempo, y únicamente consideraremos la sobrecarga que las tareas de comunicación suponen para los nudos correspondientes. En el caso de las líneas serie, el cálculo del tiempo de transmisión de un paquete se realiza a partir del baud-rate utilizado de acuerdo con la siguiente fórmula:

$$t_{paquete} = \frac{10}{baud\text{-}rate}\ l_p$$

donde, l_p es la longitud del paquete de comunicaciones, *baud-rate* es la velocidad de transmisión de la línea serie (número de bits por segundo), y 10 son los bits transmitidos por cada caracter. A este tiempo hay que añadir el consumido en la elaboración del paquete para la transmisión.

7.3. Evaluación de las colas de mensajes

La evaluación de las colas de mensajes se efectúa con la medición de los tiempos de peor caso del código correspondiente a las acciones definidas en el modelado de las colas de mensajes. Estos tiempos han sido medidos para la implementación realizada, con la ayuda de un emulador del microprocesador MC68010 de Motorola, en el que se basan las tarjetas que constituyen la plataforma de desarrollo utilizada.

El fundamento de la evaluación realizada se encuentra en la medida de los tiempos para las acciones definidas en el modelado, dentro del contexto de las operaciones que se llevan a cabo sobre las colas de mensajes y de las comunicaciones. También se han tenido en cuenta las condiciones del entorno a la hora de ejecutar una acción si éstas dan lugar a diferentes líneas de ejecución de la acción con tiempos de peor caso muy dispares.

Las operaciones sobre las colas de mensajes las vamos a evaluar según tres grandes apartados: operaciones locales, operaciones remotas en el nudo actual, y operaciones remotas en el nudo remoto. Para las comunicaciones consideraremos las distintas redes que forman parte de la implementación.

Antes de entrar de lleno en las operaciones sobre las colas de mensajes y en las comunicaciones, vamos a dar los tiempos medidos para algunas operaciones de configuración y de gestión de la memoria utilizadas por las colas de mensajes. Los tiempos medidos para estas operaciones son los que se detallan a continuación:

- Consulta del nudo actual (función *obten_nudo_actual* del paquete Ada *configuracion*):

  t_{nudo_actual} = 28 µs.

- Consulta del nudo asociado a una cola de mensajes en la configuración del sistema (función *obten_nudo* del paquete Ada *configuracion*):

  t_{nudo} = 52 µs. + (45 µs.) N_{elem_conf}

  donde, N_{elem_conf} es el número de elementos configurados recorridos hasta encontrar el que se busca.

- Consulta de la red asociada a una cola de mensajes en la configuración del sistema (función *obten_red* del paquete Ada *configuracion*):

  t_{red} = 49 µs. + (42 µs.) N_{elem_conf}

  donde, N_{elem_conf} es el definido en el punto anterior.

- Reserva de segmentos de memoria (procedimiento *reserva_segmentos* del paquete Ada *memoria*):

  $t_{reserva_seg}$ = 182 µs. + (36 µs.) $N_{seg_reservados}$

  donde, $N_{seg_reservados}$ es el número de segmentos de memoria reservados.

- Liberación de segmentos de memoria (procedimiento *libera_segmentos* del paquete Ada *memoria*):

  t_{libera_seg} = 125 µs. + (38 µs.) $N_{seg_liberados}$

donde, $N_{seg_liberados}$ es el número de segmentos de memoria liberados.

También damos el tiempo de peor caso medido para la ejecución del *tick* correspondiente al ejecutivo Ada cuando el período es de 20 ms. y tenemos 20 tareas en el procesador ejecutando la operación delay de forma alternada. Este tiempo es el siguiente:

t_{tick} = 470 µs.

Por último, en las operaciones sobre las colas de mensajes y de comunicaciones que vamos a exponer a continuación, mostramos en primer lugar las acciones que las modelan y seguidamente los tiempos asociados a dichas acciones. Cuando una misma acción tenga varios tramos de ejecución añadimos a su nombre un identificador del tramo de la manera siguiente: *_i* (inicial), *_m1* (medio 1), *_m2* (medio 2), ..., *_f* (final). La simbología y nomenclatura utilizadas para representar las secuencias de acciones o de tramos de acción es la definida en este mismo capítulo (ver página 7-14).

7.3.1. Operaciones locales

La evaluación de las operaciones locales sobre las colas de mensajes la podemos dividir en los puntos que se relacionan a continuación:

1.- Crea.

Operación *abre* en su opción de crear la cola de mensajes. La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 95 µs. + t_{nudo}
t_{colas_f} = 29 µs.

$t_{control}$ = 5882 µs. + $t_{reserva_seg}$ + (158 µs.) N_{colas} + (539 µs.) $N_{mensajes}$

donde,

$t_{reserva_seg}$ Es el tiempo de reserva de segmentos de memoria (ver página 7-21).

N_{colas} Es el número máximo de colas que admite el sistema.

$N_{mensajes}$ Es el número de mensajes con el que se crea la cola.

2.- Abre.

Operación *abre* en su opción de apertura de descriptor. La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

$$t_{colas_i} = 83\ \mu s. + t_{nudo}$$
$$t_{colas_f} = 29\ \mu s.$$

$$t_{control} = 2230\ \mu s. + (158\ \mu s.)\ N_{busca_cola}$$

donde,

t_{nudo} Es el tiempo consulta del nudo en el que está localizada una cola de mensajes (ver página 7-21).

N_{busca_cola} Es el número de colas recorridas en la búsqueda de la cola sobre la que se quiere abrir el descriptor hasta encontrar la que se busca. El peor caso es N_{colas}.

3.- Cierra.

La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

$$t_{colas_i} = 40\ \mu s.$$
$$t_{colas_f} = 20\ \mu s.$$

$$t_{control} = \begin{cases} 2234\ \mu s. & ;\ \textit{No destruye la cola} \\ 2323\ \mu s. + t_{libera_seg} & ;\ \textit{Destruye la cola} \end{cases}$$

donde,

t_{libera_seg} Es el tiempo de liberación de segmentos de memoria (ver página 7-21).

4.- Destruye.

La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 25 µs.
t_{colas_f} = 14 µs.

$t_{control}$ = 2138 µs. + t_{libera_seg} + (83 µs.) N_{busca_cola}

donde, t_{libera_seg} y N_{busca_cola} son los ya comentados en puntos anteriores.

5.- Coge atributos.

La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 43 µs.
t_{colas_f} = 20 µs.

$t_{control}$ = 1990 µs.

6.- Envía.

La secuencia de acciones de esta operación, que corresponde a los *tipos I* y *VI*, es la siguiente,

Colas_i	*Control_i*	*Conecta_red_tr*	*Control_f*	*Colas_f*

donde, *conecta_red_tr* indica la secuencia de acciones derivadas del uso de la red para la transmisión, que se verán en el apartado dedicado a las comunicaciones (ver página 7-35).

Los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 51 µs.
t_{colas_f} = 20 µs.

$$t_{control_i} = \begin{cases} 3247\ \mu s. + (209\ \mu s.)\ l_m & ;\ \textit{No hay espera} \\ 1166\ \mu s. + (113\ \mu s.)\ l_m & ;\ \textit{Espera remota} \\ 3818\ \mu s. + (209\ \mu s.)\ l_m + (90\ \mu s.)\ N_{busca_tarea} & ;\ \textit{Espera local} \end{cases}$$

$t_{control_f}$ = 1576 µs. *Si hay tarea en espera remota*

donde,

l_m Es la longitud del mensaje en los términos definidos en el apartado anterior (ver página 7-18).

N_{busca_tarea} Es el número de prioridades recorridas (desde la máxima) hasta encontrar la tarea de más prioridad en la cola de espera local.

Solamente para una secuencia de acciones del *tipo VI* tienen sentido los tiempos de *control_f* y de *red_trans*, es decir, cuando hay tareas en la cola de espera remota (ejecución de una operación *recibe* remota). En otro caso, puede ocurrir que haya tareas en la cola de espera local o que no haya tareas en espera, y ambas responden a una operación del *tipo I*.

7.- Recibe.

La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	< *Control* >	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 44 µs.
t_{colas_f} = 27 µs.

$t_{control}$ = 3228 µs. + (97 µs.) $N_{busca_mensaje}$ + (212 µs.) l_m + (2757 µs.) $N_{intentos}$

donde,

l_m Es la longitud del mensaje en los términos definidos en el apartado anterior (ver página 7-18).

$N_{intentos}$ Es el número de intentos de recibir mensaje, es decir, de llamadas a *control*.

$N_{busca_mensaje}$ Número de prioridades recorridas (desde la máxima) hasta encontrar el mensaje de más prioridad en la cola de mensajes sobre la que se realiza la operación.

8.- Recibe si hay.

La secuencia de acciones de esta operación, que corresponde al *tipo I*, es la siguiente,

Colas_i	*Control*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son,

t_{colas_i} = 44 µs.
t_{colas_f} = 27 µs.

$$t_{control} = \begin{cases} 2159\ \mu s. & ;\ \textit{No hay mensajes} \\ 3228\ \mu s. + (97\ \mu s.)\ N_{busca_mensaje} + (212\ \mu s.)\ l_m & ;\ \textit{Hay mensajes} \end{cases}$$

donde, l_m y $N_{busca_mensaje}$ son los mismos que en el punto anterior.

7.3.2. Operaciones remotas en el nudo actual

Por operaciones remotas en el nudo actual entendemos aquellas acciones que se llevan a cabo en el nudo en el que se ordena la realización de una operación sobre una cola de mensajes global que está en un nudo remoto. Al igual que para el apartado anterior, la evaluación de estas operaciones la podemos dividir en los puntos que se exponen a continuación:

1.- Abre.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo actual, es la siguiente,

Colas_i	*Abre_rem*	*Reserva_men*	*Colas_m1*	*Conecta_red_tr*

Colas_m2	*-- Suspensión --*	*Colas_m3*	*Libera_men*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son para la acción *colas*,

t_{colas_i} = 68 µs. + t_{nudo}
t_{colas_m1} = 1280 µs. + t_{red}
t_{colas_m2} = 121 µs.
t_{colas_m3} = 184 µs.
t_{colas_f} = 22 µs.

para la acción de apertura de un descriptor remoto,

t_{abre_rem} = 1090 µs.

y para la reserva y liberación de mensaje,

$t_{reserva_men}$ = 1070 µs.

t_{libera_men} = 1020 µs.

donde, t_{nudo} (ver página 7-21) y t_{red} (ver página 7-21) corresponden a los tiempos de consulta de nudo y red para una cola global en la configuración del sistema.

Conecta_red_tr representa la secuencia de acciones derivadas del uso de la red para la transmisión, que se verán en el apartado dedicado a las comunicaciones (ver página 7-35).

2.- Cierra.

Esta operación puede pertenecer a los *tipos IV* y *V*. La secuencia de acciones para la del *tipo IV*, que corresponde al tramo de acciones en el nudo actual, es la siguiente,

Colas_i	*Reserva_men*	*Colas_m*	*Conecta_red_tr*	*Cierra_rem*

Colas_f

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

t_{colas_i} = 56 µs.
t_{colas_m} = 305 µs.
t_{colas_f} = 18 µs.

y para la acción de cierre de un descriptor remoto,

t_{cierra_rem} = 1070 µs.

El tiempo $t_{reserva_men}$ y la acción *conecta_red_tr* coinciden con los detallados en la operación *abre* del punto anterior.

Si la operación corresponde al *tipo V*, es decir, se produce un error, habría que añadir las acciones correspondientes a la recepción de un mensaje que se relacionan en el apartado de comunicaciones (ver página 7-35).

3.- Coge atributos.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo actual, es la siguiente,

Colas_i	*Reserva_men*	*Colas_m1*	*Conecta_red_tr*	*Colas_m2*

-- *Suspensión* --	*Colas_m3*	*Libera_men*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

t_{colas_i} = 59 µs.
t_{colas_m1} = 307 µs.
t_{colas_m2} = 119 µs.
t_{colas_m3} = 379 µs.
t_{colas_f} = 20 µs.

Los tiempos $t_{reserva_men}$ y $t_{reserva_men}$, y la acción *conecta_red_tr* coinciden con los detallados en la operación *abre* del punto anterior.

4.- Envía.

Al igual que la operación *cierra*, esta operación puede pertenecer a los *tipos IV* y *V*, y además al *tipo III*. La secuencia de acciones para la de los *tipos III* y *IV* que corresponde al tramo de acciones en el nudo actual es la siguiente,

Colas_i	*Reserva_men*	*Colas_m*	*Conecta_red_tr*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

t_{colas_i} = 78 µs.
t_{colas_m} = 395 µs. + (108 µs.) l_m
t_{colas_f} = 18 µs.

donde, l_m es la longitud del mensaje (ver página 7-18).

El tiempo $t_{reserva_men}$ y la acción *conecta_red_tr* coinciden con los detallados en la operación *abre* de esta clasificación.

Si la operación corresponde al *tipo V*, de la misma forma que para la operación *cierra*, habría que añadir las acciones correspondientes a la recepción de un mensaje que se relacionan en el apartado de comunicaciones (ver página 7-35).

5.- Recibe.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo actual es la siguiente,

Colas_i	*Reserva_men*	*Colas_m1*	*Conecta_red_tr*	*Colas_m2*

-- *Suspensión* --	*Colas_m3*	*Libera_men*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

t_{colas_i} = 75 µs.
t_{colas_m1} = 305 µs.
t_{colas_m2} = 117 µs.
t_{colas_m3} = 243 µs. + (96 µs.) l_m
t_{colas_f} = 25 µs.

donde, l_m es la longitud del mensaje (ver página 7-18).

Los tiempos $t_{reserva_men}$ y t_{libera_men}, y la acción *conecta_red_tr* coinciden con los detallados en la operación *abre* de esta clasificación.

6.- Recibe si hay.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo actual, es la siguiente,

Colas_i	*Reserva_men*	*Colas_m1*	*Conecta_red_tr*	*Colas_m2*

-- *Suspensión* --	*Colas_m3*	*Libera_men*	*Colas_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

t_{colas_i} = 76 µs.
t_{colas_m1} = 305 µs.
t_{colas_m2} = 117 µs.

$$t_{colas_m3} = \begin{cases} 152\ \mu s. & ;\ No\ hay\ mensaje \\ 243\ \mu s. + (96\ \mu s.)\ l_m & ;\ Hay\ mensaje \end{cases}$$

t_{colas_f} = 20 µs.

donde, l_m es la longitud del mensaje (ver página 7-18).

Los tiempos $t_{reserva_men}$ y $t_{reserva_men}$, y la acción *conecta_red_tr* coinciden con los detallados en la operación *abre* de esta clasificación.

7.3.3. Operaciones remotas en el nudo remoto

Las operaciones remotas en el nudo remoto están constituidas por las acciones que se llevan a cabo en el nudo remoto, en el que se ejecuta una operación sobre una cola de mensajes global, que está en ese nudo, y que ha sido ordenada por una tarea de otro nudo (nudo actual). Vamos a considerar como ejecución en el nudo remoto las acciones realizadas por la tarea *sincroniza* del paquete Ada *control_de_colas* a través de su punto de entrada *ejecuta_remoto*. La llamada a este punto de entrada se encuadra en la secuencia de acciones que se muestra a continuación:

Ordena_rec_i	*Sinc_ejec_i*	*Ejecución*	*Sinc_ejec_f*	*Ordena_rec_f*

donde,

$t_{sinc_ejec_i}$ = 325 µs.
$t_{sinc_ejec_f}$ = 1520 µs.

son los tiempos de rendezvous con *ejecuta_remoto*, es decir, los tiempos de llamada y retorno a este punto de entrada. Al conjunto de acciones delimitadas por *ordena_rec_i* y *ordena_rec_f* las vamos a denominar *ejecuta*.

En los tiempos que vamos a dar para las operaciones remotas en el nudo remoto, incluimos los tiempos de rendezvous con *ejecuta_remoto* en las acciones *control_i* y *control_f*. Al igual que para los apartados anteriores, la evaluación de estas operaciones la podemos dividir en los puntos que se relacionan a continuación:

1.- Abre.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$t_{control_i}$ = 1719 µs. + (154 µs.) N_{busca_cola}
$t_{control_f}$ = 1538 µs.

donde, N_{busca_cola} es el número de colas recorridas hasta encontrar la que se busca (ver página 7-23).

Conecta_red_tr representa al igual que en casos anteriores la secuencia de acciones derivadas del uso de la red para la transmisión que se desarrolla en el subapartado de comunicaciones (ver página 7-35).

2.- Cierra.

Esta operación puede pertenecer a los *tipos IV* y *V*. La secuencia de acciones para la del *tipo IV*, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Libera_men*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$$t_{control_i} = \begin{cases} 946\ \mu s. & ;\ \textit{No destruye la cola de mensajes} \\ 1034\ \mu s. + t_{libera_seg} & ;\ \textit{Destruye la cola de mensajes} \end{cases}$$

$$t_{control_f} = 1538\ \mu s.$$

y para la liberación del mensaje,

$$t_{libera_men} = 496\ \mu s.$$

donde, t_{libera_seg} es el tiempo de liberación de segmentos de memoria (ver página 7-21).

Si la operación corresponde al *tipo V* la secuencia de acciones es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

donde, siguen siendo válidos los tiempos de peor caso de *control*, y habría que sustituir la liberación del mensaje por la transmisión del mismo a través de la red con el error correspondiente (*conecta_red_tr*, página 7-35).

3.- Coge atributos.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son,

$$t_{control_i} = 1336\ \mu s.$$
$$t_{control_f} = 1538\ \mu s.$$

Conecta_red_tr ya ha sido comentada en la descripción de la operación *abre* de esta clasificación.

4.- Envía.

De la misma forma que la operación *cierra*, esta operación puede pertenecer a los *tipos IV* y *V*, y además al *tipo III*.

En la operación correspondiente al tipo IV se pueden distinguir los siguientes casos:

- No hay elementos en las colas de espera, es decir, ninguna operación *recibe*, ni local, ni remota se encuentra esperando un mensaje. La secuencia de acciones, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Libera_men*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$t_{control_i}$ = 2017 µs.
$t_{control_f}$ = 1538 µs.

El tiempo t_{libera_men} es el mismo que el calculado para la operación *cierra* de esta clasificación.

- Hay elementos en la cola de espera local, es decir, que alguna operación *recibe* se encuentra esperando un mensaje. La secuencia de acciones, correspondiente al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Libera_men*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$t_{control_i}$ = 2059 µs. + (90 µs.) N_{busca_tarea} + (269 µs.) l_m
$t_{control_f}$ = 1538 µs.

donde,

l_m Es la longitud del mensaje (ver página 7-18).

N_{busca_cola} Es el número de colas recorridas hasta encontrar la que se busca (ver página 7-23).

Como en el caso anterior, el tiempo t_{libera_men} es el mismo que el calculado para la operación *cierra* de esta clasificación.

Si la operación corresponde al *tipo III*, es decir, hay elementos en la cola de espera remota (alguna operación *recibe* remota esperando un mensaje),

la secuencia de acciones, correspondiente al tramo de acciones del nudo remoto, es la siguiente,

Control_i	*Conecta_red_tr*	*Control_m*	*Libera_men*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$t_{control_i}$ = 1798 µs.
$t_{control_m}$ = 6 µs.
$t_{control_f}$ = 1538 µs.

El tiempo t_{libera_men} es el mismo que el calculado para la operación *cierra* de esta clasificación. En cuanto a *conecta_red_tr*, ya ha sido comentada en la descripción de la operación *abre*. En este caso, cabe destacar que no se efectúa copia del mensaje (se aprovecha el utilizado por *envía* para generar la respuesta de *recibe*).

Si la operación corresponde al *tipo V* la secuencia de acciones es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$t_{control_i}$ = 1798 µs.
$t_{control_f}$ = 1538 µs.

que coinciden con los tiempos de peor caso de *control* para el caso en el que hay elementos en la cola de espera remota. *Conecta_red_tr* es la comentada en los puntos anteriores.

5.- Recibe.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$$t_{control_i} = \begin{cases} 2486\ \mu s. & ;\ \textit{No hay mensaje} \\ 2185\ \mu s. + (96\ \mu s.)\ N_{busca_mensaje} + (277\ \mu s.)\ l_m & ;\ \textit{Hay mensaje} \end{cases}$$

$$t_{control_f} = 1540\ \mu s. \quad \textit{Si hay mensaje}$$

donde,

l_m Es la longitud del mensaje (ver página 7-18).

$N_{busca_mensaje}$ Es el número de prioridades recorridas para localizar el mensaje en la cola de mensajes (ver página 7-26).

La ejecución de las acciones *control_f* y *conecta_red_tr* (ya comentada en los puntos anteriores) va unida a la existencia de mensaje.

6.- Recibe si hay.

La secuencia de acciones de esta operación del *tipo II*, que corresponde al tramo de acciones en el nudo remoto, es la siguiente,

Control_i	*Conecta_red_tr*	*Control_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *colas*,

$$t_{control_i} = \begin{cases} 956\ \mu s. & ;\ \textit{No hay mensaje} \\ 2273\ \mu s. + (96\ \mu s.)\ N_{busca_mensaje} + (277\ \mu s.)\ l_m & ;\ \textit{Hay mensaje} \end{cases}$$

$$t_{control_f} = 1540\ \mu s.$$

donde, l_m y $N_{busca_mensaje}$ son los mismos que en el punto anterior.

7.3.4. Comunicaciones

En el apartado de las comunicaciones vamos a evaluar los tiempos que se consumen, tanto en las acciones de transmisión de paquetes, como en las de transmisión de caracteres cuando corresponda. Teniendo en cuenta las redes de comunicación que hemos implementado consideramos los siguientes puntos:

1.- Encolado de paquetes de comunicación para la transmisión.

En este punto se evalúan el conjunto de acciones que llevan a cabo la partición del mensaje de comunicaciones en paquetes, y que vamos a denominar globalmente *conecta_red_tr*. La secuencia de acciones a la que nos referimos es la siguiente,

Red_tr_i	*Com_tr*	*Red_tr_f*

y los tiempos asociados a las acciones o tramos de acción correspondientes a *red_tr* son,

$t_{red_tr_i}$ = 699 µs.
$t_{red_tr_f}$ = 20 µs.

y para *com_tr* tenemos que distinguir los tipos de red implementados, tal como se muestra a continuación:

- RS-232 y RS-485

$$t_{com_tr} = \begin{cases} 1316 \text{ µs.} & ; \textit{No envía el primer paquete} \\ K_{com_tr} + (112 \text{ µs.})\ l_p & ; \textit{Envía el primer paquete} \end{cases}$$

donde,

$$K_{com_tr} = \begin{cases} 2648 \text{ µs.} & ; \textit{No libera mensaje} \\ 3149 \text{ µs.} & ; \textit{Libera mensaje} \end{cases}$$

- Bus VME

$$t_{com_tr} = \begin{cases} 1501 \text{ µs.} & ; \textit{No envía el primer paquete} \\ K_{com_tr} + (112 \text{ µs.})\ l_p & ; \textit{Envía el primer paquete} \end{cases}$$

donde,

$$K_{com_tr} = \begin{cases} 2832 \text{ µs.} & ; \textit{No libera mensaje} \\ 3373 \text{ µs.} & ; \textit{Libera mensaje} \end{cases}$$

siendo l_p la longitud del paquete de comunicaciones en los términos definidos en el apartado anterior (ver página 7-18).

2.- Transmisión en las líneas serie.

La transmisión en las líneas serie consiste en la extracción de los paquetes de comunicación de la cola de transmisión y en su envío caracter a caracter, y se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Ordena_tr*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *int*,

$$t_{int_i} = \begin{cases} 231\ \mu s. & ;\ \textit{No llama a Tarea_int} \\ 208\ \mu s. & ;\ \textit{Llama a Tarea_int} \end{cases}$$

$$t_{int_f} = 24\ \mu s.$$

para la acción *tarea_int*,

$$t_{tarea_int_i} = 750\ \mu s.$$
$$t_{tarea_int_f} = 100\ \mu s.$$

y finalmente, para la acción *ordena_tr*,

$$t_{ord_tr} = \begin{cases} 526\ \mu s. & ;\ \textit{No quedan paquetes} \\ K_{ord_tr} + (26\ \mu s.)\ N_p + (126\ \mu s.)\ l_p & ;\ \textit{Quedan paquetes} \end{cases}$$

donde,

$$K_{ord_tr} = \begin{cases} 1931\ \mu s. & ;\ \textit{No es el último paquete} \\ 2755\ \mu s. & ;\ \textit{Ultimo paquete (liberando mensaje)} \\ 2251\ \mu s. & ;\ \textit{Ultimo paquete (sin liberar mensaje)} \end{cases}$$

y,

l_p Es la longitud del paquete (ver página 7-18).

N_p Es el número de prioridades recorridas (desde la máxima) hasta la del paquete en disposición de ser transmitido.

3.- Transmisión en el bus VME.

La transmisión en el bus VME consiste en la extracción de los paquetes de comunicación de la cola de transmisión y su envío en bloque. Se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Ordena_tr*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *int*,

t_{int_i} = 49 µs.
t_{int_f} = 24 µs.

para la acción *tarea_int*,

$t_{tarea_int_i}$ = 638 µs.
$t_{tarea_int_f}$ = 100 µs.

y finalmente, para la acción *ordena_tr*,

$$t_{ord_tr} = \begin{cases} 574\ \mu s. + (26\ \mu s.)\ N_n & ;\ \textit{No quedan paquetes} \\ 605\ \mu s. + (64\ \mu s.)\ N_n & ;\ \textit{Transfiere el control} \\ K_{ord_tr} + t_{busca} + (64\ \mu s.)\ N_n + (126\ \mu s.)\ l_p & ;\ \textit{Transmite paquete} \end{cases}$$

donde,

$$t_{busca} = \begin{cases} 41\ \mu s. & ;\ \textit{Cola de transmisión vacía} \\ 27\ \mu s. + (26\ \mu s.)\ N_p & ;\ \textit{Cola de transmisión no vacía} \end{cases}$$

$$K_{ord_tr} = \begin{cases} 1899\ \mu s. & ;\ \textit{No es el último paquete} \\ 2685\ \mu s. & ;\ \textit{Ultimo paquete (liberando mensaje)} \\ 2144\ \mu s. & ;\ \textit{Ultimo paquete (sin liberar mensaje)} \end{cases}$$

y,

l_p Es la longitud del paquete (ver página 7-18).

N_p Es el definido en el punto anterior.

N_n Es el número de nudos conectados al bus VME.

4.- Recepción en las líneas serie.

La recepción en las líneas serie consiste en el encolado de los mensajes de comunicación en la cola de recepción, después de su recomposición a partir de los paquetes recibidos caracter a caracter. Se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Encola_rec*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *int*,

$$t_{int_i} = \begin{cases} 231\ \mu s. & ;\ \textit{No llama a Tarea_int} \\ 208\ \mu s. & ;\ \textit{Llama a Tarea_int} \end{cases}$$

t_{int_f} = 24 µs.

para la acción *tarea_int*,

$$t_{tarea_int_i} = \begin{cases} 1979\ \mu s. + (117\ \mu s.)\ l_p & ;\ \textit{Reserva mensaje} \\ 1496\ \mu s. + (117\ \mu s.)\ l_p & ;\ \textit{No reserva mensaje} \end{cases}$$

$t_{tarea_int_f}$ = 100 µs.

y finalmente, para la acción *encola_rec*,

$$t_{encol_rec} = \begin{cases} 710\ \mu s. & ;\ \textit{Se están ejecutando mensajes} \\ 848\ \mu s. & ;\ \textit{Inicia la ejecución de mensajes} \end{cases}$$

donde l_p es la longitud del paquete (ver página 7-18).

5.- Recepción en bus VME.

La recepción en el bus VME consiste en el encolado de los mensajes de comunicación en la cola de recepción después de su recomposición a partir de los paquetes recibidos en bloque. Se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Encola_rec*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones o tramos de acción son, para la acción *int*,

t_{int_i} = 49 µs.

t_{int_f} = 24 µs.

para la acción *tarea_int*,

$$t_{tarea_int_i} = \begin{cases} 1864\ \mu s. + (117\ \mu s.)\ l_p & ;\ \textit{Reserva mensaje} \\ 1382\ \mu s. + (117\ \mu s.)\ l_p & ;\ \textit{No reserva mensaje} \end{cases}$$

$t_{tarea_int_f}$ = 100 µs.

y finalmente, para la acción *encola_rec* (igual que para las líneas serie),

$$t_{encol_rec} = \begin{cases} 710\ \mu s. & ;\ \textit{Se están ejecutando mensajes} \\ 848\ \mu s. & ;\ \textit{Inicia la ejecución de mensajes} \end{cases}$$

donde l_p es la longitud del paquete (ver página 7-18).

6.- Proceso de ejecución de mensajes de comunicación.

Consideramos en este punto las acciones correspondientes a la canalización de los mensajes de comunicación recibidos, para la ejecución de las órdenes remotas o de las respuestas a éstas que contienen. La secuencia de acciones hasta la llamada a *control* (en el caso de que proceda) es la siguiente,

Ordena_rec_i	*Desencola_rec*	*Ordena_rec_m*	*Ejecuta*	*Ordena_rec_f*
			Libera_men	

y los tiempos asociados a las acciones o tramos de acción son, para el tramo de acción *ordena_rec_i*,

$$t_{ordena_rec_i} = 5 \ \mu s.$$

para la acción *desencola_rec*,

$$t_{desencola_rec} = \begin{cases} 1196 \ \mu s. & ; \textit{Fin de ejecución} \\ 1363 \ \mu s. & ; \textit{Desencola un mensaje} \end{cases}$$

y para el resto de las acciones vamos a considerar los siguientes casos:

- Si hay que despertar a una tarea de usuario para que reciba la respuesta a una operación que ha realizado, el tiempo para la única acción que se lleva a cabo es,

  $$t_{ordena_rec_f} = 596 \ \mu s.$$

- Si tiene que realizar el conjunto de acciones de *ejecuta*, evaluadas para cada operación en el apartado de operaciones remotas en el nudo remoto (ver página 7-30), los tiempos para las acciones que se llevan a cabo son,

  $$t_{ordena_rec_m} = 112 \ \mu s.$$

  $$t_{ordena_rec_f} = 26 \ \mu s.$$

  más el tiempo correspondiente a *ejecuta*.

- Si es un error de una operación que no espera la respuesta, hay que liberar el mensaje de comunicaciones y los tiempos de las acciones son,

  $$t_{ordena_rec_m} = 256 \ \mu s.$$

$t_{ordena_rec_f}$ = 20 μs.

t_{libera_men} = 1020 μs.

7.4. Modelado y evaluación del servidor esporádico para comunicaciones

El servidor esporádico para comunicaciones consta de un conjunto de operaciones para la gestión de una cola de paquetes. Así pues, cada operación sobre esa cola se modela como un tramo de acción de la tarea que la realiza. En el servidor esporádico implementado para redes punto a punto esto es estrictamente así, pero en el implementado para redes multipunto hay que considerar además la tarea de control de rellenos.

Las operaciones que se realizan sobre la cola de paquetes de que dispone el servidor esporádico para comunicaciones son *introduce* y *extrae* (de introducción y extracción de paquetes de comunicaciones de la cola), pero es necesario tener en cuenta otro conjunto de operaciones utilizadas por éstas, que vamos a enumerar a continuación:

- Operaciones de manejo de la cola de prioridad con la que se implementa la cola de rellenos:

  Read Lectura del elemento de mayor prioridad.

  Enqueue Encola un elemento en su prioridad.

  Dequeue Desencola el elemento de mayor prioridad.

- Operaciones de control de rellenos:

  Planifica un relleno

  Ejecuta rellenos

  Ejecuta rellenos temporizado (sólo para redes multipunto)

Además, en el servidor esporádico implementado para las redes multipunto hay que considerar una operación adicional que es la consulta del paquete que está dispuesto para ser extraído, es decir, la operación *estado*.

El modelado de las comunicaciones que utilizan servidor esporádico no cambia con respecto al de las que no lo utilizan, ya que la única diferencia en el uso del servidor esporádico radica en la cola de transmisión y en su gestión.

7.4.1. Evaluación del servidor esporádico para comunicaciones

En este apartado evaluamos las operaciones que son relevantes en la utilización del servidor esporádico para comunicaciones, es decir aquellas dependientes de algún parámetro del servidor y que no pueden ser evaluadas de forma transparente en el contexto de las comunicaciones.

Comenzamos por la evaluación de las operaciones sobre la cola de prioridad que constituye la cola de rellenos, y que son las que se detallan a continuación:

- Lectura del elemento de mayor prioridad,

  $t_{read} = 68\ \mu s.$

- Encolado de un elemento en su prioridad,

$$t_{enqueue} = \begin{cases} 138\ \mu s. & ;\ Cola\ vacía \\ (303\ \mu s.)\ \ln(N_r+1) & ;\ Cola\ no\ vacía \end{cases}$$

- Desencolado del elemento de mayor prioridad:

$$t_{dequeue} = \begin{cases} 246\ \mu s. & ;\ Cola\ vacía \\ (355\ \mu s.)\ \ln(N_r+1) & ;\ Cola\ no\ vacía \end{cases}$$

donde, N_r es el número de rellenos encolados en el instante de la llamada a la operación.

Para la evaluación de las operaciones sobre los rellenos vamos a considerar los tipos de redes que hemos implementado de la siguiente forma:

1.- Redes punto a punto.

Los tiempos medidos en este tipo de redes para las operaciones de las que estamos tratando son, para la planificación de un relleno,

$$t_{plan_relleno} = 342\ \mu s. + t_{enqueue}$$

y para la ejecución de rellenos,

$$t_{ejec_rellenos} = \begin{cases} 54\ \mu s. & ;\ Cola\ vacía \\ 36\ \mu s. + (N_{desencola}+1)\ t_{consulta} + N_{desencola}\ t_{desencola} & ;\ Cola\ no\ vacía \end{cases}$$

donde,

$$t_{consulta} = 2056\ \mu s. + t_{read}$$

$$t_{desencola} = 279\ \mu s. + t_{dequeue}$$

y,

t_{read}, $t_{enqueue}$ y $t_{dequeue}$ Son los tiempos evaluados anteriormente para la cola de prioridad con la que se implementa la cola de rellenos.

$N_{desencola}$ Es el número de rellenos desencolados y ejecutados.

2.- Redes multipunto.

Los tiempos medidos en este tipo de redes para las operaciones de las que estamos tratando son, para la planificación de un relleno,

$$t_{plan_relleno} = 3286\ \mu s. + t_{enqueue}$$

para la ejecución forzada de rellenos,

$$t_{ejec_rellenos} = \begin{cases} 880\ \mu s. & ;\ Cola\ vacía \\ 853\ \mu s. + (N_{desencola}+1)\ t_{consulta} + N_{desencola}\ t_{desencola} & ;\ Cola\ no\ vacía \end{cases}$$

y para la ejecución temporizada de rellenos,

$$t_{ejec_rellenos_temp} = \begin{cases} 778\ \mu s. + t_{ejec_rellenos} & ;\ No\ hay\ cambio\ a\ normal \\ 912\ \mu s. + t_{ejec_rellenos} & ;\ Hay\ cambio\ a\ normal \end{cases}$$

donde,

$$t_{consulta} = 2139\ \mu s. + t_{read}$$

y,

t_{read}, $t_{enqueue}$ y $t_{dequeue}$	Son los tiempos evaluados anteriormente para la cola de prioridad con la que se implementa la cola de rellenos.
$t_{desencola}$ y $N_{desencola}$	Son los mismos que para las redes punto a punto.

La evaluación de las operaciones sobre la cola de transmisión implementada en el servidor esporádico de comunicaciones queda en función de las operaciones de utilidad ya evaluadas, tal como se muestra a continuación:

Introduce.

$$t_{intro} = \begin{cases} 514\ \mu s. & ;\ \textit{Estado no inactivo} \\ 2849\ \mu s. + t_{ejec_rellenos} & ;\ \textit{Estado inactivo (paso a normal)} \\ 661\ \mu s. + t_{ejec_rellenos} & ;\ \textit{Estado inactivo (paso a background)} \end{cases}$$

Extrae.

$$t_{extrae} = \begin{cases} 647\ \mu s. + t_{ejec_rellenos} + N_p\ t_{iter} & ;\ \textit{No planifica relleno} \\ 1062\ \mu s. + t_{ejec_rellenos} + t_{plan_relleno} + N_p\ t_{iter} & ;\ \textit{Planifica relleno} \end{cases}$$

donde,

$$t_{iter} = \begin{cases} 210\ \mu s. & ;\ \textit{Antes de encontrar paquete en background} \\ 146\ \mu s. & ;\ \textit{Después de encontrar paquete en background} \end{cases}$$

y,

$t_{plan_relleno}$ y $t_{ejec_rellenos}$	Son los tiempos evaluados anteriormente dependientes de la red utilizada.
N_p	Es el número de prioridades recorridas (desde la máxima) hasta la del paquete en disposición de ser transmitido (ver página 7-37).

Por último, evaluamos la operación *estado* definida únicamente para el servidor esporádico en las redes multipunto:

$$t_{estado} = 48\ \mu s. + t_{ejec_rellenos} + N_p\ t_{iter}$$

donde, N_p y t_{iter} son los vistos anteriormente.

7.4.2. Evaluación de las comunicaciones que utilizan servidor esporádico

Como ya hemos dicho el modelado de las comunicaciones no cambia por el hecho de utilizar el servidor esporádico. Sin embargo, en la evaluación de algunas acciones sí que se producen modificaciones debido al uso de las operaciones de éste. En este apartado vamos a evaluar solamente estos tiempos diferentes para las acciones de encolado y transmisión de paquetes. Teniendo en cuenta las redes de comunicación que hemos implementado vamos a considerar los siguientes puntos:

1.- Encolado de paquetes de comunicación para la transmisión.

Al igual que cuando no se utiliza servidor esporádico, en este punto se evalúan el conjunto de acciones que llevan a cabo la partición del mensaje de comunicaciones en paquetes, y que denominamos globalmente *conecta_red_tr*. La secuencia de acciones a la que nos referimos es la siguiente,

Red_tr_i	*Com_tr_ss*	*Red_tr_f*

y los tiempos asociados a los tramos de acción correspondientes a *red_tr* son los mismos que para el caso en que no se utiliza servidor esporádico (ver página 7-36), y para *com_tr_ss* tenemos que distinguir los tipos de red implementados, tal como se muestra a continuación:

RS-232 y RS-485

$$t_{com_tr_ss} = \begin{cases} 1293\ \mu s. + p_{intro}\ t_{intro} & ;\ No\ envía\ el\ 1^{er}\ paq. \\ K_{com_tr_ss} + (112\ \mu s.)\ l_p + p_{intro}\ t_{intro} + t_{extrae} & ;\ Envía\ el\ 1^{er}\ paq. \end{cases}$$

donde,

$$K_{com_tr_ss} = \begin{cases} 2739\ \mu s. & ;\ No\ libera\ mensaje \\ 3239\ \mu s. & ;\ Libera\ mensaje \end{cases}$$

Bus VME

$$t_{com_tr_ss} = \begin{cases} 1341\ \mu s. + p_{intro}\ t_{intro} + t_{estado} & ;\ No\ envía\ el\ 1^{er}\ paq. \\ K_{com_tr_ss} + (112\ \mu s.)\ l_p + p_{intro}\ t_{intro} + t_{extrae} & ;\ Envía\ el\ 1^{er}\ paq. \end{cases}$$

donde,

$$K_{com_tr_ss} = \begin{cases} 2624\ \mu s. & ;\ \textit{No libera mensaje} \\ 3120\ \mu s. & ;\ \textit{Libera mensaje} \end{cases}$$

y donde para ambos casos,

l_p Es la longitud del paquete de comunicaciones en los términos definidos en la página 7-18.

p_{intro} Es el número de paquetes que se introducen en la cola del servidor.

t_{intro} ,t_{extrae} y t_{estado} Son los tiempos asociados a las operaciones *introduce*, *extrae* y *estado* del servidor esporádico para comunicaciones (ver página 7-44).

2.- Transmisión en las líneas serie.

De la misma manera que cuando no se utiliza el servidor esporádico, la transmisión en las líneas serie consiste en la extracción de los paquetes de comunicación de la cola de transmisión y su envío caracter a caracter, y se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Ordena_tr_ss*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones *int* y *tarea_int* son los mismos que para el caso en que no se utiliza servidor esporádico (ver página 7-36), y para *ordena_tr_ss*,

$$t_{ord_tr_ss} = \begin{cases} 526\ \mu s. & ;\ \textit{No quedan paquetes} \\ K_{ord_tr_ss} + t_{extrae} + (126\ \mu s.)\ l_p & ;\ \textit{Quedan paquetes} \end{cases}$$

donde,

$$K_{ord_tr_ss} = \begin{cases} 1931\ \mu s. & ;\ \textit{No es el último paquete} \\ 2755\ \mu s. & ;\ \textit{Ultimo paquete (liberando mensaje)} \\ 2251\ \mu s. & ;\ \textit{Ultimo paquete (sin liberar mensaje)} \end{cases}$$

y,

l_p Es la longitud del paquete de comunicaciones (ver página 7-18).

t_{extrae} Es el tiempo empleado en la extracción de un paquete de la cola del servidor esporádico para comunicaciones (ver página 7-44).

3.- Transmisión en el bus VME.

De la misma manera que cuando no se utiliza el servidor esporádico, la transmisión en el bus VME consiste en la extracción de los paquetes de comunicación de la cola de transmisión y su envío en bloque, y se compone de la siguiente secuencia de acciones,

Int_i	*Tarea_int_i*	*Ordena_tr_ss*	*Tarea_int_f*	*Int_f*

y los tiempos asociados a las acciones *int* y *tarea_int* son los mismos que para el caso en que no se utiliza servidor esporádico (ver página 7-37), y para *ordena_tr_ss*,

$$t_{ord_tr_ss} = \begin{cases} 575\ \mu s. + (26\ \mu s.)\ N_n & ;\ \textit{No quedan paquetes} \\ 606\ \mu s. + (78\ \mu s.)\ N_n & ;\ \textit{Transfiere el control} \\ K_{ord_tr_ss} + t_{busca} + t_{extrae} + (78\ \mu s.)\ N_n + (126\ \mu s.)\ l_p & ;\ \textit{Transmite} \end{cases}$$

donde,

$$t_{busca} = \begin{cases} 56\ \mu s. & ;\ \textit{Cola de transmisión vacía} \\ 42\ \mu s. + t_{estado} & ;\ \textit{Cola de transmisión no vacía} \end{cases}$$

$$K_{ord_tr_ss} = \begin{cases} 1738\ \mu s. & ;\ \textit{No es el último paquete} \\ 2357\ \mu s. & ;\ \textit{Ultimo paquete (liberando mensaje)} \\ 1853\ \mu s. & ;\ \textit{Ultimo paquete (sin liberar mensaje)} \end{cases}$$

y,

l_p Es la longitud del paquete de comunicaciones (página 7-18).

N_p Es el número de prioridades recorridas (desde la máxima) hasta la del paquete en disposición de ser transmitido (ver página 7-37).

N_n Es el número de nudos conectados al bus VME (ver página 7-38).

t_{extrae} y t_{estado} Son los tiempos asociados a las operaciones *extrae* y *estado* del servidor esporádico para comunicaciones (ver página 7-44).

7.4.3. Estudio comparativo de las comunicaciones que utilizan servidor esporádico

El uso del servidor esporádico en las comunicaciones consigue eliminar el jitter y como consecuencia aumentar considerablemente la utilización de los recursos (procesadores y redes) del sistema distribuido, pero a la vez su implementación supone una sobrecarga para el sistema. Vamos a evaluar esta sobrecarga a través de varios ejemplos en los que calcularemos los tiempos de transmisión de mensajes de diferentes longitudes por las redes implementadas con y sin servidor esporádico.

El tiempo consumido en la transmisión de un mensaje lo contaremos a partir de la solicitud de transmisión por la red (utilización del procedimiento *transmite* del paquete Ada *red_transmision*), y para su cálculo habrá que tener en cuenta tanto las condiciones de encolado como las de desencolado y posterior transmisión de los paquetes de comunicaciones. Las características particulares que vamos a considerar para los mensajes y su transmisión son las que se detallan a continuación:

- Hay la suficiente actividad en la red como para que nunca se envíe el primer paquete cuando se encola un mensaje. Con esta suposición hacemos más simple el cálculo de tiempos de transmisión al tener en cuenta una sola posibilidad en el encolado del mensaje.

- Siempre quedan paquetes por transmitir, y la transmisión del último paquete libera el mensaje. Esta suposición también simplifica el cálculo de tiempos de transmisión.

- La prioridad de los mensajes es la máxima.

- Tenemos dos nudos conectados al bus VME.

- En el bus siempre se transmite un paquete y no se considera la transferencia de control. La transferencia de control se debe realizar tanto si se utiliza servidor esporádico como si no se utiliza, por lo que no constituye un punto diferenciador de las situaciones que queremos comparar.

- En la consulta del estado del servidor esporádico no se ejecutan rellenos. La ejecución de los rellenos se considerará de manera aparte para cada mensaje.

- El número de elementos en la cola de paquetes del servidor esporádico siempre es 10. También es siempre 10 el número de elementos de la cola de rellenos. Consideramos esta situación a efectos únicamente del cálculo de los tiempos de encolado y desencolado tanto de los paquetes como de los rellenos, lo que supone una cierta actividad en el servidor.

- Para el servidor esporádico suponemos que con cada mensaje se planifica y se ejecuta un relleno.

- De acuerdo con las premisas anteriores, el tiempo consumido en la transmisión de un mensaje sin servidor esporádico viene dado por,

  $$t_{trans} = t_{em} + t_{tp}$$

  donde,

  t_{em} Es el tiempo consumido en el encolado del mensaje partido en paquetes en la cola de transmisión. Este tiempo se calcula a partir del modelado realizado teniendo en cuenta los siguientes tiempos: $t_{red_tr_i}$, $t_{red_tr_f}$ y t_{com_tr}.

  t_{tp} Es el tiempo empleado para la transmisión de los paquetes que componen el mensaje. Este tiempo se calcula a partir del modelado realizado teniendo en cuenta los tiempos siguientes: t_{int_i}, t_{int_f}, $t_{tarea_int_i}$, $t_{tarea_int_f}$ y t_{ord_tr}.

- De igual modo que para el caso anterior, el tiempo consumido en la transmisión de un mensaje con servidor esporádico viene dado por,

  $$t_{trans_ss} = t_{em_ss} + t_{tp_ss}$$

  donde,

  t_{em_ss} Es el tiempo consumido en el encolado del mensaje partido en paquetes en la cola del servidor. Este tiempo se calcula a partir del modelado realizado teniendo en cuenta los siguientes tiempos: $t_{red_tr_i}$, $t_{red_tr_f}$ y $t_{com_tr_ss}$.

  t_{tp_ss} Es el tiempo empleado para la transmisión de los paquetes que componen el mensaje. Este tiempo se calcula a partir del modelado realizado teniendo en cuenta los tiempos siguientes: t_{int_i}, t_{int_f}, $t_{tarea_int_i}$, $t_{tarea_int_f}$ y $t_{ord_tr_ss}$.

- En el cálculo de los tiempos consumidos en la transmisión consideramos que la copia de los caracteres en el buffer de transmisión se realiza mediante un código optimizado que emplea 4µs. por caracter copiado.

- El tiempo de consulta del reloj que se utiliza servidor esporádico es anormalmente grande (2.056 ms.), por lo tanto, calculamos dos tiempos diferentes: el consumido en la transmisión cuando se utiliza este reloj, y el consumido cuando se usa un reloj con tiempo de consulta muy pequeño (idealmente suponemos que vale cero).

- La elección de la longitud de los paquetes para las distintas redes se realiza tratando de mantener constante la relación entre el tiempo de preparación de un paquete para su transmisión y el tiempo necesario para su transmisión propiamente dicha (copia del paquete al buffer de transmisión y procesado de las interrupciones para las líneas serie, y únicamente copia del paquete en el caso del bus VME). Así pues, las longitudes de los paquetes que vamos a utilizar en este estudio son:

  20 y 200 caracteres por paquete para las líneas serie.

  1000 y 10000 caracteres por paquete para el bus VME.

- Las longitudes de los mensajes se eligen de manera que se mantenga la correspondencia en número de paquetes para los dos tipos de red utilizados. Consideramos mensajes formados por 1, 5 y 10 paquetes para la longitud de paquete más pequeña, y además el mensaje de longitud 0 que se genera en las solicitudes de operación remota o en la respuesta sin mensaje. En todos los mensajes hay que tener en cuenta que el protocolo de codificación de las operaciones sobre las colas de mensajes (ver página 7-18) está formado por 5 caracteres. De esta forma, las longitudes l_m de los mensajes (de las colas de mensajes) sobre los que vamos a realizar este estudio son:
  - Para las líneas serie: 0, 15, 95 y 195.
  - Para el bus VME: 0, 995, 4995 y 9995.

Una vez definidas las características de los mensajes sobre los que hacemos el estudio y las condiciones de su transmisión, determinamos los tiempos consumidos en dicha transmisión para los dos tipos de redes implementados. Estos tiempos son los siguientes:

1.- Líneas serie.

Los tiempos que aparecen entre paréntesis en el servidor esporádico corresponden al caso en el que se considera el reloj con tiempo de consulta cero. En las condiciones de transmisión que hemos supuesto se realiza una consulta del reloj por paquete transmitido, salvo para el último paquete en el que se realizan dos consultas.

Sin servidor esporádico (20 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em} (ms.)	t_{tp} (ms.)	t_{trans} (ms.)
0 / 1	2.035	6.655	8.690
15 / 1	2.035	10.180	12.215
95 / 5	2.035	47.604	49.639
195 / 10	2.035	94.384	96.419
Con servidor esporádico (20 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em_ss} (ms.)	t_{tp_ss} (ms.)	t_{trans_ss} (ms.)
0 / 1	2.526 (2.526)	14.664 (10.552)	17.190 (13.078)
15 / 1	2.526 (2.526)	18.189 (14.077)	20.715 (16.603)
95 / 5	4.582 (4.582)	67.577 (55.241)	72.159 (59.823)
195 / 10	7.152 (7.152)	129.312 (106.696)	136.464 (113.848)

Sin servidor esporádico (200 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em} (ms.)	t_{tp} (ms.)	t_{trans} (ms.)
0 / 1	2.035	6.655	8.690
15 / 1	2.035	10.180	12.215
95 / 1	2.035	28.980	31.015
195 / 1	2.035	52.480	54.515
Con servidor esporádico (200 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em_ss} (ms.)	t_{tp_ss} (ms.)	t_{trans_ss} (ms.)
0 / 1	2.526 (2.526)	14.664 (10.552)	17.190 (13.078)
15 / 1	2.526 (2.526)	18.189 (14.077)	20.715 (16.603)
95 / 1	2.526 (2.526)	36.989 (32.877)	39.515 (35.403)
195 / 1	2.526 (2.526)	60.489 (56.377)	63.015 (58.903)

2.- Bus VME.

Como para el caso anterior los tiempos que aparecen entre paréntesis en el servidor esporádico corresponden al caso en el que se considera el reloj con tiempo de consulta cero. Aquí además de las consultas del reloj que se realizan en el caso anterior, se efectúa otra en el encolado del mensaje.

Sin servidor esporádico (1000 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em} (ms.)	t_{tp} (ms.)	t_{trans} (ms.)
0 / 1	2.220	3.697	5.917
995 / 1	2.220	7.677	9.897
4995 / 5	2.220	35.241	37.461
9995 / 10	2.220	69.696	71.916
Con servidor esporádico (1000 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em_ss} (ms.)	t_{tp_ss} (ms.)	t_{trans_ss} (ms.)
0 / 1	5.892 (3.836)	18.138 (14.026)	24.030 (17.862)
995 / 1	5.892 (3.836)	22.118 (18.006)	28.010 (21.842)
4995 / 5	7.948 (5.892)	90.402 (78.066)	98.350 (83.958)
9995 / 10	10.518 (8.462)	175.757 (153.141)	186.275 (161.603)

Sin servidor esporádico (10000 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em} (ms.)	t_{tp} (ms.)	t_{trans} (ms.)
0 / 1	2.220	3.697	5.917
995 / 1	2.220	7.677	9.897
4995 / 1	2.220	23.677	25.897
9995 / 1	2.220	43.677	45.897
Con servidor esporádico (10000 caracteres por paquete)			
l_m / **Nº paquetes**	t_{em_ss} (ms.)	t_{tp_ss} (ms.)	t_{trans_ss} (ms.)
0 / 1	5.892 (3.836)	18.138 (14.026)	24.030 (17.862)
995 / 1	5.892 (3.836)	22.118 (18.006)	28.010 (21.842)
4995 / 1	5.892 (3.836)	38.118 (34.006)	44.010 (37.842)
9995 / 1	5.892 (3.836)	58.118 (54.006)	64.010 (57.842)

Una vez calculados los tiempos consumidos en la transmisión, pasamos a evaluar la sobrecarga que supone para el sistema la utilización del servidor esporádico para comunicaciones. Esta sobrecarga es el exceso de tiempo consumido en la transmisión con servidor esporádico sobre el consumido en la transmisión sin servidor, relativo a este último tiempo y en tantos por cien. Se puede expresar de la forma siguiente:

$$Sobrecarga = \frac{t_{trans_ss} - t_{trans}}{t_{trans}} \; 100$$

Así pues, la sobrecarga del sistema debida al uso del servidor esporádico para comunicaciones en las diferentes redes es la siguiente:

1.- Líneas serie.

20 caracteres por paquete			
l_m / **Nº paquetes**	t_{trans} (ms.)	t_{trans_ss} (ms.)	**Sobrecarga** (%)
0 / 1	8.609	17.190 (13.078)	99.67 (51.91)
15 / 1	12.215	20.715 (16.603)	69.59 (35.92)
95 / 5	49.639	72.159 (59.823)	45.37 (20.51)
195 / 10	96.419	136.464 (113.848)	41.53 (18.08)
200 caracteres por paquete			
l_m / **Nº paquetes**	t_{trans} (ms.)	t_{trans_ss} (ms.)	**Sobrecarga** (%)
0 / 1	8.690	17.190 (13.078)	99.67 (51.91)
15 / 1	12.215	20.715 (16.603)	69.59 (35.92)
95 / 1	31.015	39.515 (35.403)	27.41 (14.15)
195 / 1	54.515	63.015 (58.903)	15.59 (8.05)

2.- Bus VME.

1000 caracteres por paquete			
l_m / **Nº paquetes**	t_{trans} (ms.)	t_{trans_ss} (ms.)	**Sobrecarga** (%)
0 / 1	5.917	24.030 (17.862)	306.12 (201.88)
995 / 1	9.897	28.010 (21.842)	183.02 (120.69)

4995 / 5	37.461	98.350 (83.958)	162.54 (124.12)
9995 / 10	71.916	186.275 (161.603)	159.02 (124.71)
10000 caracteres por paquete			
l_m / **Nº paquetes**	t_{trans} (ms.)	t_{trans_ss} (ms.)	**Sobrecarga** (%)
0 / 1	5.917	24.030 (17.862)	306.12 (201.88)
995 / 1	9.897	28.010 (21.842)	183.02 (120.69)
4995 / 1	25.897	44.010 (37.842)	69.94 (46.13)
9995 / 1	45.897	64.010 (57.842)	39.46 (26.03)

A partir de los resultados obtenidos en la evaluación de la sobrecarga podemos realizar las siguientes observaciones:

- La sobrecarga disminuye con el aumento en el tamaño de los paquetes de comunicaciones. La elección del tamaño del paquete se efectuará en función de la sobrecarga y del bloqueo que supone cada paquete de comunicaciones en sí mismo. Así, como en el bus VME el tiempo de procesador consumido en la transmisión podemos suponer que coincide con el propio tiempo de la transmisión, es mejor utilizar paquetes grandes (reducen la sobrecarga y el incremento relativo del tiempo de transmisión es pequeño). Sin embargo, en las líneas serie el tiempo de transmisión es mucho mayor que el tiempo de procesador consumido en la transmisión, por lo que el uso de paquetes de gran tamaño puede dar lugar a grandes bloqueos sobre paquetes de mayor prioridad.

- El envío de paquetes incompletos contribuye al aumento de la sobrecarga.

- Una gran parte de la sobrecarga al usar el servidor esporádico se debe a la consulta del reloj. Con el uso de un reloj con tiempos de consulta pequeños se reduce sensiblemente esta sobrecarga (aproximadamente el 50% en las líneas serie, y el 30% en el bus VME).

Por último, podemos afirmar que la sobrecarga que se produce por el uso del servidor esporádico para comunicaciones tiene relativamente poca importancia comparada con los

aumentos que se consiguen en las utilizaciones de los recursos. Además, esta sobrecarga será tanto menos importante cuanto mayor sea la potencia de los recursos. Hay que tener en cuenta que la sobrecarga de las comunicaciones suele ser un porcentaje pequeño de la carga del sistema, por lo que la sobrecarga que produce el *SEC* seguirá siendo pequeña. En cambio, el aumento que se consigue en la planificabilidad del sistema al eliminar el *jitter* puede ser extremadamente grande, como vimos en el capítulo 6.

7.5. Ejemplos

Hasta el momento hemos definido, implementado, modelado y evaluado una serie de redes de comunicación capaces de soportar requerimientos de tiempo real estricto, aplicables por tanto a sistemas multiprocesadores y distribuidos de tiempo real estricto, y que disponen además de un mecanismo para la planificación de la transmisión de paquetes de comunicaciones en las mismas: el *servidor esporádico para comunicaciones*, que permite eliminar el *jitter* (activación retrasada), y conseguir así, un incremento considerable de las utilizaciones, tanto de las redes, como de los procesadores del sistema distribuido.

En este apartado detallaremos un conjunto de ejemplos del uso de las redes de comunicación implementadas, en las que vamos a poner de manifiesto tanto las peculiaridades propias de la utilización de dichas redes y del modelado de estas aplicaciones para el análisis de tiempo real, como las ventajas del mecanismo de planificación desarrollado.

Las aplicaciones de las redes de comunicación y de las colas de mensajes en las que se basan las elegimos con el propósito de resaltar los aspectos más importantes de la implementación realizada, en cuanto a su característica de tiempo real estricto. Para ello, consideramos los cuatro ejemplos siguientes en los que destacamos:

Sistema monoprocesador de tiempo real estricto. Una visión pormenorizada de las operaciones locales de envío y recepción de mensajes de una cola de mensajes, en un marco monoprocesador con requerimientos de tiempo real estricto.

Sistema multiprocesador de tiempo real estricto basado en una línea serie. Aplicación sencilla de tiempo real estricto con intercambio de mensajes entre tareas de dos procesadores haciendo uso de colas de mensajes globales, a través de una línea serie RS-485.

Sistema multiprocesador de tiempo real estricto basado en el bus VME. Igual que el ejemplo anterior, pero utilizando como red de comunicación el bus VME.

Aplicación práctica de un sistema multiprocesador de tiempo real estricto. Ejemplo de sistema multiprocesador sencillo (dos procesadores) en el que las respuestas a una serie de eventos con requerimientos de tiempo real estricto están compuestas por varias tareas distribuidas por los procesadores, y que se comunican mediante colas de mensajes globales, a través de una línea serie RS-485 y del bus VME. En este ejemplo veremos la necesidad del uso de prioridades y las ventajas que conlleva la utilización del servidor esporádico para comunicaciones.

Antes de comenzar con la descripción de los ejemplos, vamos a dar las pautas de configuración y utilización en general tanto de las colas de mensajes, como del servidor esporádico para comunicaciones. En líneas generales antes de que las tareas de usuario comiencen su ejecución, se deben realizar las siguientes actividades:

- Antes de la activación de las tareas de usuario hay que establecer el nudo actual (*configuracion.establece_nudo_actual(…)*).

- Antes de la activación de las tareas de usuario hay que declarar también la localización de las colas globales (*configuracion.configura_nudo(…)*).

- Antes de la activación de las tareas de usuario, se deben establecer los parámetros del servidor esporádico (si se está utilizando) al menos para las prioridades que se vayan a usar (*ss_comunicaciones_xxx.establece_parametros(…)*).

- Para poder sincronizar el instante de activación de la ejecución normal de las tareas de usuario (comienzo de la actividad periódica), hay que establecer la hora inicial y pasársela en el instante en el que se activen éstas.

- La primera operación que deben hacer las tareas de usuario cuando se activan es la conexión con las colas de mensajes que vayan a utilizar. Esta conexión la realizan mediante la operación *abre* (una tarea crea la cola, y las demás se conectan).

Finalmente, definimos los modelos para las tareas de usuario que vamos a utilizar en nuestros ejemplos, a partir del instante en el que han concluido su etapa de inicialización, es decir, se ha inicializado el sistema y las tareas ya se han conectado a las colas de mensajes que van a utilizar y pueden comenzar su actividad. Definimos estos modelos con independencia de que las operaciones sobre las colas de mensajes sean locales o remotas, tal como se muestra a continuación:

- *Productor*. Tarea de activación periódica que después de la ejecución de su trabajo, activa a otra tarea mediante el envío de un mensaje a una cola de mensajes. Se implementa como un bucle de la siguiente forma:

  loop

  Ejecuta trabajo;

  Envía mensaje;

  Espera hasta nueva activación;

  end loop;

- *Consumidor*. Tarea activada por otra tarea mediante la recepción de un mensaje (se queda esperando hasta que la otra tarea envía el mensaje y después ejecuta su trabajo). Se implementa como un bucle de la siguiente forma:

  loop

  Recibe mensaje;

  Ejecuta trabajo;

  end loop;

- *Productor-consumidor*. Tarea activada por otra tarea que después de la ejecución de su trabajo activa a una tercera. Se implementa como un bucle de la siguiente forma:

```
loop
        Recibe mensaje;
        Ejecuta trabajo;
        Envía mensaje;
end loop;
```

La ejecución de las tareas de usuario se lleva a cabo mediante la llamada a un procedimiento que consume tiempo de CPU, que consiste en un bucle que ejecuta una operación un número de veces proporcional al tiempo que se quiere consumir.

7.5.1. Sistema monoprocesador de tiempo real estricto

En este ejemplo daremos una visión global del uso de las operaciones de envío y recepción de mensajes de una cola de mensajes y también haremos un estudio detallado del modo local de estas operaciones.

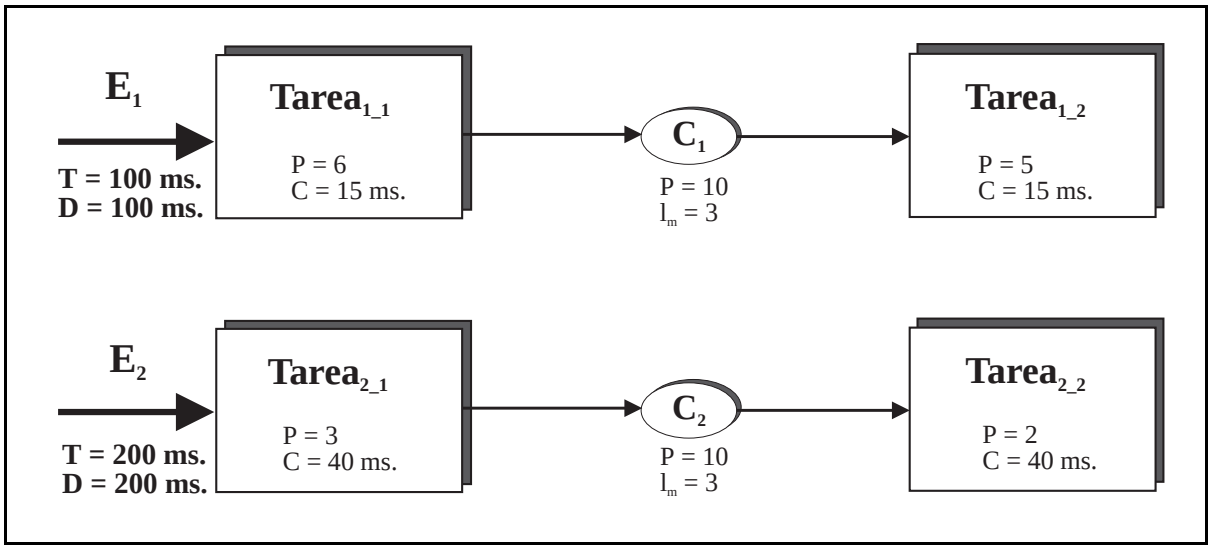

Figura 7-3. Ejemplo de sistema monoprocesador de tiempo real estricto.

Consideramos el sistema monoprocesador de la Figura 7-3, en el que tenemos dos eventos periódicos (de períodos T_i) sobre cuyas respuestas existen requerimientos de tiempo real estricto (plazos D_i). Las respuestas a estos eventos se componen de dos tareas que se

comunican a través de una cola de mensajes. Las tareas que responden directamente a los eventos pertenecen al tipo *productor* ($tarea_{1_1}$ y $tarea_{2_1}$), y las que responden a los mensajes enviados por éstas pertenecen al tipo *consumidor* ($tarea_{1_2}$ y $tarea_{2_2}$). Todas las tareas tienen un tiempo de ejecución de peor caso C_{i_j} (simulado por el procedimiento de consumo de tiempo de CPU), y ejecutan a una prioridad P_{i_j}. En cuanto a los mensajes que se utilizan, se caracterizan por su longitud (l_{mi}) y por su prioridad (P_i). La asignación de las prioridades se ha hecho acorde a la teoría RMA aplicable a estos casos [GON91A]. A efectos de cálculo de tiempos consideramos el rango creciente de prioridades 1..32 (utilizado por la implementación de las colas de mensajes), tanto para las tareas como para los mensajes.

Para realizar el análisis de tiempo real de este sistema, extraemos previamente su modelo a partir de las características propias de su configuración, y del modelado realizado para las operaciones sobre las colas de mensajes que se utilizan. Por lo tanto, vamos a considerar el modelado para el análisis de este sistema compuesto por las siguientes tareas: (--- indica que el dato no es relevante)

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
Tick	Alta	20	---	0.470	---
$Tarea_1$	Media	100	100	47.972	20
$Tarea_2$	Baja	200	200	98.242	0

donde:

- *Tick*. Es el tick utilizado por el planificador de Ada y se puede considerar como una tarea periódica (período 20 ms.) de alta prioridad cuyo tiempo de ejecución de peor caso fue evaluado en la página 7-22.

- $Tarea_1$. Es la tarea formada por la secuencia de respuesta al evento E_1, o sea, por las tareas $tarea_{1_1}$ y $tarea_{1_2}$, y por el envío y recepción del mensaje que dispara a la segunda. Su período y su plazo coinciden con los del evento E_1, y además el mayor bloqueo que puede sufrir es el correspondiente al período del tick. El tiempo de ejecución de peor caso asociado a esta tarea es el siguiente:

$$C_1 = C_{1_1} + C_{1_2} + t_{esp} + t_{envia_1} + t_{recibe_1} + 2\ t_{cc}$$

donde,

C_{1_1} y C_{1_2} Son los tiempos de ejecución de peor caso de $tarea_{1_1}$ (15 ms.) y $tarea_{1_2}$ (15 ms.).

t_{envia_1} Es el tiempo de envío del mensaje para $tarea_{1_1}$, calculado a partir del modelado que hemos realizado para una operación *envía* local con tarea en espera local y un mensaje de la longitud y prioridad dados. Este tiempo vale,

$$t_{envia_1} = 7.036\ ms.$$

con, $l_m = 3$ y $N_{busca_tarea} = 28$.

t_{recibe_1} Es el tiempo de recepción del mensaje para $tarea_{1_2}$, calculado a partir del modelado que hemos realizado para una operación *recibe* local sobre un mensaje de la longitud y prioridad dados. Este tiempo vale,

$$t_{recibe_1} = 8.923\ ms.$$

con, $l_m = 3$, $N_{busca_mensaje} = 23$ y $N_{intentos} = 1$.

t_{cc} Es el tiempo de cambio de contexto, estimado en 0.626 ms.

t_{esp} Es el tiempo consumido en la ejecución del *espera hasta próxima activación* por la $tarea_{1_1}$, estimado en 0.761 ms.

- $Tarea_2$. Es la tarea formada por la secuencia de respuesta al evento E_2, o sea, por las tareas $tarea_{2_1}$ y $tarea_{2_2}$, y por el envío y recepción del mensaje que dispara a la segunda. Su período y su plazo coinciden con los del evento E_1. El tiempo de ejecución de peor caso asociado a esta tarea es el siguiente:

$$C_2 = C_{2_1} + C_{2_2} + t_{esp} + t_{envia_2} + t_{recibe_2} + 2\ t_{cc}$$

donde,

C_{2_1} y C_{2_2} Son los tiempos de ejecución de peor caso de $tarea_{2_1}$ y $tarea_{2_2}$.

t_{envia_2} Es el tiempo de envío del mensaje para $tarea_{2_1}$, el cual se calcula de forma análoga al de $tarea_1$. Este tiempo vale,

$$t_{envia_2} = 7.306\ ms.$$

con, $l_m = 3$ y $N_{busca_tarea} = 31$.

t_{recibe_2} Es el tiempo de recepción del mensaje para $tarea_{2_2}$. Este tiempo es el mismo que para $tarea_1$,

$$t_{recibe_2} = 8.923\ ms.$$

con, $l_m = 3$, $N_{busca_mensaje} = 23$ y $N_{intentos} = 1$.

y,

t_{cc} y t_{esp} ($tarea_{2_1}$) son los mismos que para $tarea_1$.

De acuerdo con lo expuesto, realizamos el análisis de tiempo real y obtenemos los siguientes resultados para las respuestas de peor caso a los eventos del sistema:

$$r_1 = 69.852\ ms. \quad para\ E_1$$
$$r_2 = 198.886\ ms. \quad para\ E_2$$

con una utilización de la CPU del 97.12%.

Por lo tanto, a la vista de los resultados obtenidos del análisis teórico, la primera observación que podemos hacer es que se verifican los plazos. En esta misma línea, la Figura 7-4 muestra la evolución teórica de este sistema según los tiempos obtenidos en el modelado, y suponiendo que todas las tareas están preparadas para la ejecución en el instante inicial. En esta figura se detalla también la ejecución de las operaciones de envío y recepción de mensaje (con su paso por *control*), teniendo en cuenta por un lado los tiempos utilizados en el modelado del ejemplo, y por otro, el hecho de que la operación *recibe* se realiza en dos tramos: el intento inicial de recepción con resultado de suspensión (tiempo de peor caso igual a 2.757 ms.), y la recepción después de que la tarea es despertada (tiempo de peor caso igual a 6.166 ms.).

En la Figura 7-5 se representan los tiempos obtenidos experimentalmente para la ejecución del sistema de este ejemplo. En la figura se detallan los tiempos de ejecución de las operaciones *envía* y *recibe* sobre las colas de mensajes, junto con los tiempos de ejecución de las tareas, y los instantes en los que se produce el tick. Como se puede observar se verifican los plazos tal como habíamos previsto mediante el análisis realizado. Además, los tiempos de peor caso obtenidos en el análisis de respuesta a los eventos son realmente peores que los medidos experimentalmente. También comprobamos que los tiempos medidos para las operaciones sobre las colas de mensajes se ajustan a los calculados a partir del modelado de las mismas.

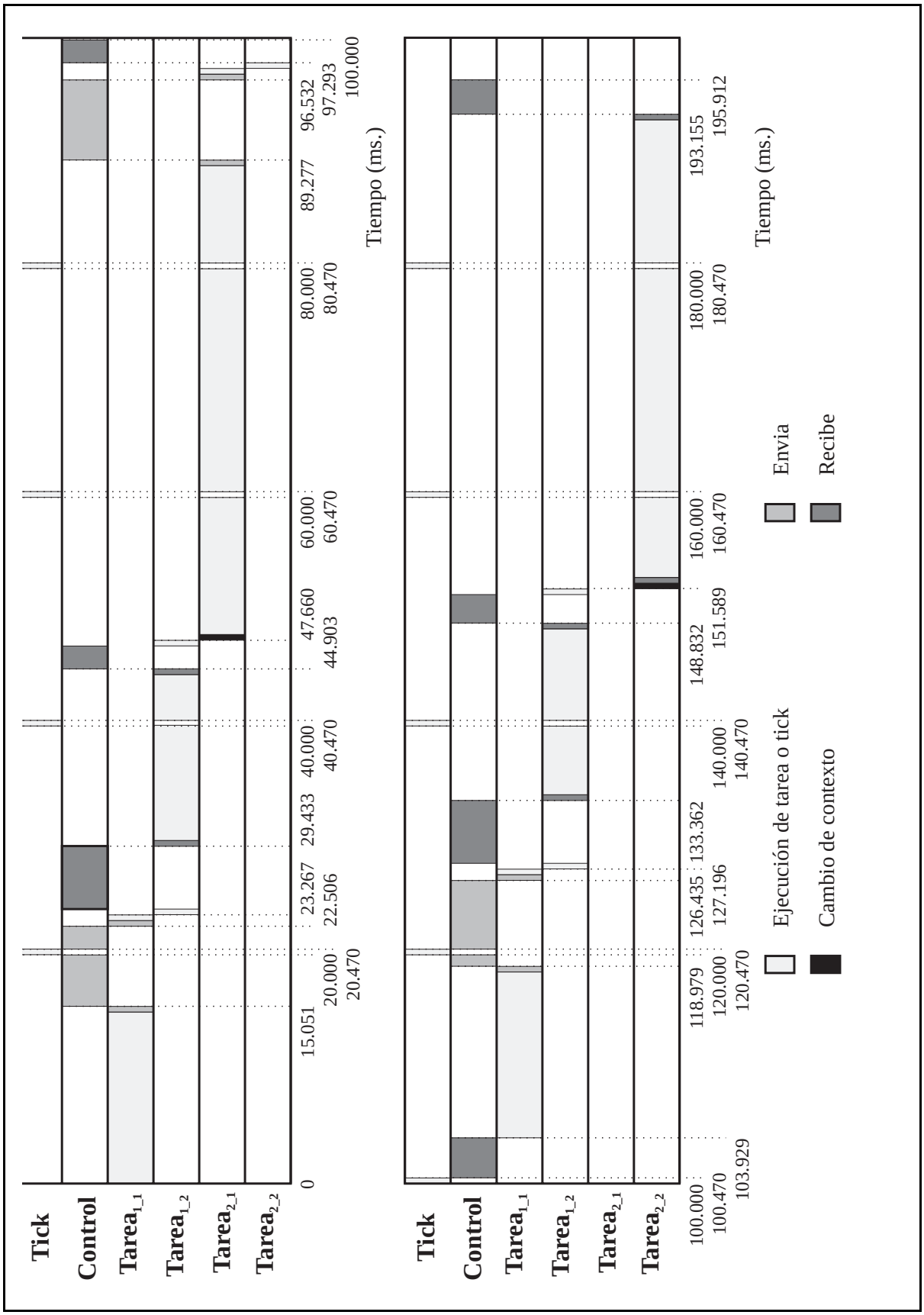

Figura 7-4. Ejecución teórica del sistema monoprocesador.

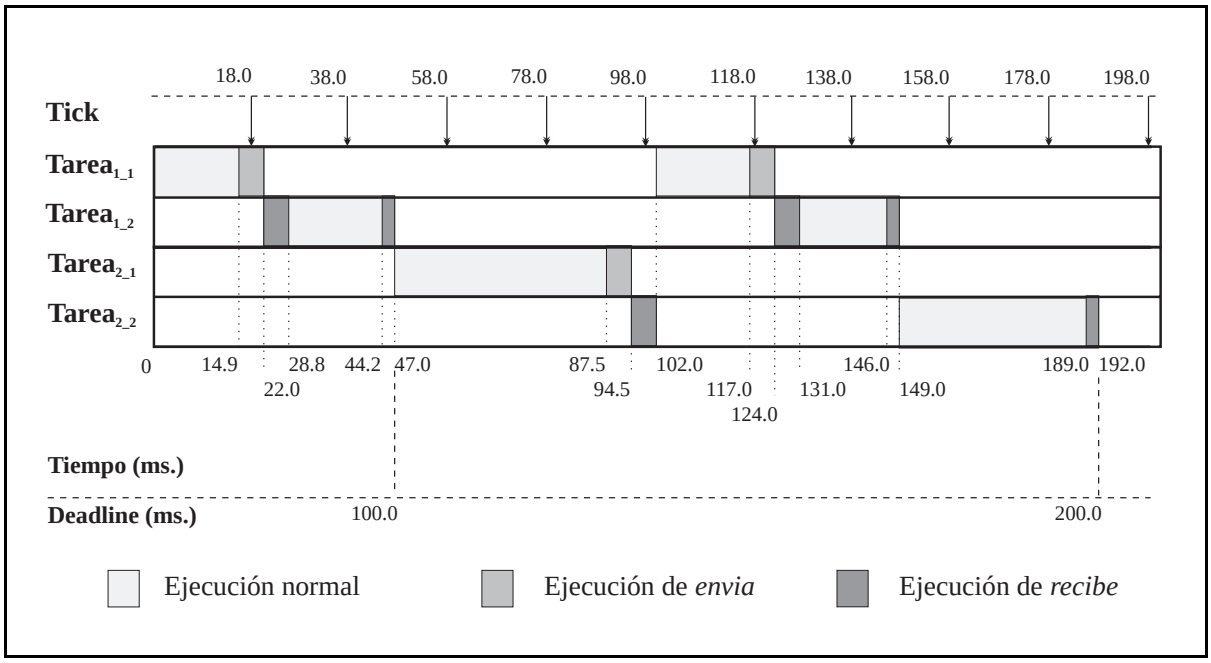

Figura 7-5. Ejecución real del sistema monoprocesador.

7.5.2. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones punto a punto

En el siguiente ejemplo, vemos una aplicación sencilla de tiempo real estricto con intercambio de mensajes entre tareas de dos procesadores a través de una línea serie RS-485, haciendo uso de colas de mensajes globales. Realizaremos además la asignación de prioridades a las tareas y a los mensajes con la herramienta que hemos desarrollado (capítulos 3 y 4 de esta Tesis), utilizando los tiempos de ejecución de peor caso calculados a partir del modelado de las colas de mensajes.

Consideramos el sistema de la Figura 7-6, constituido por dos procesadores y una red de conexión entre ambos. Tenemos dos eventos periódicos (de períodos T_i) sobre cuyas respuestas existen requerimientos de tiempo real estricto (plazos D_i). Estas respuestas están distribuidas entre los dos procesadores, y ambas se componen de dos tareas (una en cada procesador), que se comunican a través de una cola de mensajes (localizada en el procesador 2) que utiliza como red de comunicaciones una línea serie RS-485 a 1200 baudios. Al igual que en el ejemplo anterior, las tareas que responden directamente a los eventos pertenecen al tipo *productor* ($tarea_{1_1}$ y $tarea_{2_1}$), y las que responden a los mensajes enviados por éstas pertenecen al tipo *consumidor* ($tarea_{1_2}$ y $tarea_{2_2}$). Todas las tareas tienen un tiempo de

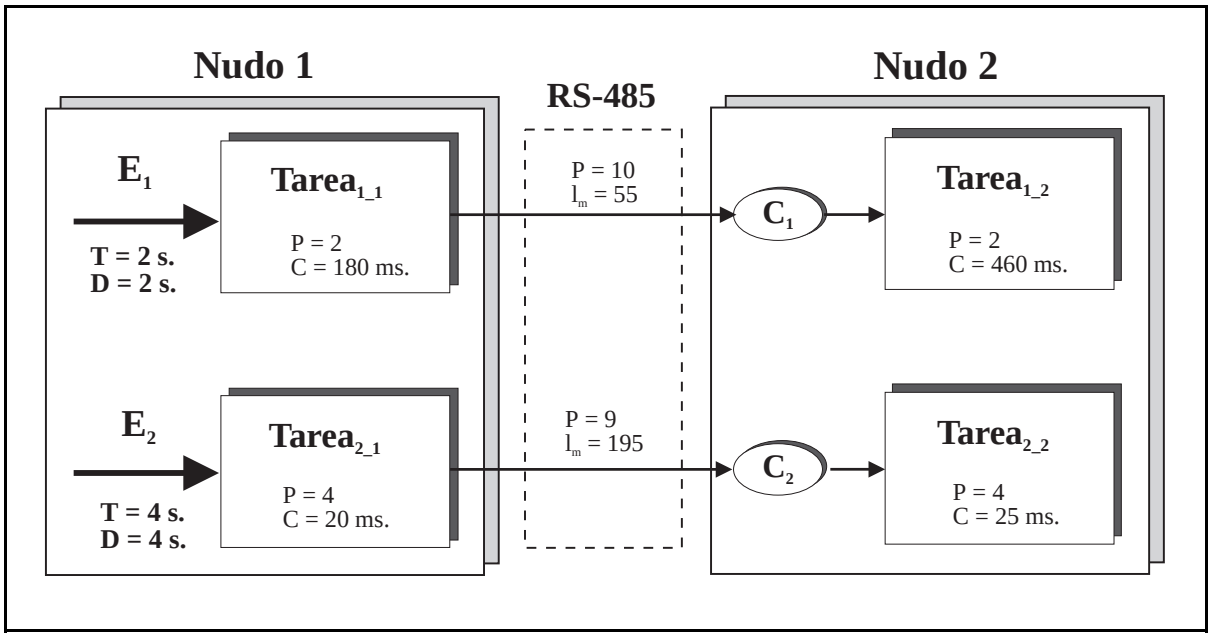

Figura 7-6. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones punto a punto.

ejecución de peor caso C_{i_j} (simulado por el procedimiento de consumo de tiempo de CPU), y ejecutan a una prioridad P_{i_j}. En cuanto a los mensajes que se utilizan, se caracterizan por su longitud (l_{mi}) y por su prioridad (P_i).

Las operaciones que se realizan sobre las colas de mensajes, al estar éstas localizadas en el procesador 2, son las siguientes:

- *Envía* remoto con la suposición de peor caso de que siempre se encolan los paquetes sin enviar el primero, es decir, la transmisión de paquetes nunca está parada. Esta suposición la podemos hacer debido al elevado porcentaje de uso de la red. Su ejecución en el nudo remoto siempre se efectúa con elementos en la cola de espera local.

- *Recibe* local con espera, o sea, la tarea que ejecuta esta operación se suspende y espera en la cola de espera local a que se ejecute el *envía* correspondiente.

Para la transmisión de los mensajes por la red vamos a utilizar una longitud de paquete l_p = 20, lo que significa que el número de caracteres transmitidos por cada paquete teniendo en cuenta el modelado (ver página 7-18) va a ser:

$$l_{p_trans} = l_p + l_{cab} + l_{STX} + l_{ETX} = 20 + 5 + 1 + 1 = 27 \ caracteres$$

La asignación de las prioridades a las tareas y a los mensajes se ha hecho utilizando la herramienta de optimización desarrollada en los capítulos 3 y 4 de esta Tesis, asignando prioridades forzadas a las tareas que surgen del modelado de las comunicaciones (las más altas), y dejando libre la asignación de prioridades a las tareas de usuario y a los mensajes con los que se comunican estas tareas. Como para el ejemplo anterior, también a efectos de cálculo de tiempos vamos a considerar el rango de prioridades 1..32 (utilizado por la implementación de las colas de mensajes), tanto para las tareas como para los mensajes.

Para realizar la asignación de prioridades, extraemos previamente el modelo del sistema a partir de las características propias de su configuración, y del modelado realizado para las operaciones sobre las colas de mensajes que se utilizan, sin tener en cuenta términos de bloqueo. Después, con la asignación de prioridades obtenida de la que solamente nos interesa el orden (las prioridades asignadas se muestran en la Figura 7-6), se calculan los términos de bloqueo y se efectúa el análisis. Por lo tanto, el modelo sobre el que aplicamos el análisis queda de la forma siguiente para cada recurso del sistema:
(--- indica que el dato no es relevante)

- **Nudo 1**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
Tick	Muy alta	20	---	0.470	---
T_car	Alta	8.333	---	0.231	---
T_paq_1	Alta	225	---	6.330	1.316
$Tarea_{2_1}$	Media	4000	---	45.906	20
$Tarea_{1_1}$	Baja	2000	---	190.786	0

- **Nudo 2**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
Tick	Muy alta	20	---	0.470	---
T_car	Alta	8.333	---	0.231	---

T_paq_2	Alta	225	---	4.914	---
T_men_1	Media	2000	---	24.365	---
T_men_2	Media	4000	---	61.846	---
$Tarea_{2_2}$	Baja	4000	4000	75.957	20
$Tarea_{1_2}$	Muy baja	2000	2000	481.180	0

- **Red**.

Mensaje	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
M_1	Alta	2000	---	697.938	232.646
M_2	Baja	4000	---	2326.460	0

donde:

- *Tick*. Es el tick utilizado por el planificador de Ada y se considera como en el ejemplo anterior.

- *T_car*. Es la tarea con la que modelamos la transmisión y recepción de caracteres. Debido a la alta utilización de la red (93.06%), podemos considerar que tanto la transmisión como la recepción de caracteres son actividades continuas y por lo tanto, se pueden modelar mediante una tarea periódica, cuyo período viene determinado por el baud-rate utilizado de la forma siguiente:

$$T = \frac{10}{baud-rate}$$

  El tiempo de ejecución de peor caso de esta tarea es el que emplea el procedimiento de atención de interrupción para transmitir o recibir un caracter (ver páginas 7-37 y 7-38).

- T_paq_1. Es la tarea que modela la actividad que se realiza para el inicio de la transmisión de cada paquete de comunicaciones, que de igual manera que para la transmisión de cada caracter, podemos considerar como una actividad periódica de período igual al tiempo de transmisión del paquete por la red

(dado por la ecuación de la página 7-20), y con tiempo de ejecución de peor caso calculado como indica el modelado de las comunicaciones. Esta tarea solamente existe en el procesador 1, ya que es el único que realiza actividades de transmisión. Por otra parte, en esta tarea nos interesa su tiempo de bloqueo además de su tiempo de ejecución de peor caso, ya que ambos deberán ser tenidos en cuenta en el cálculo del tiempo de transmisión de peor caso de cada paquete por la red.

- T_paq_2. Es la tarea que modela la actividad que se realiza cuando ha sido recibido un paquete completo. De igual manera que para el inicio de la transmisión de un paquete de comunicaciones, podemos considerarla como una actividad periódica de período igual al tiempo de transmisión del paquete por la red, y tiempo de ejecución de peor caso dado por el modelado de las comunicaciones. Esta tarea sólo existe en el procesador 2 y el posible bloqueo que pueda sufrir no es relevante para el análisis de la red.

- T_men_1 y T_men_2. Son las tareas que modelan en el procesador 2 las ejecuciones de las operaciones *envía* remotas (acción *ordena_rec* para operaciones remotas en el nudo remoto). Heredan los períodos de los eventos a cuya respuesta pertenecen. Sus tiempos de ejecución se calculan teniendo en cuenta que hay tareas en la cola de espera local en las prioridades 2 y 4 respectivamente para T_men_1 y T_men_2 (N_{busca_tarea} igual a 31 y 29).

- $Tarea_{1_1}$. Es la tarea de usuario a cuyo tiempo de ejecución hemos añadido el correspondiente a la ejecución de la operación *envía* remota en el nudo local (t_{envia} = 9.534 ms.), y a dos cambios de contexto (principio y final de la ejecución, t_{cc} = 0.626). El tiempo de ejecución de la *espera hasta próxima activación* en este caso lo hemos tenido en cuenta al programar el consumo de tiempo.

- $Tarea_{1_2}$. Es la tarea de usuario a cuyo tiempo de ejecución hemos añadido el correspondiente a la ejecución de la operación *recibe* remota en el nudo local (t_{recibe} = 19.928 ms.), y a dos cambios de contexto. El tiempo de ejecución de

la *espera hasta próxima activación*, como para el caso anterior, también lo hemos tenido en cuenta al programar el consumo de tiempo.

- $Tarea_{2_1}$ y $tarea_{2_2}$. Son análogas a las dos anteriores considerando t_{envia} = 24.654 ms. para $tarea_{2_1}$, y t_{recibe} = 49.705 ms. para $tarea_{2_2}$. Los tiempos de bloqueo que sufren estas tareas son los correspondientes al período del tick, porque ningún acceso a recursos compartidos de las tareas de menor prioridad supera este tiempo.

- M_1. Es el mensaje de comunicaciones generado a partir de la operación *envía* realizada por $tarea_{1_1}$, con un período igual al del evento E_1 a cuya respuesta pertenece. Si aplicamos la fórmula de la página 7-20 obtenemos el número de paquetes de comunicaciones en los que se divide este mensaje:

$$N_{paquetes} = \left\lceil \frac{5 + 55}{20} \right\rceil = 3$$

El tiempo de transmisión de peor caso para un paquete de comunicaciones es la suma del tiempo de transmisión de los caracteres que lo componen (dado por la formula de la página 7-20) y del tiempo de peor caso de preparación del paquete para la transmisión, que se corresponde con el tiempo de ejecución de T_paq_1 más su tiempo de bloqueo, es decir:

$$t_p = t_{paquete} + C_{T_paq_1} + B_{T_paq_1} = 225.0 + 6.33 + 1.316 = 232.646 \ ms.$$

El bloqueo que puede sufrir este mensaje debido al de menor prioridad es el correspondiente a la transmisión de un paquete (232.646 ms.).

- M_2. Es el mensaje de comunicaciones generado a partir de la operación *envía* realizada por $tarea_{2_1}$, con un período igual al del evento E_2 a cuya respuesta pertenece. De la misma forma que para el mensaje anterior, si aplicamos la fórmula de la página 7-20 obtenemos el número de paquetes de comunicaciones en los que se divide este mensaje:

$$N_{paquetes} = \left\lceil \frac{5 + 195}{20} \right\rceil = 10$$

El tiempo de transmisión de peor caso para un paquete de comunicaciones es el calculado en el mensaje anterior.

De acuerdo con los parámetros temporales calculados y las prioridades asignadas, realizamos el análisis de tiempo real y obtenemos los siguientes resultados para las respuestas de peor caso a los eventos que se producen en el sistema:

$$r_1 = 1972.913\ ms. \quad para\ E_1$$
$$r_2 = 3996.088\ ms. \quad para\ E_2$$

con las siguientes utilizaciones para los recursos:

$$U_{procesador_1} = 18.62\%$$
$$U_{procesador_2} = 36.03\%$$
$$U_{red} = 93.06\%$$

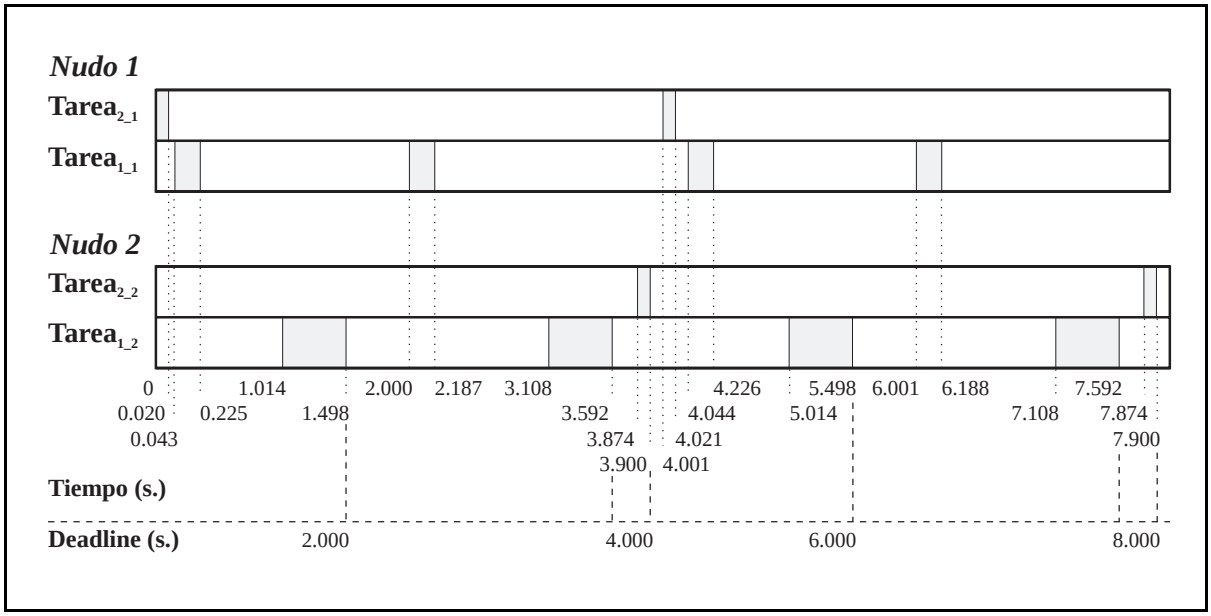

Figura 7-7. Ejecución del sistema multiprocesador basado en una red punto a punto.

Por consiguiente, el sistema de este ejemplo cumple las restricciones temporales que se le han impuesto, según se desprende de los resultados del análisis realizado. En la Figura 7-7 podemos ver el diagrama temporal correspondiente a la ejecución real de las tareas de este sistema. Se observa que ciertamente se verifican los plazos impuestos a las respuestas de los eventos, y que además los tiempos de estas respuestas son menores que los de peor caso obtenidos en el análisis.

Los tiempos que aparecen en la Figura 7-7 han sido medidos mediante el uso de dos emuladores del microprocesador MC68010 de Motorola en los que se basan las tarjetas utilizadas. Para efectuar las medidas se anotan los tiempos relativos a cada ejecución de las tareas a partir de un instante inicial. El instante inicial lo fijamos para cada procesador en el punto en el que comienza la ejecución de la primera tarea ($tarea_{2_1}$ para el procesador 1, y $tarea_{1_2}$ para el procesador 2). Entonces hay que tener en cuenta el desfase existente entre los tiempos obtenidos para los dos procesadores, es decir, el tiempo transcurrido desde que se activa $tarea_{2_1}$ hasta que lo hace $tarea_{1_2}$. Este desfase se calcula del modo siguiente:

$$t_{desfase} = I_{envio_M_1} + t_{envia_M_1} + t_{M_2/M_1} + C_{M_1} + C_{T_pak_2} + C_{T_men_1} + t_{recibe_M_1}$$

donde,

$I_{envio_M_1}$ Es el instante en el que $tarea_{1_1}$ comienza a enviar el mensaje M_1 (225 ms., ver Figura 7-7).

$t_{envia_M_1}$ Es el tiempo de ejecución de la operación *envía* en el nudo local realizada por $tarea_{1_1}$ (9.534 ms.).

t_{M_2/M_1} Es el bloqueo que sufre el mensaje M_1 por un paquete del mensaje M_2 que se esté enviando. Para calcular este bloqueo, vemos primero el tiempo que se lleva transmitiendo M_2 antes de que se envíe M_1, es decir, la diferencia entre los instantes de envío de los dos mensajes, más la diferencia entre los tiempos de las operaciones de envío remotas en el nudo local. Esto es lo siguiente:

$$t_{diferencia} = (225 - 20) + (9.534 - 24.654) = 189.88 \ ms.$$

En este tiempo, el número de caracteres que se han transmitido del mensaje M_2 es,

$$N_{caracteres} = \frac{t_{diferencia}}{T_{T_car}} = \frac{189.88}{8.333} = 22.786$$

es decir, aún quedan $\lceil 27\text{-}22.786 \rceil = 5$ caracteres por enviar que son los que influyen en el bloqueo, por lo que,

$$t_{M_2/M_1} = 5 * 8.333 = 41.665 \ ms.$$

C_{M_1} Es el tiempo de transmisión por la red de los 3 paquetes integrantes de M_1 (697.938 ms.).

$C_{T_paq_2}$ Es el tiempo empleado en la recepción del último paquete de M_1 (4.914 ms.), ya que los demás no contribuyen al desfase.

$C_{T_men_1}$ Es el tiempo de ejecución de la operación *envía* remota en el nudo remoto (24.365 ms.).

$t_{recibe_M_1}$ Es el tiempo de ejecución de la operación recibe por $tarea_{1_2}$ (19.928 ms.).

por lo que, $t_{desfase} = 1024\ ms$.

A la vista de los resultados obtenidos en este ejemplo, comprobamos que verdaderamente en los sistemas distribuidos de tiempo real estricto, la asignación de prioridades no solamente tiene que tener únicamente en cuenta los períodos y plazos de las acciones en cada recurso de una manera aislada, sino que también hay que considerar otras características (por ejemplo la cadena de respuesta al evento a la que pertenecen las acciones), tal y como se describe en el algoritmo de asignación de prioridades del capítulo 3 de esta Tesis.

En este ejemplo se ponen de manifiesto las bajas utilizaciones que se consiguen en los sistemas distribuidos de tiempo real estricto, cuando los plazos asociados a las respuestas de los eventos no son proporcionales a los períodos, y al número de recursos que intervienen en la ejecución de dichas respuestas. En este caso además, las bajas utilizaciones de los procesadores estás producidas por la concentración en la red de la mayor parte del tiempo de las respuestas a los eventos.

7.5.3. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones multipunto

En este ejemplo al igual que en el anterior, vemos una aplicación sencilla de tiempo real estricto con intercambio de mensajes entre tareas que se encuentran en dos procesadores, pero en este caso a través del bus VME. Realizaremos también la asignación de prioridades a las tareas y a los mensajes con la herramienta que hemos desarrollado (capítulos 3 y 4 de esta Tesis), utilizando los tiempos de ejecución de peor caso calculados a partir del modelado de las colas de mensajes.

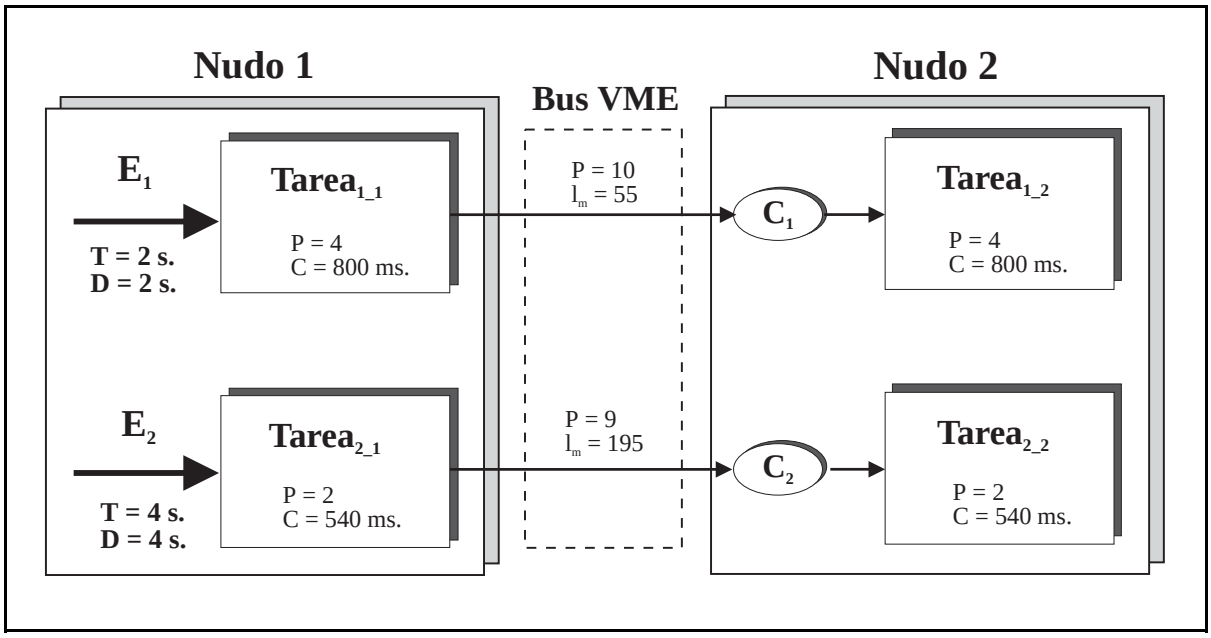

Figura 7-8. Sistema multiprocesador de tiempo real estricto basado en una red de comunicaciones multipunto.

La Figura 7-8 que representa un sistema con la misma estructura que el del ejemplo anterior, pero que utiliza como red de interconexión entre los procesadores un bus VME, y con valores diferentes en los parámetros de descripción del sistema. Las operaciones que se realizan sobre las colas de mensajes son las mismas que en el ejemplo anterior, pero las condiciones sobre la operación *envía* remota varían en el sentido de que siempre se transmite el primer paquete, debido a la baja utilización de la red (el tiempo empleado en la elaboración y transmisión de paquetes es despreciable frente al período con el que éstos se transmiten). Para la transmisión de los mensajes por la red vamos a utilizar una longitud de paquete l_p = 20.

La asignación de prioridades a las tareas y a los mensajes se ha hecho utilizando la herramienta de optimización desarrollada en los capítulos 3 y 4 de esta Tesis, asignando prioridades forzadas a las tareas que surgen del modelado de las comunicaciones (las más altas), y dejando libre la asignación de prioridades a las tareas de usuario y a los mensajes con los que se comunican estas tareas. Como para los ejemplos anteriores, a efectos de cálculo de tiempos vamos a considerar el rango de prioridades 1..32 (utilizado por la implementación de las colas de mensajes), tanto para las tareas como para los mensajes.

Para realizar la asignación de prioridades, extraemos previamente el modelo del sistema a partir de las características propias de su configuración, y del modelado realizado para las operaciones sobre las colas de mensajes que se utilizan, sin tener en cuenta términos de bloqueo. Después, con la asignación de prioridades obtenida de la que solamente nos interesa el orden (las prioridades asignadas se muestran en la Figura 7-8), se calculan los términos de bloqueo y se efectúa el análisis. Además, tenemos que añadir que no existe la red como tal, es decir, como un recurso en el que los mensajes emplean cierto tiempo para su transmisión, podemos considerar que este tiempo es nulo y únicamente se consume el tiempo de procesador necesario para la manipulación de cada paquete. Así pues, el modelo sobre el que aplicamos el análisis queda de la forma siguiente para cada recurso del sistema:
(--- indica que el dato no es relevante)

- **Nudo 1**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
$T_paq_{1_1}$	Muy alta	2000	---	13.496	---
$T_paq_{2_1}$	Muy alta	4000	---	60.957	---
Tick	Alta	20	---	0.470	---
$Tarea_{1_1}$	Media	2000	---	813.543	20
$Tarea_{2_1}$	Baja	4000	---	569.037	0

- **Nudo 2**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
$T_paq_{1_2}$	Muy alta	2000	---	17.865	---
$T_paq_{2_2}$	Muy alta	4000	---	59.550	---
Tick	Alta	20	---	0.470	---
T_men_1	Media	2000	---	45.906	---
T_men_2	Media	4000	---	190.786	---
$Tarea_{1_2}$	Baja	2000	2000	821.180	49.705
$Tarea_{2_2}$	Muy baja	4000	4000	590.957	0

donde:

- *Tick*. Es el tick utilizado por el planificador de Ada y se considera como en los ejemplos anteriores, aunque aquí no es la tarea de máxima prioridad.

- $T_paq_{1_1}$. Es la tarea que modela la actividad que se realiza para la transmisión de los paquetes de comunicaciones correspondientes al mensaje M_1. Suponemos que se produce la situación de peor caso en la que todos los paquetes se mandan seguidos (hay que tener en cuenta que el primer paquete se manda en la ejecución de la operación *envía*). Entonces, podemos considerar la transmisión como una actividad periódica de período igual al del evento a cuya respuesta pertenece, y con el tiempo de ejecución de peor caso siguiente:

  $$t_{T_paq_{1_1}} = (N_{paquetes} - 1)\ t_{trans_paq}$$

  donde, $N_{paquetes}$ = 3 (calculado como en el ejemplo anterior), y t_{trans_paq} = 6.748 ms. (calculado a partir del modelado de las comunicaciones).

- $T_paq_{2_1}$. Es la tarea que modela la actividad que se realiza para la transmisión de los paquetes de comunicaciones correspondientes al mensaje M_2. Haciendo las mismas suposiciones que para el mensaje M_1 (tarea anterior), podemos considerar la transmisión como una actividad periódica de período igual al del evento a cuya respuesta pertenece, y con el tiempo de ejecución de peor caso siguiente:

  $$t_{T_paq_{2_1}} = (N_{paquetes} - 1)\ t_{trans_paq}$$

  donde, $N_{paquetes}$ = 10, y t_{trans_paq} = 6.773 ms. (calculados ambos como en la tarea anterior).

- $T_paq_{1_2}$. Es la tarea que modela la actividad que se realiza en la recepción de los paquetes de comunicaciones correspondientes al mensaje M_1. Al igual que para la transmisión, suponemos que se produce la situación de peor caso en la que todos los paquetes llegan seguidos, y podemos considerar entonces la transmisión como una actividad periódica de período igual al del evento a cuya respuesta pertenece, y con el tiempo de ejecución de peor caso siguiente:

  $$t_{T_paq_{1_2}} = N_{paquetes}\ t_{rec_paq}$$

donde, $N_{paquetes}$ = 3, y t_{rec_paq} = 5.955 ms. (calculado a partir del modelado de las comunicaciones, teniendo en cuenta la transferencia del control al nudo que tiene que transmitir).

- $T_paq_{2_2}$. Es la tarea que modela la actividad que se realiza en la recepción de los paquetes de comunicaciones correspondientes al mensaje M_2. Al igual que en la tarea anterior, suponemos que se produce la situación de peor caso en la que todos los paquetes llegan seguidos, y podemos considerar entonces la transmisión como una actividad periódica de período igual al del evento a cuya respuesta pertenece, y con el tiempo de ejecución de peor caso siguiente:

$$t_{T_paq_{2_2}} = N_{paquetes}\ t_{rec_paq}$$

  donde, $N_{paquetes}$ = 10, y t_{rec_paq} = 5.955 ms. (calculado como en la tarea anterior a partir del modelado de las comunicaciones, teniendo en cuenta la transferencia del control al nudo que tiene que transmitir).

- T_men_1 y T_men_2. Como para el ejemplo anterior, son las tareas que modelan en el procesador 2 las ejecuciones de las operaciones *envía* remotas (acción *ordena_rec* para operaciones remotas en el nudo remoto). De igual manera heredan los períodos de los eventos a cuya respuesta pertenecen, y sus tiempos de ejecución se calculan teniendo en cuenta que hay tareas en la cola de espera local en las prioridades 4 y 2 respectivamente para T_men_1 y T_men_2 (N_{busca_tarea} igual a 29 y 31).

- $Tarea_{1_1}$. Es la tarea de usuario, a cuyo tiempo de ejecución hemos añadido el correspondiente a la ejecución de la operación *envía* remota en el nudo local con transmisión del primer paquete (t_{envia} = 13.291 ms.), y a dos cambios de contexto (principio y final de la ejecución, t_{cc} = 0.626). El bloqueo que puede sufrir esta tarea es el período del tick, ya que ningún acceso a recursos compartidos de la tarea de menor prioridad supera este tiempo. El tiempo de ejecución de la *espera hasta próxima activación*, como en el ejemplo anterior, lo hemos tenido en cuenta al programar el consumo de tiempo.

- $Tarea_{1_2}$. Es la tarea de usuario a cuyo tiempo de ejecución hemos añadido el correspondiente a la ejecución de la operación *recibe* remota en el nudo local (t_{recibe} = 19.928 ms.) y a dos cambios de contexto. El tiempo de bloqueo que sufre esta tarea es el tiempo de la ejecución de la operación *recibe* (*control*) por parte de $tarea_{2_1}$, que en este caso es mayor que el período del tick. El tiempo de ejecución de la *espera hasta próxima activación*, como para el caso anterior, también lo hemos tenido en cuenta al programar el consumo de tiempo.

- $Tarea_{2_1}$ y $tarea_{2_2}$. Son análogas a las dos anteriores considerando t_{envia} = 28.411 ms. para $tarea_{2_1}$, y t_{recibe} = 49.705 ms. para $tarea_{2_2}$.

De acuerdo con los parámetros temporales calculados y las prioridades asignadas realizamos el análisis de tiempo real y obtenemos los siguientes resultados para las respuestas de peor caso a los eventos que se producen en el sistema:

$$r_1 = 1989.507\ ms. \qquad para\ E_1$$
$$r_2 = 3990.026\ ms. \qquad para\ E_2$$

con las siguientes utilizaciones para los recursos:

$$U_{procesador_1} = 59.45\%$$
$$U_{procesador_2} = 63.32\%$$

Por consiguiente, el sistema de este ejemplo también cumple las restricciones temporales que se le han impuesto, según se desprende de los resultados del análisis realizado. En la Figura 7-9 podemos ver el diagrama temporal correspondiente a la ejecución de las tareas de este sistema. Se observa que se verifican los plazos impuestos a las respuestas de los eventos, y que además los tiempos de estas respuestas son menores que los de peor caso obtenidos en el análisis.

Los tiempos que aparecen en la Figura 7-9 han sido medidos, como para el ejemplo anterior, mediante el uso de dos emuladores del microprocesador MC68010 de Motorola. El método de medida también es el mismo, pero en este caso el instante inicial cambia en el sentido de que las tareas que comienzan la ejecución en cada procesador son: $tarea_{1_1}$ para el procesador 1, y $tarea_{1_2}$ para el procesador 2. De este modo, el desfase se calcula como se

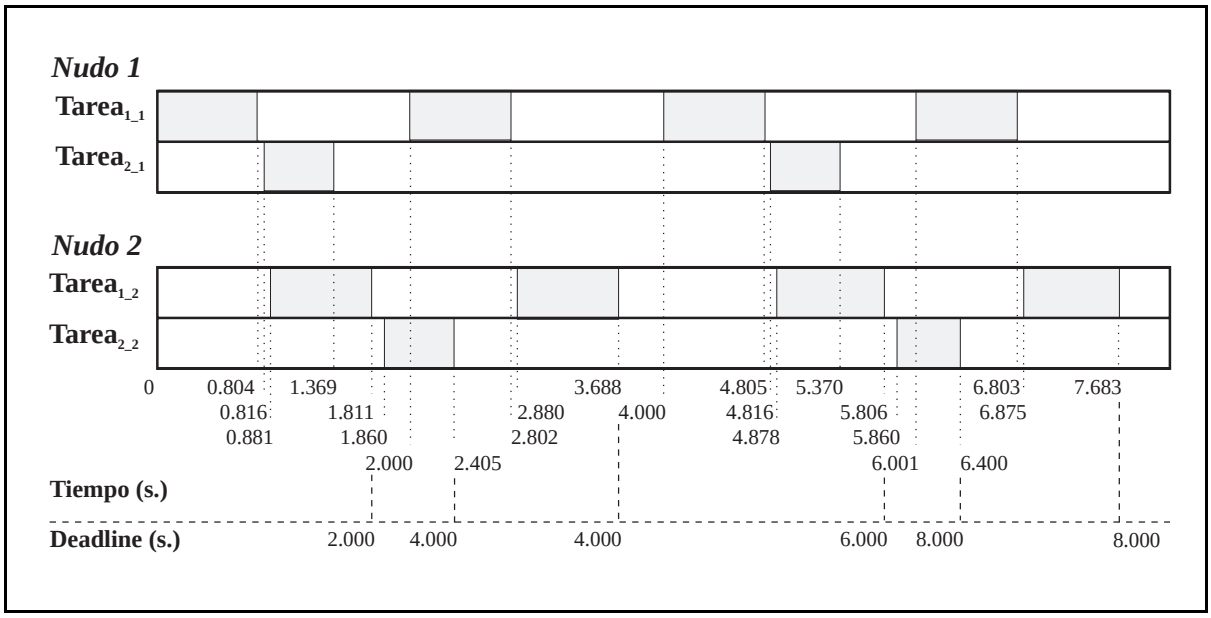

Figura 7-9. Ejecución del sistema multiprocesador basado en un red multipunto.

muestra a continuación:

$$t_{desfase} = I_{envio_M_1} + t_{envia_M_1} + C_{T_paq_{1_1}} + C_{Ult_T_paq_{1_2}} + C_{T_men_1} + t_{recibe_M_1}$$

donde,

$I_{envio_M_1}$ Es el instante en el que $tarea_{1_1}$ comienza a enviar el mensaje M_1 (804 ms., ver Figura 7-9).

$t_{envia_M_1}$ Es el tiempo de ejecución de la operación *envía* en el nudo local realizada por $tarea_{1_1}$ (13.291 ms.).

$C_{T_paq_{1_1}}$ Es el tiempo de transmisión de los dos paquetes restantes de M_1 (13.496 ms.).

$C_{Ult_T_paq_{1_2}}$ Es el tiempo empleado en la recepción del último paquete de M_1 (5.955 ms.), ya que los otros no contribuyen al desfase.

$C_{T_men_1}$ Es el tiempo de ejecución de la operación *envía* remota en el nudo remoto (24.185 ms.).

$t_{recibe_M_1}$ Es el tiempo de ejecución de la operación *recibe* por $tarea_{1_2}$ (19.928 ms.).

por lo que, $t_{desfase} = 881\ ms.$.

La asignación de prioridades obtenida para este ejemplo coincide con la que se realizaría según la teoría RMA, teniendo en cuenta únicamente los períodos de las acciones

para cada recurso separadamente. Al contrario que en el ejemplo anterior, en éste la red apenas tiene influencia (la transmisión se realiza en tiempo cero) y únicamente supone una pequeña sobrecarga en los procesadores.

7.5.4. Aplicación de un sistema multiprocesador distribuido de tiempo real estricto

En este apartado vamos a ver un ejemplo de sistema multiprocesador distribuido de tiempo real estricto con una estructura más cercana a lo que puede ser una aplicación real de este tipo de sistemas.

Consideramos el sistema representado en la Figura 7-10, constituido por dos procesadores, que se interconectan mediante dos redes, y un monitor. Tenemos seis eventos periódicos (de períodos T_i) sobre cuyas respuestas existen requerimientos de tiempo real estricto (plazos D_i). Estas respuestas están distribuidas por los dos procesadores como se indica en la figura, y todas se componen de tres tareas que se comunican a través de colas de mensajes utilizando las redes disponibles, que son: una línea serie RS-485 a 2400 baudios, y un bus VME. La conexión con el monitor se hace mediante una línea serie RS-232 (para el envío de caracteres no se utilizan interrupciones). Las tareas que responden directamente a los eventos pertenecen al tipo *productor* ($tarea_{i_1}$), las que responden a los mensajes enviados por éstas pertenecen al tipo *productor-consumidor* ($tarea_{i_2}$), que a su vez envían otros mensajes a las tareas del *consumidor* ($tarea_{i_3}$). Todas las tareas se caracterizan por su tiempo de ejecución de peor caso C_{i_j} y por su prioridad P_{i_j}. En cuanto a los mensajes que se utilizan, se caracterizan por su longitud (l_{mi_j}) y por su prioridad (P_{i_j}). Al igual que para los ejemplos anteriores, para la transmisión de los mensajes por la red vamos a utilizar una longitud de paquete l_p = 20.

Los sistemas con una estructura como la representada por la Figura 7-10 son muy comunes en aplicaciones de tiempo real, y en diversos campos como la robótica o la visión artificial. Esta estructura responde por ejemplo, a la de un sistema de monitorización y control de un robot, en el que las tareas $tarea_{i_1}$ realizan la lectura de ciertos sensores, transmiten los datos recogidos a las tareas $tarea_{i_2}$ de otro procesador para que realicen los cálculos que sean necesarios y finalmente, éstas devuelven los resultados para que las tareas $tarea_{i_3}$ los

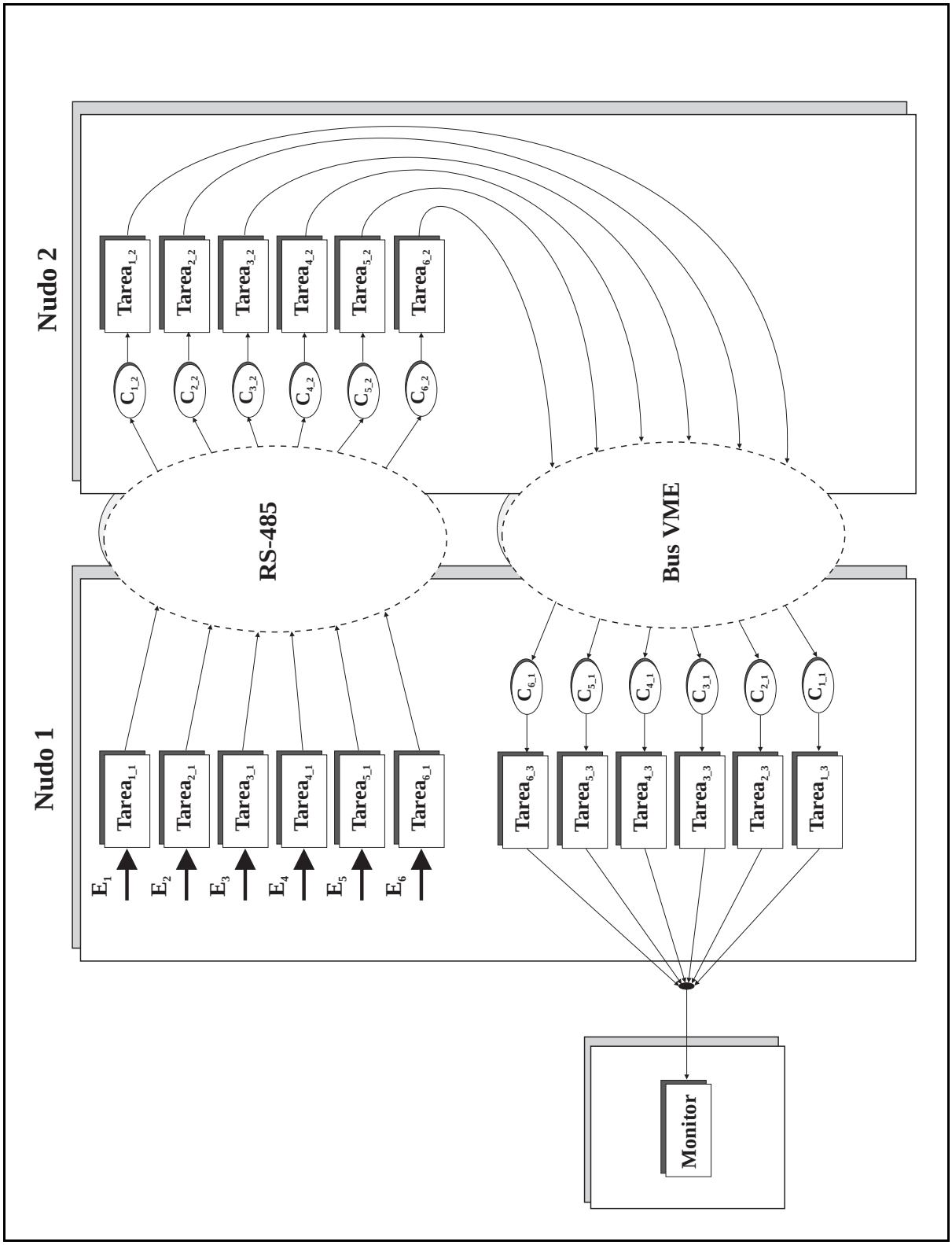

Figura 7-10. Sistema distribuido de tiempo real estricto monitorizado.

monitoricen y efectúen las acciones oportunas. También puede responder a un sistema de visión artificial de captación de imágenes ($tarea_{i_1}$), procesamiento de las mismas ($tarea_{1_2}$), y muestra de resultados con la realización de las acciones pertinentes ($tarea_{i_3}$).

La aplicación que desarrollamos pretende clarificar las ventajas o inconvenientes que puedan tener los mecanismos de comunicación implementados en las redes que utilizan los sistemas distribuidos de tiempo real estricto. Para ello, implementamos diversos sistemas basados en la estructura de la Figura 7-10 que se corresponden con distintas configuraciones de las longitudes y prioridades de los mensajes, de los períodos y plazos de los eventos, y de las prioridades y tiempos de ejecución de peor caso de las tareas (emulados por el procedimiento de consumo de tiempo de CPU). El fin que se persigue en este ejemplo es comprobar si en el sistema en ejecución se verifican o no los plazos impuestos a las respuestas a los eventos, para las distintas configuraciones.

En las distintas configuraciones con las que vamos a trabajar (todas ellas correspondientes a la estructura de la Figura 7-10), pretendemos verificar el cumplimiento de los plazos de una manera visual a través del monitor. Por otra parte, podemos establecer una analogía entre un sistema con la estructura referida y una carrera de atletismo, en la que cada evento es un atleta y su evolución sobre la pista de atletismo depende del cumplimiento del plazo de principio-a-fin que se le ha impuesto al evento. Básicamente, la idea que se quiere reflejar es que si los eventos van cumpliendo los plazos, entonces los atletas van avanzando sin retrasos por la pista; si por el contrario no los cumplen, se van retrasando.

Así pues, definimos el monitor como una pista de atletismo con seis atletas (Figura 7-11), en la que cada atleta es animado por uno de los eventos de nuestro sistema. La evolución de los atletas se rige por las siguientes reglas y definiciones:

- Definimos la longitud de la pista de atletismo en función del *paso* del atleta, haciendo corresponder éste con un pixel de la pantalla del monitor (la longitud de la pista es la resolución horizontal en pixels).

- El atleta puede avanzar solamente si la respuesta al evento que le anima cumple su plazo. Las tareas $tarea_{i_3}$ son las encargadas de verificar si se

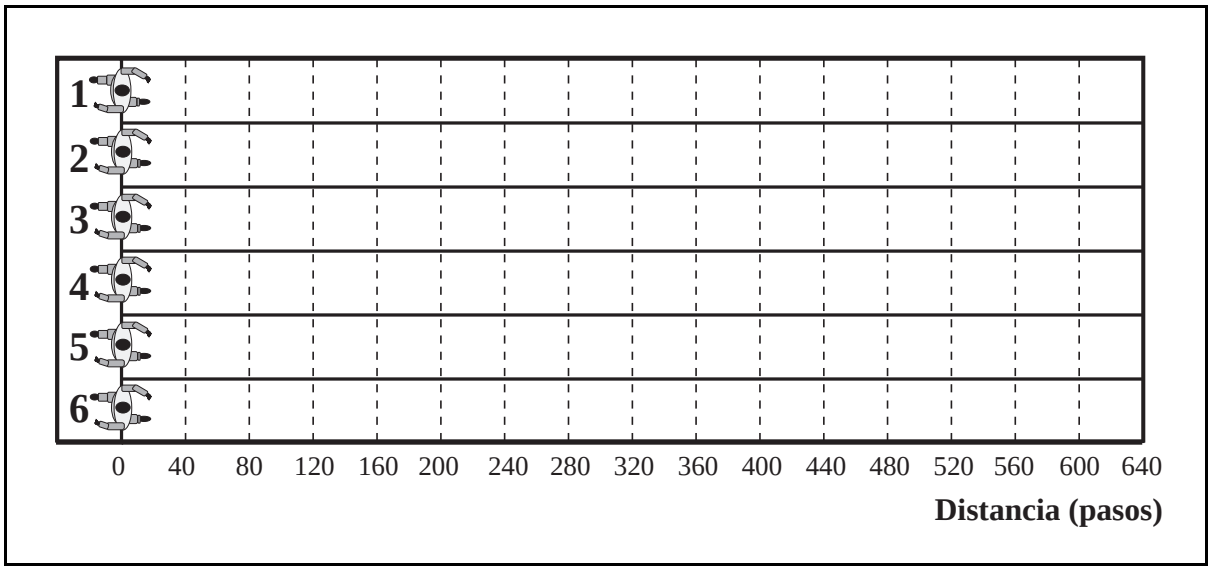

Figura 7-11. Monitor: pista de atletismo.

cumplen los plazos, en cuyo caso envían al monitor el correspondiente caracter de control del atleta.

- El número de pasos que el atleta puede dar por cada evento es proporcional al período del evento.

Con esta definición del monitor podemos comprobar de una manera gráfica si se verifican los plazos (si un evento no cumple su plazos el atleta correspondiente se retrasará con respecto a los que sí los cumplen). Para determinar si un atleta se retrasa, hacemos que en el instante inicial todos los eventos se produzcan a la vez, y tomamos como puntos de referencia aquellos en los que se sincronizan todos los atletas, es decir, los puntos en los que los eventos se vuelven a producir a la vez (múltiplos del mínimo común múltiplo de los períodos de los eventos).

La primera configuración sobre la que vamos a realizar las pruebas y con la que vamos a comprobar la influencia de la asignación de prioridades, es la que se muestra a continuación:

- **Eventos.**

Evento	**T** (s.)	**D** (s.)	**Nº Pasos**
E_1	1.0	1.2	10
E_2	2.0	3.0	20
E_3	3.0	9.0	30
E_4	4.0	12.0	40
E_5	5.0	15.0	50
E_6	6.0	18.0	60

- **Nudo 1.**

Tarea	**P**	**C** (s.)
$Tarea_{1_3}$	20	0.05
$Tarea_{1_1}$	19	0.1
$Tarea_{2_3}$	18	0.05
$Tarea_{2_1}$	17	0.1
$Tarea_{3_3}$	16	0.05
$Tarea_{3_1}$	15	0.1
$Tarea_{4_3}$	14	0.05
$Tarea_{4_1}$	13	0.1
$Tarea_{5_3}$	12	0.05
$Tarea_{5_1}$	11	0.1
$Tarea_{6_3}$	10	0.05
$Tarea_{6_1}$	9	0.1

- **Nudo 2.**

Tarea	**P**	**C** (s.)
$Tarea_{1_2}$	19	0.1
$Tarea_{2_2}$	18	0.1
$Tarea_{3_2}$	17	0.1

$Tarea_{4_2}$	16	0.1
$Tarea_{5_2}$	15	0.1
$Tarea_{6_2}$	14	0.1

- **Línea serie RS-485.**

Mensaje	**P / No P**	$\mathbf{l_m}$ **/** $\mathbf{N_{paquetes}}$
M_{1_1}	31 / 31	15 / 1
M_{2_1}	30 / 31	15 / 1
M_{3_1}	29 / 31	55 / 3
M_{4_1}	28 / 31	55 / 3
M_{5_1}	27 / 31	195 / 10
M_{6_1}	26 / 31	195 / 10

- **Bus VME.**

Mensaje	**P / No P**	$\mathbf{l_m}$ **/** $\mathbf{N_{paquetes}}$
M_{1_2}	31 / 31	5 / 1
M_{2_2}	30 / 31	5 / 1
M_{3_2}	29 / 31	5 / 1
M_{4_2}	28 / 31	5 / 1
M_{5_2}	27 / 31	5 / 1
M_{6_2}	26 / 31	5 / 1

En esta configuración de eventos, tareas y mensajes, se describen dos situaciones para estos últimos: una en la que se les asigna a todos la misma prioridad, y otra en la que se les asignan prioridades distintas. Cuando todos llevan la misma prioridad la transmisión de los paquetes de comunicaciones se realiza siguiendo una política FIFO. Los resultados obtenidos de la ejecución del sistema con esta configuración en sus dos vertientes se muestran en la Figura 7-12.

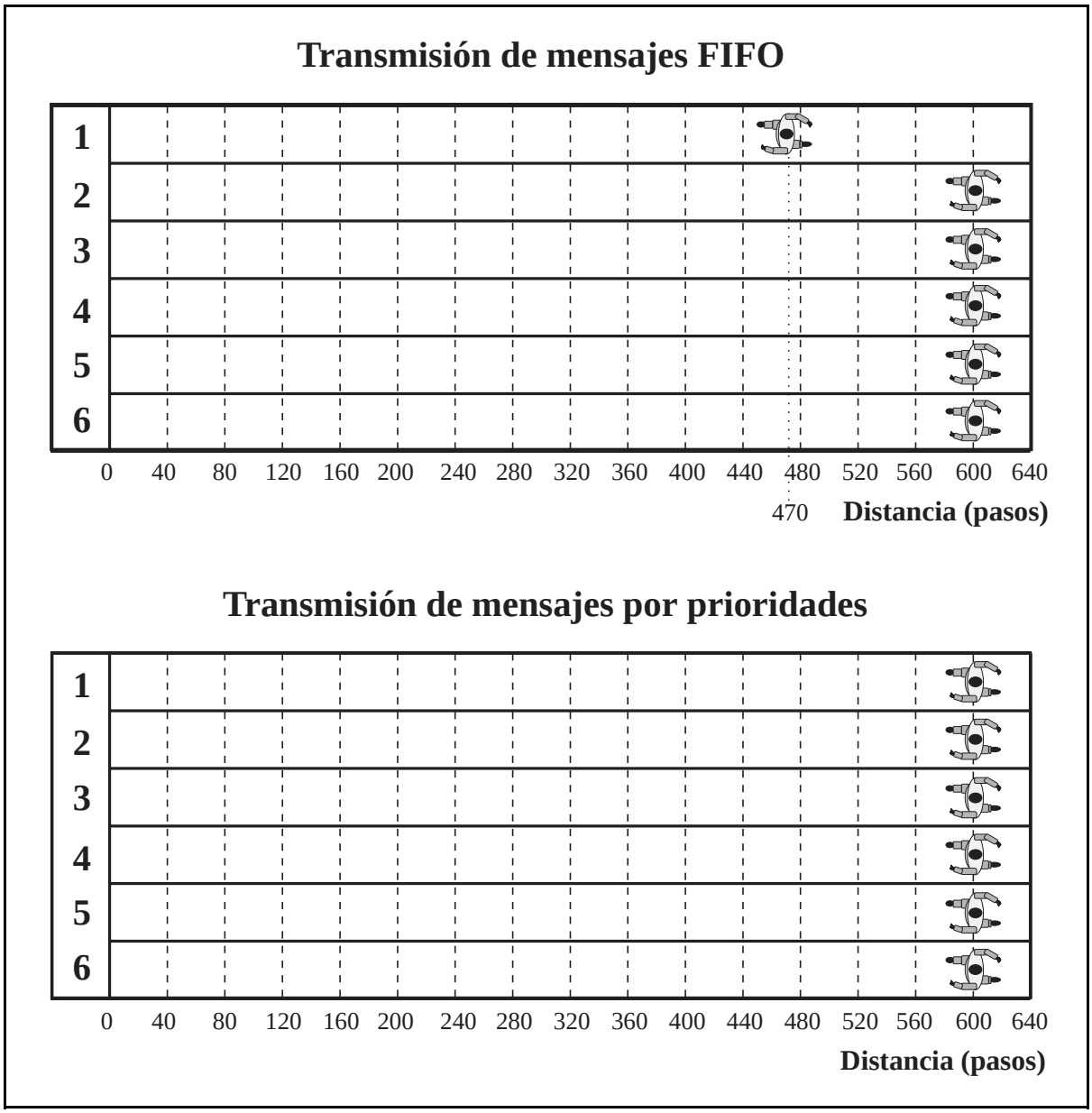

Figura 7-12. Resultados comparativos para la ejecución del sistema con transmisión de mensajes FIFO o por prioridades.

Como podemos comprobar, en el caso de que la transmisión de mensajes se lleve a cabo siguiendo una política FIFO, el atleta *1* solamente da 470 pasos cuando debería haber llegado como los demás al punto de sincronización (estimado en 600 pasos a partir de los períodos de los eventos y de los pasos recorridos por evento producido). Este retraso se produce en las redes de comunicación, y su explicación se encuentra en que los mensajes pertenecientes a la respuesta al evento E_1 deben esperar a cualquier otro mensaje de los otros eventos que se haya enviado primero.

Sin embargo, cuando se utilizan prioridades en la transmisión de mensajes por las redes de comunicación, ninguna respuesta a los eventos pierde su plazo y todos los atletas alcanzan el punto de sincronización en el instante deseado.

Estos resultados se pueden corroborar si realizamos el análisis de tiempo real y calculamos el tiempo de respuesta al evento E_1. Siguiendo el modelado de las colas de mensajes que hemos realizado en este capítulo, y de una forma análoga a los dos ejemplos anteriores, podemos obtener el modelado para el sistema de este ejemplo y la configuración con la que hemos realizado estas pruebas. Este modelado especificado para cada recurso es el que se muestra a continuación:
(--- indica que el dato no es relevante)

- **Nudo 1**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
$T_paq_{1_2}$	31	1000	---	5.955	---
$T_paq_{2_2}$	31	2000	---	5.955	---
$T_paq_{3_2}$	31	3000	---	5.955	---
$T_paq_{4_2}$	31	4000	---	5.955	---
$T_paq_{5_2}$	31	5000	---	5.955	---
$T_paq_{6_2}$	31	6000	---	5.955	---
Tick	30	20	---	0.470	---
T_car	29	4.166	---	0.231	---
T_paq1_{485}	29	112.5	---	5.927	1.316
$T_men_{1_2}$	23	1000	---	10.288	---
$T_men_{2_2}$	23	2000	---	10.288	---
$T_men_{3_2}$	23	3000	---	10.288	---
$T_men_{4_2}$	23	4000	---	10.288	---
$T_men_{5_2}$	23	5000	---	10.288	---
$T_men_{6_2}$	23	6000	---	10.288	---
$Tarea_{1_3}$	20	1000	1200	63.137	20
$Tarea_{1_1}$	19	1000	---	108.466	20
$Tarea_{2_3}$	18	2000	3000	63.137	---

$Tarea_{2_1}$	17	2000	---	108.466	---
$Tarea_{3_3}$	16	3000	9000	63.137	---
$Tarea_{3_1}$	15	3000	---	112.786	---
$Tarea_{4_3}$	14	4000	12000	63.137	---
$Tarea_{4_1}$	13	4000	---	112.786	---
$Tarea_{5_3}$	12	5000	15000	63.137	---
$Tarea_{5_1}$	11	5000	---	127.906	---
$Tarea_{6_3}$	10	6000	18000	63.137	---
$Tarea_{6_1}$	9	6000	---	127.906	---

- **Nudo 2.**

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
Tick	30	20	---	0.470	---
T_car	29	4.166	---	0.231	---
T_paq2_{485}	29	112.5	---	4.914	---
$T_men_{1_1}$	23	1000	---	12.978	---
$T_men_{2_1}$	23	2000	---	12.978	---
$T_men_{3_1}$	23	3000	---	23.738	---
$T_men_{4_1}$	23	4000	---	23.738	---
$T_men_{5_1}$	23	5000	---	61.398	---
$T_men_{6_1}$	23	6000	---	61.398	---
$Tarea_{1_2}$	20	1000	---	118.569	48.165
$Tarea_{2_2}$	19	2000	---	118.569	---
$Tarea_{3_2}$	18	3000	---	127.049	---
$Tarea_{4_2}$	17	4000	---	127.049	---
$Tarea_{5_2}$	16	5000	---	156.729	---
$Tarea_{6_2}$	15	6000	---	156.729	---

- **Línea serie RS-485.**

Mensaje	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
M_1	31	1000	---	119.743	119.743
M_2	30	2000	---	119.743	---
M_3	29	3000	---	359.229	---
M_4	28	4000	---	359.229	---
M_5	27	5000	---	1197.430	---
M_6	26	6000	---	1197.430	---

- **Bus VME.**

  No se modela como una red al considerar nulo el tiempo de transmisión de mensaje.

donde:

- *Tick*. Es el tick utilizado por el planificador de Ada y se considera como en los ejemplos anteriores (tampoco aquí es la tarea de máxima prioridad).

- $T_paq_{i_j}$. Es la tarea que modela la actividad que se realiza para la recepción de paquetes a través del bus VME correspondientes al mensaje M_{i_j}. La transmisión se considera que la realiza la operación *envía* al encolar el paquete de comunicaciones (transmite el primer y único paquete).

- *T_car*. Es la tarea con la que modelamos la transmisión y recepción de caracteres. Debido a la alta utilización de la red (82.82%), podemos considerar que tanto la transmisión, como la recepción de caracteres son actividades continuas y por lo tanto, se pueden modelar mediante una tarea periódica.

- T_paq1_{485}. Es la tarea que modela la actividad que se realiza para el inicio de la transmisión de cada paquete de comunicaciones, que de igual manera que para la transmisión de cada caracter, podemos considerar como una actividad periódica de período igual al tiempo de transmisión del paquete por la red

(dado por la ecuación de la página 7-20), y con tiempo de ejecución de peor caso calculado como indica el modelado de las comunicaciones.

- T_paq2_{485}. Es la tarea que modela la actividad que se realiza para la recepción de cada paquete de comunicaciones, que de igual manera que la tarea anterior podemos considerar como una actividad periódica de período igual al tiempo de transmisión del paquete por la red (dado por la ecuación de la página 7-20), y con tiempo de ejecución de peor caso calculado como indica el modelado de las comunicaciones.

- $T_men_{i_j}$. Son las tareas que modelan las ejecuciones de las operaciones *envía* remotas (acción *ordena_rec* para operaciones remotas en el nudo remoto).

- $Tarea_{i_j}$. Son las tareas de usuario a cuyos tiempo de ejecución hemos añadido los correspondientes a la ejecución de la operaciones *envía* remota en el nudo local o *recibe* local.

Si realizamos el análisis de tiempo real para obtener el tiempo de respuesta al evento E_1, vemos que para el caso en el que no se utilizan prioridades en los mensajes, el tiempo de respuesta para el mensaje M_{1_1} es de 3.353 s., por lo que únicamente con esta contribución la respuesta al evento E_1 ya no cumpliría su plazo. Sin embargo, con el uso de prioridades el tiempo de respuesta del mensaje M_{1_1} es de 239.486 ms., y el tiempo de respuesta total al evento E_1 es 1.198 s., por lo tanto menor que el plazo (1.2 s.). Las utilizaciones de los recursos calculadas a partir del modelado realizado son las siguientes:

$$U_{procesador_1} = 60.15\%$$
$$U_{procesador_2} = 48.79\%$$
$$U_{RS-485} = 82.82\%$$

Pasamos ahora a describir la segunda configuración del sistema distribuido del ejemplo, con la que vamos a probar las ventajas de la eliminación del jitter mediante el uso de servidores esporádicos, tanto para las tareas como para los mensajes. Vamos a detallar las nuevas características para este sistema, a las que añadimos los parámetros de configuración de los servidores esporádicos para comunicaciones (los servidores de las tareas se reducen a

una anotación y cálculo de los instantes de activación en función de sus períodos). Estas características son las siguientes:

- **Eventos y pasos de los atletas por evento.**

Evento	**T** (s.)	**D** (s.)	**Nº Pasos**
E_1	4.0	6.0	40
E_2	4.0	8.0	40
E_3	4.0	12.0	40
E_4	8.0	24.0	80
E_5	8.0	24.0	80
E_6	8.0	32.0	80

- **Nudo 1.**

Tarea	**P**	**C** (s.)
$Tarea_{1_3}$	20	0.05
$Tarea_{2_3}$	19	0.05
$Tarea_{3_3}$	18	0.05
$Tarea_{4_3}$	17	0.05
$Tarea_{5_3}$	16	0.05
$Tarea_{6_3}$	15	0.05
$Tarea_{1_1}$	14	0.05 - 2.25
$Tarea_{2_1}$	13	0.5
$Tarea_{3_1}$	12	0.1
$Tarea_{4_1}$	11	0.1
$Tarea_{5_1}$	10	0.1
$Tarea_{6_1}$	9	0.1

- **Nudo 2.**

Tarea	**P**	**C** (s.)
$Tarea_{1_2}$	19	0.1
$Tarea_{2_2}$	18	0.1
$Tarea_{3_2}$	17	0.1
$Tarea_{4_2}$	16	0.1
$Tarea_{5_2}$	15	0.1
$Tarea_{6_2}$	14	0.1

- **Línea serie RS-485.**

Mensaje	**P**	$\mathbf{l_m}$ **/** $\mathbf{N_{paquetes}}$
M_{1_1}	31	195 / 10
M_{2_1}	30	195 / 10
M_{3_1}	29	55 / 3
M_{4_1}	28	55 / 3
M_{5_1}	27	55 / 3
M_{6_1}	26	55 / 3

- **Bus VME.**

Mensaje	**P**	$\mathbf{l_m}$ **/** $\mathbf{N_{paquetes}}$
M_{1_2}	31	5 / 1
M_{2_2}	30	5 / 1
M_{3_2}	29	5 / 1
M_{4_2}	28	5 / 1
M_{5_2}	27	5 / 1
M_{6_2}	26	5 / 1

- **Parámetros del servidor esporádico para la línea serie RS-485.**

P	**Capacidad**	**T_relleno** (s.)
31	10	4.0
30	10	4.0
29	3	4.0
28	3	8.0
27	3	8.0
26	3	8.0

- **Parámetros del servidor esporádico para el bus VME.**

P	**Capacidad**	**T_relleno** (s.)
31	1	4.0
30	1	4.0
29	1	4.0
28	1	8.0
27	1	8.0
26	1	8.0

Para poder observar las ventajas en la utilización del servidor esporádico para comunicaciones, introducimos el efecto del jitter en el evento E_1 haciendo que $tarea_{1_1}$ ejecute alternativamente el procedimiento de consumo de tiempo con dos valores diferentes (suficientemente distanciados para que el jitter tenga efectos apreciables al intentar conseguir la ejecución de una situación próxima al peor caso).

Para esta configuración del sistema vamos a realizar en primer lugar el análisis de tiempo real con el fin de apoyarnos en él para realizar una prueba comparativa del sistema funcionando con y sin servidor esporádico. Siguiendo el modelado de las colas de mensajes que hemos realizado en este capítulo, y de una forma análoga a los ejemplos anteriores, podemos obtener el modelado para el sistema de este ejemplo y la configuración que acabamos de detallar. Si consideramos el caso en el que no se utiliza servidor esporádico, el

modelado se especifica para cada recurso de la forma que se muestra a continuación: (--- indica que el dato no es relevante):

- **Nudo 1**.

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
$T_paq_{1_2}$	31	4000	---	5.955	---
$T_paq_{2_2}$	31	4000	---	5.955	---
$T_paq_{3_2}$	31	4000	---	5.955	---
$T_paq_{4_2}$	31	8000	---	5.955	---
$T_paq_{5_2}$	31	8000	---	5.955	---
$T_paq_{6_2}$	31	8000	---	5.955	---
Tick	30	20	---	0.470	---
T_car_{1}	29	4.166	---	0.231	---
T_paq1_{485}	29	112.5	---	5.927	1.316
$T_men_{1_2}$	23	4000	---	10.288	---
$T_men_{2_2}$	23	4000	---	10.288	---
$T_men_{3_2}$	23	4000	---	10.288	---
$T_men_{4_2}$	23	8000	---	10.288	---
$T_men_{5_2}$	23	8000	---	10.288	---
$T_men_{6_2}$	23	8000	---	10.288	---
$Tarea_{1_3}$	20	4000	6000	63.137	20
$Tarea_{2_3}$	19	4000	6000	63.137	20
$Tarea_{3_3}$	18	4000	9000	63.137	---
$Tarea_{4_3}$	17	8000	24000	63.137	---
$Tarea_{5_3}$	16	8000	24000	63.137	---
$Tarea_{6_3}$	15	8000	32000	63.137	---
$Tarea_{1_1}$	14	4000	---	2277.906	20
$Tarea_{2_1}$	13	4000	---	527.906	20
$Tarea_{3_1}$	12	4000	---	112.786	---
$Tarea_{4_1}$	11	8000	---	112.786	---
$Tarea_{5_1}$	10	8000	---	112.786	---
$Tarea_{6_1}$	9	8000	---	112.786	---

- **Nudo 2.**

Tarea	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
Tick	30	20	---	0.470	---
T_car	29	4.166	---	0.231	---
T_paq2_{485}	29	112.5	---	4.914	---
$T_men_{1_1}$	23	4000	---	61.398	---
$T_men_{2_1}$	23	4000	---	61.398	---
$T_men_{3_1}$	23	4000	---	23.738	---
$T_men_{4_1}$	23	8000	---	23.738	---
$T_men_{5_1}$	23	8000	---	23.738	---
$T_men_{6_1}$	23	8000	---	23.738	---
$Tarea_{1_2}$	20	4000	---	156.729	48.165
$Tarea_{2_2}$	19	4000	---	156.729	20
$Tarea_{3_2}$	18	4000	---	127.049	---
$Tarea_{4_2}$	17	8000	---	127.049	---
$Tarea_{5_2}$	16	8000	---	127.049	---
$Tarea_{6_2}$	15	8000	---	127.049	---

- **Línea serie RS-485.**

Mensaje	**P**	**T** (ms.)	**D** (ms.)	**C** (ms.)	**B** (ms.)
M_1	31	4000	---	1197.430	119.743
M_2	30	4000	---	1197.430	119.743
M_3	29	4000	---	359.229	---
M_4	28	8000	---	359.229	---
M_5	27	8000	---	359.229	---
M_6	26	8000	---	359.229	---

donde, los términos que aparecen se explican en el modelado de la configuración anterior, y de la misma manera el bus VME tampoco se modela como una red (tiempo de transmisión de paquete igual a cero).

Si realizamos el análisis de tiempo real sobre el evento E_2 obtenemos un tiempo de respuesta de peor caso de 7.234 s. para el caso en el que no consideramos el jitter, es decir, utilizamos servidor esporádico aunque sin tener en cuenta la sobrecarga que supone (relativamente pequeña para los tiempos que se están manejando). Si tenemos en cuenta el jitter y aplicamos las mismas técnicas de análisis sobre el evento E_2 (cálculo reiterado de respuestas y jitter hasta la igualdad de éstos últimos), obtenemos un tiempo de respuesta superior a 8.4 s. en el primer paso del algoritmo de análisis (no seguimos puesto que ya supera el plazo). Por lo tanto, el sistema del ejemplo cumple sus restricciones temporales para las características dadas. Las utilizaciones de los recursos calculadas a partir del modelado realizado son las siguientes:

$$U_{procesador_1} = 99.28\%$$
$$U_{procesador_2} = 32.59\%$$
$$U_{RS-485} = 82.32\%$$

Conseguir que se produzca la situación de peor caso para la respuesta al evento E_2 sobre el que queremos estudiar los efectos del jitter es una labor un tanto difícil, ya que por un lado se pretende que el sistema ejecute libremente (no queremos forzar las situaciones, únicamente la sincronización en el instante inicial), y por otro, deben concurrir circunstancias muy específicas para que se den las condiciones de peor caso. Así pues, para facilitar la realización de las pruebas hemos disminuido el plazo asociado a la respuesta de este evento a 6.0 s., con el fin de discriminar situaciones de peor caso relativas. Los resultados obtenidos de la ejecución del sistema con esta configuración para los casos en los que se utiliza y no se utiliza servidor esporádico se muestran en la Figura 7-13. La ejecución se ha llevado hasta un punto de sincronización de todos los atletas (eventos) situado en 560 pasos.

El jitter se manifiesta sobre las tareas o mensajes de menor prioridad que las tareas o mensajes que lo producen. Cuando no se utiliza servidor esporádico para comunicaciones, el jitter que produce la ejecución de $tarea_{1_1}$ sobre el mensaje M_1 hace que el mensaje M_2 pueda ser expulsado dos veces en el mismo período. Consecuentemente esto provoca un aumento del tiempo de respuesta al evento E_2 que hace que no cumpla el plazo que se le ha impuesto. Esta situación se puede apreciar en la Figura 7-13, en la que se observa como el atleta *2* se queda en la mitad del recorrido cuando no se utiliza servidor esporádico, es decir, el evento E_2 pierde la mitad de los plazos. La pérdida de los plazos sucede justamente cuando

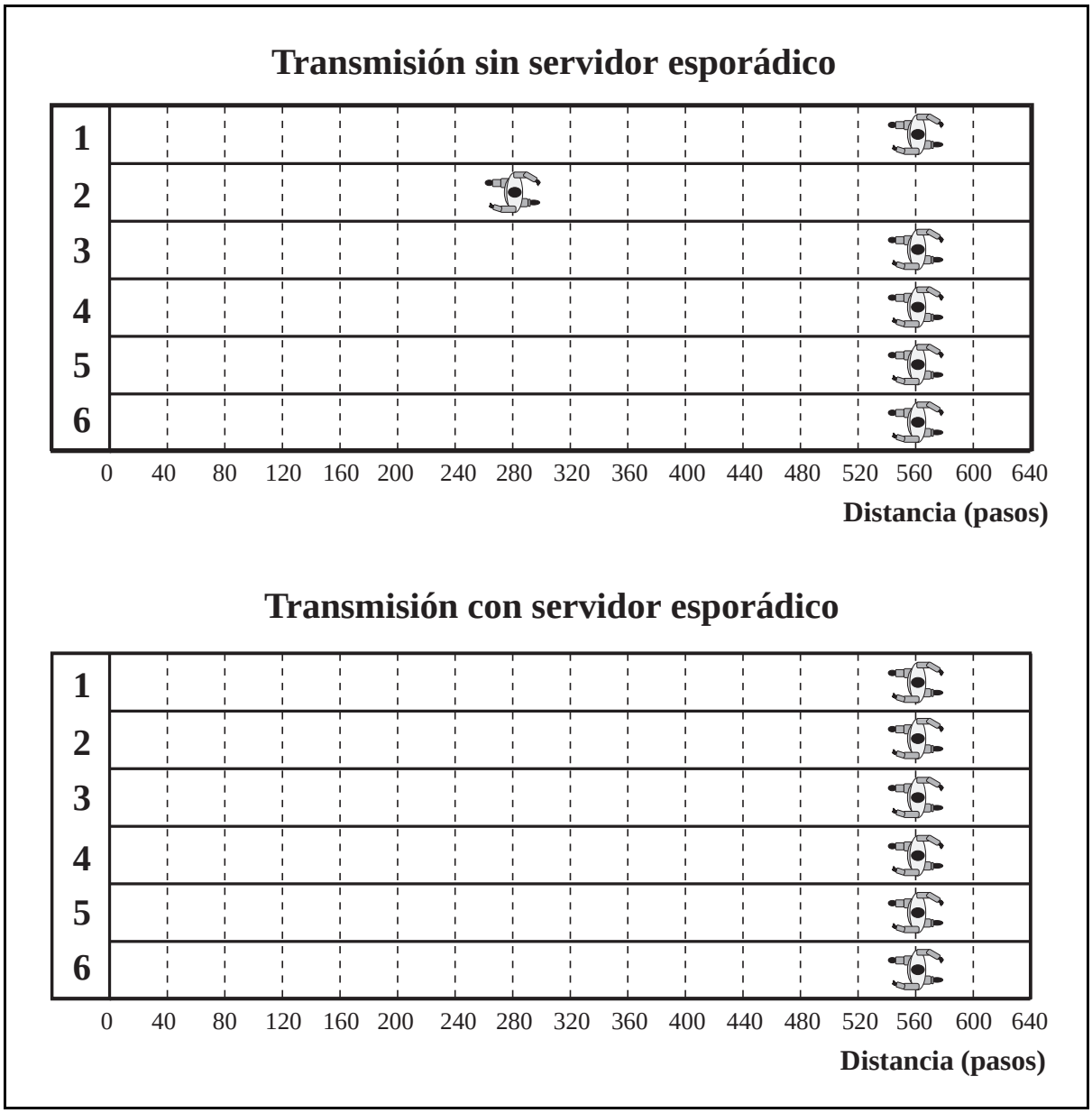

Figura 7-13. Resultados comparativos de la ejecución del sistema con y sin servidor esporádico.

la tarea $tarea_{1_1}$ primero consume 2.25 s. y después 0.05 s., ya que la situación contraria favorece una transmisión de M_2 con menor número de expulsiones.

8. Conclusiones

A modo de conclusiones del trabajo que hemos llevado a cabo y que se presenta en esta Tesis, vamos a hacer una breve revisión del mismo en la que pasaremos revista a los objetivos propuestos en el capítulo 1, en virtud de la medida en la que han sido alcanzados; seguidamente, destacaremos cuáles son las principales contribuciones que aporta, y finalmente, aventuraremos algunas ideas acerca de futuros desarrollos que pueden tomar el presente trabajo como referencia.

8.1. Revisión de los objetivos

Uno de los objetivos principales planteados para la realización de esta Tesis era el desarrollo de una ***metodología global de análisis y diseño de sistemas distribuidos de tiempo real estricto***. Para realizar este desarrollo nos basamos en las técnicas de análisis que se aplican actualmente a los sistemas distribuidos de tiempo real derivadas de la teoría *RMA*.

De esta forma, en el capítulo 2 revisamos el modelo encontrado en la bibliografía para la descripción de sistemas distribuidos de tiempo real estricto gobernados por eventos sobre el que se pueden aplicar directamente las técnicas de análisis existentes. En este modelo, al que denominamos *modelo lineal*, se considera que un sistema distribuido está constituido por múltiples recursos procesadores que se comunican a través de uno o más recursos de comunicación. Las respuestas a los eventos están compuestas por secuencias lineales de acciones (tareas y mensajes) sobre las que se imponen los requerimientos de tiempo real (plazos locales y globales). Además, asumimos una política de planificación expulsora basada en prioridades fijas para permitir el uso de la teoría *RMA* en el análisis del comportamiento temporal de peor caso.

En el capítulo 2 también hacemos un estudio de la problemática que se plantea a raíz del análisis en este tipo de sistemas de tiempo real. Esta problemática se puede resumir en

tres aspectos, que constituyen a la postre el pilar fundamental de la metodología que se ha desarrollado, y que son: la definición de un modelo más general de sistema distribuido que contempla la sincronización, la extensión del análisis a este modelo, y la optimización de la planificación (en cuanto a la asignación de prioridades y a la eliminación del jitter).

Tomando el *modelo lineal* como base para la definición de un sistema distribuido, en el capítulo 3 afrontamos la optimización de la asignación de prioridades. Por un lado, mejoramos la aplicación de la técnica del *templado simulado* a la asignación de prioridades (realizada por Tindell, Burns y Wellings [TIN92]) en el sentido de conseguir la optimización de las soluciones; y por otro, desarrollamos un nuevo algoritmo de planificación heurístico (*HOPA*).

Con objeto de poder evaluar y comparar los dos algoritmos de asignación de prioridades, desarrollamos una herramienta para la aplicación de ambos de forma automática sobre una descripción del sistema distribuido extraída del *modelo lineal*. Además, realizamos una herramienta de generación automática de ejemplos que nos permite analizar el funcionamiento de los algoritmos de planificación de una manera más profunda y sobre un número mayor de casos. Esta herramienta opera sobre la descripción del sistema empleada en la herramienta de asignación de prioridades. Ambas herramientas se han desarrollado en lenguaje Ada.

De la comparación entre los resultados que se obtienen a partir de la aplicación de las herramientas desarrolladas podemos concluir que:

- Ambas técnicas permiten optimizar la asignación de prioridades.

- El algoritmo heurístico encuentra en general más y mejores soluciones que el *templado simulado*. Estas son tanto mejores cuanto mayores sean los plazos con respecto a los períodos, y cuanto mayor sea la utilización de los recursos.

- El tiempo de CPU consumido en la ejecución de nuestro algoritmo heurístico es en promedio dos órdenes de magnitud menor que el empleado por el *templado simulado*.

En el capítulo 4 extendemos el *modelo lineal* de sistema distribuido a un *modelo general* que permite representar a un mayor número de este tipo de sistemas de tiempo real. Este nuevo modelo contempla por un lado la sincronización entre tareas, y por otro, la activación de las mismas por combinaciones de eventos, y la generación de varios eventos por parte de una tarea. Este modelo es muy adecuado para el tratamiento de sistemas de paso de mensajes.

Con la definición del *modelo general*, desarrollamos además una nueva técnica de análisis para el mismo tomando como base la que se aplicaba al *modelo lineal*. Esta técnica permite igualmente obtener la respuesta de peor caso del sistema garantizando así el cumplimiento de los requerimientos de tiempo que se hayan impuesto (aunque sigue siendo ligeramente pesimista).

Una vez que hemos extendido el modelo y hemos definido el análisis sobre este modelo, extendemos los algoritmos de asignación de prioridades para que sean aplicables sobre este modelo, y de la misma forma, extendemos las herramientas de asignación de prioridades y de generación automática de ejemplos. La aplicación de estas herramientas a algunos ejemplos sencillos de sistemas distribuidos que responden al *modelo general* demuestra que es posible realizar la asignación de prioridades para este modelo. Los métodos empleados para la planificación arrojan similares resultados a los obtenidos para el *modelo lineal* (el algoritmo heurístico es más rápido, y si encuentra solución, ésta es generalmente mejor que la que pueda alcanzar el algoritmo de *templado simulado*). Sin embargo, en este caso el *templado simulado* es capaz de alcanzar soluciones planificables para algunos casos sobre los que el algoritmo heurístico da un resultado no planificable. Por ello hemos propuesto la aplicación en primer lugar del algoritmo heurístico de planificación (obtiene una respuesta más rápida), y si éste no consigue alcanzar una solución planificable para el sistema, entonces se aplica el algoritmo de *templado simulado*.

Uno de los principales resultados que obtenemos de la aplicación de las herramientas desarrolladas es la demostración del enorme impacto que tiene el *jitter* en los sistemas distribuidos de tiempo real (se puede perder hasta un 50% en la utilización de los recursos), y por lo tanto, de la imperiosa necesidad de eliminarlo para conseguir mayores utilizaciones de los recursos. En el capítulo 6 proponemos la utilización del *servidor esporádico* (tanto en

los procesadores como en las redes) para eliminar el efecto negativo del *jitter* en este tipo de sistemas. El *servidor esporádico* para las tareas ya existía, pero aún no se había aplicado a la planificación de las comunicaciones. De esta manera, definimos el *servidor esporádico para comunicaciones* como mecanismo de control en la transmisión de mensajes por las redes de comunicación de tiempo real. Concretamos esta definición para redes punto a punto y para redes multipunto.

El otro objetivo importante planteado en este trabajo era la ***aplicación de la metodología desarrollada a sistemas distribuidos programados en lenguaje Ada***. En el capítulo 5 llevamos a cabo una implementación en Ada de redes de comunicación de tiempo real estricto con el propósito de construir una plataforma de trabajo para la realización de pruebas.

Esta implementación se basa en la interfase para comunicaciones de tiempo real del POSIX [POS93] que utiliza el mecanismo de paso de mensajes con prioridad para la intercomunicación de procesos o tareas. En la elaboración de las redes de comunicación se implementan las colas de mensajes definidas en esta interfase. Las colas de mensajes son colas de prioridad en las que las tareas depositan y extraen los mensajes con los que se comunican, independientemente del nudo del sistema distribuido en el que se estén ejecutando. Implementamos las redes de comunicación de tiempo real para líneas punto a punto (líneas serie RS-232 y RS-485) y para líneas multipunto (bus VME).

El banco de trabajo desarrollado utiliza las redes de comunicación implementadas para realizar la comunicación entre procesadores mediante el intercambio de mensajes con requerimientos de tiempo real. El sistema distribuido que constituye este banco de trabajo está formado por tarjetas procesadoras basadas en el microprocesador MC68010 de Motorola sobre un bus VME con puertas serie RS-232 y RS-485, y por PC's con microprocesadores 80386/80486 de Intel que únicamente disponen de una puerta serie RS-232.

En el capítulo 6 llevamos a cabo la implementación del *servidor esporádico para comunicaciones* sobre las redes de comunicación desarrolladas. De esta forma disponemos ya de una plataforma de trabajo sobre la que podemos realizar algunos ejemplos prácticos, y por lo tanto, podemos aplicar la metodología desarrollada.

En el capítulo 7 modelamos las redes de comunicación implementadas para hacer posible el análisis y la planificación de los sistemas que las utilizan. Este modelado consiste por un lado, en la definición de modelos de representación de las operaciones que se realizan en las redes, es decir, en la identificación de las acciones o tareas que forman parte de dichas operaciones; y por otro, en la evaluación de los tiempos de peor caso para cada acción.

También se realiza el modelado del servidor esporádico para comunicaciones en los mismos términos que las redes de comunicación. A partir de este modelado observamos que la implementación del servidor produce una cierta sobrecarga, pero que es relativamente pequeña comparada con los aumentos que se pueden conseguir en las utilizaciones de los recursos. Además, la importancia de esta sobrecarga disminuye con el incremento en la potencia de los recursos.

Finalmente, elaboramos algunos ejemplos en los que se pueden ver algunas aplicaciones del uso de las redes de comunicación implementadas. En estas aplicaciones ponemos de manifiesto las peculiaridades propias de la utilización de dichas redes y del modelado de estas aplicaciones para el análisis de tiempo real. También podemos ver las ventajas del mecanismo de planificación desarrollado para la eliminación del *jitter*.

El modelado realizado para las redes de comunicación de tiempo real (unido a la aplicación de las técnicas de análisis y planificación), junto con las pautas de programación de aplicaciones que se han marcado a través de los ejemplos, constituyen en sí mismos una metodología que nos permite diseñar, implementar y analizar sistemas de tiempo real estricto con un alto grado de utilización, garantizando que se cumplen los requerimientos de tiempo real, y programados en lenguaje Ada.

8.2. Contribuciones de este trabajo

Uno de los aspectos más importantes del trabajo presentado en esta Tesis es el desarrollo de nuevas técnicas de análisis y planificación de sistemas distribuidos de tiempo real estricto.

En el capítulo 3 mostramos cómo a partir de las técnicas de análisis existentes para este tipo de sistemas basadas en la Teoría *RMA*, hemos sido capaces de optimizar la asignación de prioridades a las tareas y a los mensajes que constituyen estos sistemas. Esta optimización se ha conseguido abordar desde dos perspectivas diferentes: por un lado, mediante el desarrollo de un nuevo algoritmo heurístico de planificación que obtiene buenas soluciones y muy rápidamente; y por el otro, a través de una modificación de la técnica existente de asignación de prioridades basada en el *templado simulado* que permite la optimización de las soluciones alcanzadas, y que además constituye un buen punto de referencia para evaluar la calidad de las asignaciones de prioridades realizadas por el heurístico.

En el capítulo 4 extendimos el actual modelo de representación para los sistemas distribuidos a un nuevo modelo más general que permite la sincronización. También dotamos a este nuevo modelo de las técnicas de análisis adecuadas para la obtención de los tiempos de respuesta de peor caso y su posterior comparación con los plazos. Y más aún, a partir del nuevo análisis desarrollado, también extendimos las técnicas de asignación de prioridades desarrolladas en el capítulo 3 para que sean operativas sobre este modelo.

Otro de los aspectos importantes de este trabajo es el desarrollo del *servidor esporádico para comunicaciones* como mecanismo de control de la transmisión de mensajes por las redes de comunicación de tiempo real, llevado a cabo en el capítulo 6. Además, utilizamos este mecanismo conjuntamente con el *servidor esporádico* que se aplica a las tareas, para eliminar el efecto del *jitter* en la planificación de los sistemas distribuidos, con lo que se consiguen aumentos en las utilizaciones de los recursos del sistema de hasta un 50%.

El último aspecto que queremos poner de relieve es el desarrollo de redes de comunicación de tiempo real que se ha llevado a cabo en el capítulo 5. Estas redes incorporan la implementación de un planificador con *servidor esporádico para comunicaciones* (acometido en el capítulo 6), y a partir de ellas, hemos desarrollado un banco de trabajo que nos permite la realización de pruebas. Las redes que hemos desarrollado posibilitan la comunicación entre tareas en tiempo real a través de un mecanismo de paso de mensajes con independencia de los nudos en los que estén localizadas estas tareas.

Finalmente, en el capítulo 7 hemos visto a través del modelado de las redes de comunicación desarrolladas y de varios ejemplos, cómo aplicar una metodología al diseño y a la programación de sistemas distribuidos de tiempo real estricto en lenguaje Ada.

8.3. Trabajo futuro

Las técnicas actuales para el análisis de sistemas distribuidos de tiempo real estricto, sean las ya existentes, o sean las desarrolladas en esta Tesis, son ligeramente pesimistas en la obtención de la respuesta temporal de peor caso, en gran medida porque se basan en el análisis de cada recurso del sistema por separado. Por lo tanto, aún queda pendiente el desarrollo de una técnica capaz de realizar el análisis exacto de este tipo de sistemas con tareas que se suspenden y teniendo en cuenta el *jitter*. Esta técnica deberá contemplar quizás los aspectos globales de la ejecución de las tareas y de la transmisión de los mensajes en el sistema completo, de modo que sus resultados puedan ser lo más próximos posible al sistema en funcionamiento real.

Si tomamos como referencia el algoritmo heurístico para la asignación de prioridades que hemos desarrollado, se puede intentar encontrar un algoritmo similar para resolver el problema de la asignación de tareas a procesadores y de mensajes a redes de comunicación. De esta forma, se podría desarrollar una herramienta integrada de asignación de prioridades y de asignación de acciones a recursos, en la misma línea que la aplicación del *templado simulado* realizada por Tindell, Burns y Wellings [TIN92], pero con una aplicación mucho más global (al *modelo general* de sistema distribuido definido en esta Tesis).

Dado que la herramienta de planificación desarrollada en esta Tesis tiene un carácter exclusivamente experimental, se puede pensar en desarrollar una herramienta "no experimental" de análisis y planificación, y por lo tanto mucho más elaborada, en la que se contemplarían todas las técnicas de análisis y de asignación de prioridades utilizadas y desarrolladas en esta Tesis, incluyendo aquellos aspectos que quedaron fuera de la herramienta experimental (como la automatización del cálculo de los términos de bloqueo a partir de las secciones críticas).

A partir del software que hemos desarrollado en nuestras redes de comunicación de tiempo real estricto, se podrían implementar fácilmente otras redes de comunicación. En concreto sería bastante sencillo realizar la implementación de estas comunicaciones de tiempo real para un bus CAN. También sería inmediata la aplicación del *servidor esporádico para comunicaciones* a una red como ésta. Otra posible aplicación podría ser el uso de una red bastante extendida actualmente como es la Ethernet, para realizar comunicaciones punto a punto en tiempo real.

Un trabajo interesante que se puede abordar tomando como punto de partida nuestra implementación de redes de comunicación es la implementación del subsistema de comunicación de particiones (*Partition Communication Subsistem*) del anexo de sistemas distribuidos del Ada95. Esto nos permitiría conseguir un comportamiento de tiempo real estricto en programas que se comunicasen de la forma estandarizada en el Ada95.

A. Implementación en Ada de las redes de comunicación

A.1. Especificaciones de los paquetes Ada

En este apartado daremos una relación de las especificaciones de todos los paquetes Ada que componen la implementación realizada en Ada de las colas de mensajes y de las comunicaciones, dentro de lo que constituye el desarrollo de redes de comunicación de tiempo real.

***Package** tipos_de_colas*

```
package tipos_de_colas is

type byte is new integer range 0..255;
for byte'size use 8;

max_numero_de_colas                : constant positive := 10;
max_numero_de_mensajes             : constant positive := 100;
max_numero_de_descriptores         : constant positive := 200;
max_numero_de_descriptores_locales : constant positive := 100;
max_longitud_de_mensaje            : constant positive := 256;
long_nombre_de_cola                : constant positive := 8;

type tipo_nombre is new string (1..long_nombre_de_cola);
type tipo_prioridad is new positive range 1..32;
type tipo_acceso is (lectura, escritura, lectura_escritura);
type tipo_mensaje is array (1..max_longitud_de_mensaje) of byte;
type tipo_atributos is
   record
      numero_mensajes  : positive;
      longitud_mensaje : positive;
      mensajes_en_cola : integer;
      global           : boolean;
   end record;
type tipo_error is
   record
      envia   : integer;
      cierra  : integer;
      interno : integer;
   end record;

cola_ya_creada                     : exception;
cola_perteneciente_a_otro_nudo     : exception;
cola_en_proceso_de_destruccion     : exception;
```

```
numero_de_mensajes_fuera_de_rango  : exception;
longitud_de_mensaje_fuera_de_rango : exception;
no_hay_colas_libres                : exception;
no_hay_descriptores_libres         : exception;
descriptor_no_activo               : exception;
cola_no_existente                  : exception;
insuficiente_espacio_en_la_cola    : exception;
acceso_no_permitido                : exception;
problemas_de_comunicacion          : exception;

end tipos_de_colas;
```

***Package** configuracion*

```
with tipos_de_colas;

package configuracion is

max_elementos_configurables : constant positive := 30;

type tipo_nudo is new integer range 1..32;
type tipo_red is new integer range 1..3;

procedure configura_nudo (nombre : in tipos_de_colas.tipo_nombre;
                          nudo   : in tipo_nudo;
                          red    : in tipo_red);
procedure obten_nudo (nombre : in  tipos_de_colas.tipo_nombre;
                      global : out boolean;
                      nudo   : out tipo_nudo);
procedure obten_red (nombre : in  tipos_de_colas.tipo_nombre;
                     red    : out tipo_nudo);
procedure establece_nudo_actual (nudo : in tipo_nudo);
function obten_nudo_actual return tipo_nudo;
procedure establece_uso_de_servidor_esporadico (uso : in boolean);
function uso_servidor_esporadico return boolean;

lista_completa      : exception;
nudo_no_configurado : exception;

end configuracion;
```

***Package** memoria*

```
with tipos de colas;
with configuracion;

package memoria is

max_numero_de_segmentos : constant positive := 32;
longitud_de_segmento    : constant positive := 1024;
max_espacio_de_mensajes : constant positive := 20;
longitud_de_protocolo   : constant positive := 5;
longitud_de_cabeza      : constant positive := 5;
longitud_de_paquete     : constant positive := 20;
```

```
max_numero_de_paquetes  : constant positive := 15;

type tipo_lista_de_segmentos is array
   (1..max_numero_de_segmentos) of positive;
type tipo_segmento is new string (max_longitud_de_segmento);
type tipo_protocolo_de_mensaje is array
   (1..longitud_de_protocolo) of tipos_de_colas.byte;
type tipo_cabeza_de_mensaje is array
   (1..longitud_de_cabeza) of tipos_de_colas.byte;
type tipo_espacio_de_mensajes is
   record
      cabeza                  : tipo_cabeza_de_mensaje;
      protocolo               : tipo_protocolo_de_mensaje;
      mensaje                 : tipos_de_colas.tipo_mensaje;
      long_ultimo_paquete     : integer;
   end record;

espacio_de_mensajes :
   array (1..max_espacio_de_mensajes) of tipo_espacio_de_mensajes;
espacio_de_memoria :
   array (1..max_numero_de_segmentos) of tipo_segmento;

procedure libera_segmentos
   (numero_de_segmentos  : in positive;
    segmentos_reservados : in tipo_lista_de_segmentos);
procedure reserva_segmentos
   (numero_de_segmentos  : in  positive
    segmentos_reservados : out tipo_lista_de_segmentos);
procedure libera_espacio_mensaje (mensaje : in positive);
function reserva_espacio_mensaje return positive;

no_hay_espacio_de_memoria  : exception;
no_hay_espacio_de_mensajes : exception;

end memoria;
```

***Package** colas_de_mensajes*

```
with configuracion;
with tipos_de_colas;

package colas_de_mensajes is

type tipo_descriptor is limited private;

procedure abre (nombre           : in  tipos_de_colas.tipo_nombre;
                acceso           : in  tipos_de_colas.tipo_acceso;
                crear            : in  boolean;
                numero_mensajes  : in  positive;
                longitud_mensaje : in  positive;
                descriptor       : out tipo_descriptor);
procedure cierra (descriptor : in tipo_descriptor);
procedure destruye (nombre : in tipos_de_colas.tipo_nombre);
procedure envia
        (descriptor       : in tipo_descriptor;
         mensaje          : in tipos_de_colas.tipo_mensaje;
         longitud_mensaje : in positive;
         prioridad        : in tipos_de_colas.tipo_prioridad);
procedure recibe
        (descriptor       : in  tipo_descriptor;
```

```
          mensaje          : out tipos_de_colas.tipo_mensaje;
          longitud_mensaje : out positive;
          prioridad        : out tipos_de_colas.tipo_prioridad);

procedure recibe_si_hay
         (descriptor       : in  tipo_descriptor;
          mensaje          : out tipos_de_colas.tipo_mensaje;
          longitud_mensaje : out positive;
          prioridad        : out tipos_de_colas.tipo_prioridad;
          no_hay           : out boolean);

procedure coge_atributos
         (descriptor : in  tipo_descriptor;
          atributos  : out tipos_de_colas.tipo_atributos);
function obten_error return tipos_de_colas.tipo_error;

private

type tipo_descriptor is new positive range
     1..tipos_de_colas.max_numero_de_descriptores;

end colas_de_mensajes;
```

Package *control_de_colas*

```
with tipos_de_colas;
with configuracion;
with memoria;
with artk;

package control_de_colas is

subtype tipo_descriptor is positive range
        1..tipos_de_colas.max_numero_de_descriptores;
subtype tipo_descriptor_local is tipo_descriptor range
        1..tipos_de_colas.max_numero_de_descriptores_locales;
subtype tipo_descriptor_remoto is tipo_descriptor range
        (tipos_de_colas.max_numero_de_descriptores_locales+1)..
        tipos_de_colas.max_numero_de_descriptores;
type tipo_cuerpo_descriptor_remoto is
   record
      activo              : boolean := false;
      indice_descriptor   : tipo_descriptor;
      nudo                : configuracion.tipo_nudo;
      red                 : configuracion.tipo_red;
      acceso              : tipos_de_colas.tipo_acceso;
      suspendido          : boolean;
      identificador_tarea : artk.task_ids.task_id;
      prioridad_tarea     : artk.dynamic_priorities.priority;
   end record;
type tipo_cuerpo_descriptor_local is
   record
      activo              : boolean := false;
      indice_nombre       : positive;
      indice_descriptor   : tipo_descriptor;
      acceso              : tipos_de_colas.tipo_acceso;
      suspendido          : boolean;
      identificador_tarea : artk.task_ids.task_id;
      prioridad_tarea     : artk.dynamic_priorities.priority;
   end record;
```

```
lista_descriptores_remotos : array (tipo_descriptor_remoto)
                             of tipo_cuerpo_descriptor_remoto;
lista_descriptores_locales : array (tipo_descriptor_local)
                             of tipo_cuerpo_descriptor_local;

task sincroniza is
   pragma priority(24);
   entry crea
        (nombre              : in  tipos_de_colas.tipo_nombre;
         acceso              : in  tipos_de_colas.tipo_acceso;
         numero_mensajes     : in  positive;
         longitud_mensaje    : in  positive;
         identificador_tarea : in  artk.task_ids.task_id;
         global              : in  boolean;
         descriptor          : out tipo_descriptor);
   entry abre
        (nombre              : in  tipos_de_colas.tipo_nombre;
         acceso              : in  tipos_de_colas.tipo_acceso;
         identificador_tarea : in  artk.task_ids.task_id;
         descriptor          : out tipo_descriptor);
   entry cierra (descriptor : in tipo_descriptor);
   entry destruye (nombre : in tipos_de_colas.tipo_nombre);
   entry envia
        (descriptor       : in tipo_descriptor;
         mensaje          : in tipos_de_colas.tipo_mensaje;
         longitud_mensaje : in positive;
         prioridad        : in tipos_de_colas.tipo_prioridad);
   entry recibe
        (descriptor       : in  tipo_descriptor;
         mensaje          : out tipos_de_colas.tipo_mensaje;
         longitud_mensaje : out positive;
         prioridad        : out tipos_de_colas.tipo_prioridad;
         no_hay           : out boolean);
   entry recibe_si_hay
        (descriptor       : in  tipo_descriptor;
         mensaje          : out tipos_de_colas.tipo_mensaje;
         longitud_mensaje : out positive;
         prioridad        : out tipos_de_colas.tipo_prioridad;
         no_hay           : out boolean);
   entry coge_atributos
        (descriptor : in  tipo_descriptor;
         atributos  : out tipos_de_colas.tipo_atributos);
   entry ejecuta_remoto (red      : in configuracion.tipo_red;
                         mensaje  : in positive);
end sincroniza;

task recursos_remotos is
   pragma priority(23);
   entry abre (descriptor : out tipo_descriptor);
   entry cierra (descriptor : in tipo_descriptor);
end recursos_remotos;

procedure recibe (red     : in configuracion.tipo_red;
                  mensaje : in positive);

function errores_remotos return tipos_de_colas.tipo_error;

cola_de_espera_completa : exception;

end control_de_colas;
```

***Package** protocolo*

```
with tipos_de_colas; use tipos_de_colas;

package protocolo is

-- Constantes que definen los códigos de operación y mensaje.
-- Byte 2 del protocolo.

abre           : constant byte := 1;
cierra         : constant byte := 2;
envia          : constant byte := 3;
recibe         : constant byte := 4;
recibe_si_hay  : constant byte := 5;
coge_atributos : constant byte := 6;

-- Constantes para el atributo global.

global            : constant byte := 0;
no_global         : constant byte := 1;

-- Códigos de error en los mensajes, y mensaje, respuesta o error.
-- Byte 1 del protocolo.

mensaje        : constant byte := 0;
error          : constant byte := 1;
respuesta      : constant byte := 2;
no_hay         : constant byte := 3;
error_cola_en_proceso_de_destruccion     : constant byte := 4;
error_longitud_de_mensaje_fuera_de_rango : constant byte := 5;
error_no_hay_descriptores_libres         : constant byte := 6;
error_descriptor_no_activo               : constant byte := 7;
error_cola_no_existente                  : constant byte := 8;
error_insuficiente_espacio_en_la_cola    : constant byte := 9;
error_cola_de_espera_completa            : constant byte := 10;

-- Byte de comienzo y final de mensaje.

stx : tipos_de_colas.byte := 5;
pragma export (assembler,stx);
pragma external_name (stx,"STX");

etx : tipos_de_colas.byte := 13;
pragma export (assembler,etx);
pragma external_name (etx,"ETX");

end protocolo;
```

***Package** red_transmision*

```
with tipos_de_colas;
with configuracion;

package red_transmision is

procedure transmite
               (nudo      : in configuracion.tipo_nudo;
                red       : in configuracion.tipo_red;
```

```
                mensaje   : in positive;
                longitud  : in integer;
                prioridad : in tipos_de_colas.tipo_prioridad);

no_hay_espacio_en_la_cola : exception;

end red_transmision;
```

***Package** red_recepcion*

```
with tipos_de_colas;
with configuracion;

package red_recepcion is

procedure encola (red       : in configuracion.tipo_red;
                  mensaje   : in positive;
                  prioridad : in tipos_de_colas.tipo_prioridad);

no_hay_espacio_en_la_cola : exception;

end red_recepcion;
```

***Package** priority_queue*

```
generic

   qsize : in positive;
   type element is private;
   type priority_t is private;
   with function ">" (left,right : in priority_t) return boolean;
   with function compare (left,right : in element) return boolean;

package piority_queue is

function empty return boolean;
function full return boolean;
procedure enqueue(e : in element;
                  p : in priority_t);
procedure dequeue(e : out element;
                  p : out priority_t);
procedure dequeue_middle(e     : in out element;
                         p     : out priority_t;
                         found : out boolean);
procedure read_first (e : out element;
                      p : out priority_t);

empty_queue : exception;
full_queue  : exception;

end priority_queue;
```

***Package** SS_comunicaciones*

```
generic

   max_elementos_cola_servidor : in positive;
   max_elementos_cola_rellenos : in positive;
   type tipo_prioridad is range <>;
   type tipo_paquete is private;

package SS_comunicaciones is

procedure establece_parametros
          (prioridad         : in tipo_prioridad;
           capacidad_inicial : in integer;
           periodo_relleno   : in duration);
procedure introduce (paquete   : in tipo_paquete;
                     prioridad : in tipo_prioridad);
procedure extrae (paquete : out tipo_paquete);
function cola_vacia return boolean;
function espacio_disponible return integer;

no_hay_espacio_en_la_cola_servidor : exception;
no_hay_espacio_en_la_cola_rellenos : exception;

end SS_comunicaciones;
```

***Package** SS_distribuido*

```
generic

   max_elementos_cola_servidor : in positive;
   max_elementos_cola_rellenos : in positive;
   type tipo_prioridad is range <>;
   type tipo_paquete is private;
   with procedure cambio_a_normal (prioridad : in tipo_prioridad);

package SS_distribuido is

procedure establece_parametros
          (prioridad         : in tipo_prioridad;
           capacidad_inicial : in integer;
           periodo_relleno   : in duration);
procedure introduce (paquete   : in tipo_paquete;
                     prioridad : in tipo_prioridad);
procedure extrae (paquete : out tipo_paquete);
function cola_vacia return boolean;
function espacio_disponible return integer;
procedure estado (prioridad   : out tipo_prioridad;
                  modo_normal : out boolean);

no_hay_espacio_en_la_cola_servidor : exception;
no_hay_espacio_en_la_cola_rellenos : exception;

end SS_distribuido;
```

***Package** ss_comunicaciones_232*

```
with tipos_de_colas;
with ss_comunicaciones;

package ss_comunicaciones_232 is new ss_comunicaciones
      (max_elementos_cola_servidor => 300,
       max_elementos_cola_rellenos => 100,
       tipo_prioridad => tipos_de_colas.tipo_prioridad,
       tipo_paquete   => positive);
```

***Package** ss_comunicaciones_485*

```
with tipos_de_colas;
with ss_comunicaciones;

package ss_comunicaciones_485 is new ss_comunicaciones
      (max_elementos_cola_servidor => 300,
       max_elementos_cola_rellenos => 100,
       tipo_prioridad => tipos_de_colas.tipo_prioridad,
       tipo_paquete   => positive);
```

***Package** ss_comunicaciones_bus*

```
with tipos_de_colas;
with ss_distribuido;
with hardware;

package ss_comunicaciones_bus is new ss_distribuido
      (max_elementos_cola_servidor => 300,
       max_elementos_cola_rellenos => 100,
       tipo_prioridad => tipos_de_colas.tipo_prioridad,
       tipo_paquete   => positive,
       cambio_a_normal => hardware.marca_cambio_a_normal);
```

***Package** comunicaciones_transmision_232*

```
with tipos_de_colas;

package comunicaciones_transmision_232 is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;
```

```
end comunicaciones_transmision_232;
```

***Package** comunicaciones_transmision_485*

```
with tipos_de_colas;

package comunicaciones_transmision_485 is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;

end comunicaciones_transmision_485;
```

***Package** comunicaciones_transmision_bus*

```
with tipos_de_colas;

package comunicaciones_transmision_bus is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;

end comunicaciones_transmision_bus;
```

***Package** comunicaciones_transmision_232_ss*

```
with tipos_de_colas;

package comunicaciones_transmision_232_ss is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;

end comunicaciones_transmision_232_ss;
```

***Package** comunicaciones_transmision_485_ss*

```
with tipos_de_colas;

package comunicaciones_transmision_485_ss is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;

end comunicaciones_transmision_485_ss;
```

***Package** comunicaciones_transmision_bus_ss*

```
with tipos_de_colas;

package comunicaciones_transmision_bus_ss is

procedure transmite
               (mensaje   : in positive;
                prioridad : in tipos_de_colas.tipo_prioridad);
procedure orden_transmision;

no_hay_espacio_en_la_cola : exception;

end comunicaciones_transmision_bus_ss;
```

***Package** comunicaciones_recepcion_232*

```
package comunicaciones_recepcion_232 is

procedure recibe;

end comunicaciones_recepcion_232;
```

***Package** comunicaciones_recepcion_485*

```
package comunicaciones_recepcion_485 is

procedure recibe;

end comunicaciones_recepcion_485;
```

***Package** comunicaciones_recepcion_bus*

```
package comunicaciones_recepcion_bus is

procedure recibe;

end comunicaciones_recepcion_bus;
```

***Package** comunicaciones*

```
package comunicaciones is

end comunicaciones;
```

***Package** hardware*

```
with system;
with tipos_de_colas; use tipos_de_colas;
with configuracion; use configuracion;

package hardware is

num_proc_en_bus : constant configuracion.tipo_nudo := 4;

subtype tipo_nudo_bus is
        configuracion.tipo_nudo range 1..num_proc_en_bus;

-- Configuración de direcciones de comunicación.

direccion_buffer : constant system.address :=
                                    system.value("16#0017F000#");
direccion_semaforo_cola : constant system.address :=
                                    system.value("16#0017F200#");
direccion_cola_bus : constant system.address :=
                                    system.value("16#0017F300#");

-- Semáforo.

semaforo_cola : tipos_de_colas.byte := 0;
for semaforo_cola use at direccion_semaforo_cola;

-- Cola de control del bus.

type tipo_elemento_cola_bus is
   record
      prioridad_normal : boolean;
      cola_vacia       : boolean := true;
      prioridad        : tipos_de_colas.tipo_prioridad :=
                         tipos_de_colas.tipo_prioridad'first;
   end record;
type tipo_lista_procesadores is
     array (1..num_proc_en_bus) of tipo_elemento_cola_bus;
type tipo_cola_bus is
```

```
   record
      estado_transmitiendo : boolean := false;
      disparo_transmision  : boolean := false;
      lista_procesadores   : tipo_lista_procesadores;
   end record;

cola_bus : tipo_cola_bus;
for cola_bus use at direccion_cola_bus;

-- CSR de cada tarjeta.

type bit is range 0..1;
for bit'size use 1;

type tipo_csr_u is
   record
      irq_local_2_1 : bit;
      irq_local_2_2 : bit;
      irq_local_2_3 : bit;
      irq_local_7   : bit;
   end record;
for tipo_csr_u use
   record at mod 1;
      irq_local_2_1 at 0 range 7..7;
      irq_local_2_2 at 0 range 6..6;
      irq_local_2_3 at 0 range 5..5;
      irq_local_7   at 0 range 4..4;
   end record;
for tipo_csr_u'size use 8;

csr_nudo_1 : tipo_csr_u;
for csr_nudo_1 use at system.value("16#00FF0000#");

csr_nudo_2 : tipo_csr_u;
for csr_nudo_2 use at system.value("16#00FF0100#");

-- Buffers de transmisión de las líneas serie.

buf_232 : tipos_de_colas.tipo_mensaje;
pragma export (assembler,buf_232);
pragma external_name (buf_232,"BUF_TRANS_B");

buf_485 : tipos_de_colas.tipo_mensaje;
pragma export (assembler,buf_485);
pragma external_name (buf_485,"BUF_TRANS_A");

-- Procedimiento que marca el cambio de modo de transmisión en el
-- servidor esporádico distribuido.

procedure marca_cambio_a_normal
          (prioridad : in tipos_de_colas.tipo_prioridad);

end hardware;
```

A.2. Estructura de datos de las colas de mensajes

En este apartado vamos a describir la parte de la estructura de datos de las colas de mensajes que no es visible para los programas de usuario, destacando aquellas constantes, tipos o variables, que más importancia tienen en la constitución de las colas o en el control de las mismas. Para realizar esta descripción, consideramos los siguientes apartados:

1.- Constantes.

max_numero_de_segmentos : **constant** positive;

Número máximo de segmentos en el nudo.

longitud_de_segmento : **constant** positive;

Longitud del segmento en *bytes*.

max_espacio_de_mensajes : **constant** positive;

Número máximo de mensajes en el espacio de mensajes.

longitud_de_protocolo : **constant** positive;

Número de *bytes* utilizados para el protocolo de las operaciones remotas.

longitud_de_cabeza : **constant** positive;

Número de *bytes* utilizados para el protocolo de comunicaciones, es decir, para la partición de mensajes en paquetes de comunicación.

longitud_de_paquete : **constant** positive;

Longitud del paquete de comunicaciones (unidad mínima de comunicación no expulsable).

max_numero_de_paquetes : **constant** positive;

Número máximo de paquetes de comunicación por mensaje.

max_elementos_cola : **constant** positive := 100;

Dimensión del espacio reservado para los elementos de las colas de espera.

2.- Tipos.

type *tipo_lista_de_segmentos* **is array**
(1..*max_numero_de_segmentos*) **of** positive;

Lista de índices a segmentos.

type *tipo_segmento* **is new** string (*max_longitud_de_segmento*);

Definición de un segmento.

type *tipo_protocolo_de_mensaje* **is array**
(1..*longitud_de_protocolo*) **of** *tipos_de_colas.byte*;

Espacio para el protocolo de codificación de operaciones remotas.

type *tipo_cabeza_de_mensaje* **is array**
(1..*longitud_de_cabeza*) **of** *tipos_de_colas.byte*;

Espacio para el protocolo de comunicaciones.

type *tipo_espacio_de_mensajes* **is**
record
cabeza : *tipo_cabeza_de_mensaje*;
protocolo : *tipo_protocolo_de_mensaje*;
mensaje : *tipos_de_colas.tipo_mensaje*;
long_ultimo_paquete : integer;
end record;

Definición del *mensaje* del *espacio de mensajes*. Está formado por el protocolo de comunicaciones y de codificación de operaciones remotas, por el mensaje como elemento de las colas de mensajes (si se trata de una orden de operación puede ser vacío), y por la longitud del último paquete de comunicaciones (asociado a la partición en paquetes).

subtype *tipo_descriptor* **is** positive **range**
1..*tipos_de_colas.max_numero_de_descriptores*;

subtype *tipo_descriptor_local* **is** *tipo_descriptor* **range**
1..*tipos_de_colas.max_numero_de_descriptores_locales*;

subtype *tipo_descriptor_remoto* **is** *tipo_descriptor* **range**

(*tipos_de_colas.max_numero_de_descriptores_locales*+1)..

tipos_de_colas.max_numero_de_descriptores;

Definición de tipos de los descriptores manejados internamente.

type *tipo_cuerpo_descriptor_remoto* **is**
record
activo : boolean := false;
Indica si está o no activo.
indice_descriptor : *tipo_descriptor*;
Descriptor local que identifica a la cola global en el remoto donde se encuentra ésta.
nudo : *configuracion.tipo_nudo*;
Nudo en el que se encuentra la cola global.
red : *configuracion.tipo_red*;
Red por la que se accede a la cola global.

acceso : *tipos_de_colas.tipo_acceso*;
Modo de acceso que se permite sobre la cola a la que apunta.
suspendido : boolean;
Indica si la operación remota está suspendida.
identificador_tarea : dependiente de la implementación;
Identificador de la tarea propietaria del descriptor.
prioridad_tarea : dependiente de la implementación;
Prioridad de la tarea propietaria del descriptor.
end record;

Definición de la información asociada a un descriptor remoto, es decir, que identifica a una cola global en otro nudo.

type *tipo_cuerpo_descriptor_local* **is**
record
activo : boolean := false;
Indica si está o no activo.
indice_nombre : positive;
Indice que apunta a la cola de mensajes a la que identifica.
indice_descriptor : *tipo_descriptor*;
Descriptor remoto en el caso de que haya sido abierto por una operación remota.
acceso : *tipos_de_colas.tipo_acceso*;
Modo de acceso que se permite sobre la cola a la que apunta.
suspendido : boolean;
Indica si la operación realizada a través del descriptor está suspendida.
identificador_tarea : dependiente de la implementación;
Identificador de la tarea propietaria del descriptor.
prioridad_tarea : dependiente de la implementación;
Prioridad de la tarea propietaria del descriptor.
end record;

Definición de la información asociada a un descriptor local.

type *tipo_estado* **is** (*abierta*, *cerrada*);

Estado de la cola de mensajes.

type *tipo_elemento_lista* **is**
record
segmento : positive;
indice_en_segmento : positive;
Segmento en el que se localiza el mensajes y posición en el segmento.
longitud : integer;
Longitud del mensaje (menor que la máxima permitida en la cola a que corresponda).
siguiente_elemento : positive;
end record;

Elemento de una lista enlazada de mensajes de una cola de mensajes con localización en su segmento correspondiente.

type *tipo_cabeza_lista* **is**
record
cabeza : positive;
cola : positive;
numero_de_elementos : integer := 0;
end record;

Cabecera de una lista enlazada por índices con información del número de elementos que contiene.

type *tipo_lista_de_mensajes* **is array**
(1..*tipos_de_colas.max_numero_de_mensajes*)
of *tipo_elemento_lista*;

Lista con la información de la localización de los mensajes de una cola de mensajes en los segmentos de memoria.

type *tipo_lista_prioridades* **is array** (*tipos_de_colas.tipo_prioridad*)
of *tipo_cabeza_lista*;

Lista de mensajes encolados en una cola de mensajes, ordenada por las prioridades de los mismos.

type *tipo_cola_de_mensajes* **is**
record
abierta : boolean := false;
cola_en_destruccion : boolean;
Indicadores de cola abierta y en proceso de destrucción.
nombre : *configuracion.tipo_nombre*;
Nombre de la cola.
atributos : *tipos_de_colas.tipo_atributos*;
Atributos asignados a la cola.
numero_de_descriptores_abiertos : integer;
Número de descriptores apuntando a la cola.
numero_de_segmentos_reservados : positive;
lista_de_segmentos_reservados : *memoria.tipo_lista_de_segmentos*;
Número y lista de los segmentos de memoria reservados.
lista_de_mensajes : *tipo_lista_de_mensajes*;
Lista de índices de localización de mensajes en segmentos.
lista_prioridades : *tipo_lista_prioridades*;
lista_de_mensajes_libres : *tipo_cabeza_lista*;
Listas con los mensajes encolados por prioridad y los libres.
end record;

Definición de la información asociada a una cola de mensajes.

type *tipo_elemento_cola_de_espera* **is**
record
descriptor : *tipo_descriptor*;
Descriptor por el que se accede a la tarea u operación en espera.
red : *configuracion.tipo_red*;
Red de comunicación (solo para espera remota).
mensaje : positive;
Mensaje del espacio de mensajes (solo para espera remota).
siguiente : positive;
end record;

Definición del elemento de una lista enlazada para formar las colas de espera locales o remotas.

type *tipo_cola_de_espera* **is array** (tipo prioridad de las tareas)
of *tipo_cabeza_lista*;

Definición de la cola de espera local como una cola de prioridad. La prioridad que se utiliza es la que tienen asignadas las tareas de usuario y por lo tanto es dependiente de la implementación.

3.- Variables.

espacio_de_mensajes : **array** (1..*max_espacio_de_mensajes*)
of *tipo_espacio_de_mensajes*;

Reserva de memoria para los mensajes de comunicaciones.

espacio_de_memoria : **array** (1..*max_numero_de_segmentos*)
of *tipo_segmento*;

Memoria de la que disponen las colas de mensajes.

numero_de_segmentos_libres : integer := *max_numero_de_segmentos*;
lista_de_segmentos_libres : *tipo_lista_de_segmentos*;

Variables que configuran la lista de segmentos de memoria libres.

lista_descriptores_remotos : **array** (*tipo_descriptor_remoto*)
of *tipo_cuerpo_descriptor_remoto*;
lista_descriptores_locales : **array** (*tipo_descriptor_local*)
of *tipo_cuerpo_descriptor_local*;

Listas de descriptores locales y remotos.

error_remoto : *tipos_de_colas.tipo_error* := (*envia* => 0, *cierra* => 0);

Variable donde se apuntan los errores remotos que no se tramitan por el camino de la excepción.

numero_descriptores_locales_libres : integer := *tipo_descriptor_local*'**last**;

lista_descriptores_locales_libres : **array** (*tipo_descriptor_local*)
of *tipo_descriptor_local*;

Variables que configuran la lista de descriptores locales libres.

lista_de_colas : **array** (1..*tipos_de_colas.max_numero_de_colas*)
of *tipo_cola_de_mensajes*;

Colas de mensajes.

numero_de_colas_libres : integer := *tipos_de_colas.max_numero_de_colas*;
lista_de_colas_libres : **array** (1..*tipos_de_colas.max_numero_de_colas*)
of positive;

Variables que configuran la lista de colas de mensajes libres.

elementos_cola_de_espera : **array** (1..*max_elementos_cola*)
of *tipo_elemento_cola_de_espera*;
cola_de_espera_local : **array** (1..*tipos_de_colas.max_numero_de_colas*)
of *tipo_cola_de_espera*;
cola_de_espera_remota : **array** (1..*tipos_de_colas.max_numero_de_colas*)
of *tipo_cabeza_lista*;
numero_de_elementos_en_cola : **array**
(1..*tipos_de_colas.max_numero_de_colas*)
of integer := (**others** => 0);
libres_cola_de_espera : *tipo_cabeza_lista* :=
(*cabeza* => 1,
cola => *max_elementos_cola*,
numero_de_elementos => *max_elementos_cola*);

Variables que configuran las colas de espera, como un conjunto de *elementos de colas de espera*, y tres listas que los enlazan: cola de espera local, cola de espera remota y elementos libres.

4.- Excepciones.

nudo_no_configurado

Se produce cuando se realiza la operación *configuracion.obten_nudo* sobre una cola que no ha sido previamente configurada.

no_hay_espacio_de_memoria

Se intentan reservar más segmentos de memoria de los que se dispone.

no_hay_espacio_de_mensajes

Se intenta reservar un mensaje de comunicación pero no quedan libres.

cola_de_espera_completa

Sucede si alguna de las colas de espera termina con los elementos de cola de espera disponibles.

A.3. Operaciones sobre las colas de mensajes

En este apartado se muestran los pseudocódigos tanto de los procedimientos y funciones que implementan las operaciones que se pueden realizar sobre las colas de mensajes, como de las tareas de control de estas operaciones, y de la tarea de control del espacio de mensajes.

Atendiendo a la funcionalidad de los procedimientos, funciones y tareas que vamos a describir, podemos considerar la siguiente clasificación:

1.- Procedimientos de interfase a los programas de usuario.

Las especificaciones concretas de los procedimientos se encuentran en el paquete *colas_de_mensajes* del apartado A.1 de este mismo apéndice.

Procedure *colas_de_mensajes.abre (…)*

```
procedure abre (…) is

begin
   if operación local then
      if crear then
         Chequeo de excepciones:
            raise tipos_de_colas.numero_de_mensajes_fuera_de_rango;
            raise tipos_de_colas.
                  longitud_de_mensaje_fuera_de_rango;
         control_de_colas.sincroniza.crea(…);
         Establece descriptor;
      else
         control_de_colas.sincroniza.abre(…);
         Establece descriptor;
      end if;
   else
      Chequeo de excepciones:
         raise tipos_de_colas.cola_perteneciente_a_otro_nudo;
      control_de_colas.recursos_remotos.abre(…);
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Establece los parámetros del descriptor;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
      Suspende la tarea;
      Interpreta la respuesta;
      Libera el mensaje: memoria.libera_espacio_mensaje(…);
      if respuesta correcta then
         Establece descriptor local en el nudo remoto;
      else
         Eleva las excepciones:
            raise tipos_de_colas.no_hay_descriptores_libres;
            raise tipos_de_colas.cola_en_proceso_de_destruccion;
            raise tipos_de_colas.cola_no_existente;
            raise tipos_de_colas.problemas_de_comunicacion;
```

```
      end if;
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         control_de_colas.recursos_remotos.cierra(…);
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         control_de_colas.recursos_remotos.cierra(…);
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end abre;
```

Procedure *colas_de_mensajes.cierra (…)*

```
procedure cierra (…) is

begin
   if operación local then
      control_de_colas.sincroniza.cierra(…);
   else
      Chequeo de excepciones:
         raise tipos_de_colas.descriptor_no_activo;
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
      control_de_colas.recursos_remotos.cierra(…);
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end cierra;
```

Procedure *colas_de_mensajes.destruye (…)*

```
procedure destruye (…) is

begin
   control_de_colas.sincroniza.destruye(…);
end destruye;
```

Procedure *colas_de_mensajes.envia (…)*

```
procedure envia (…) is

begin
   if operación local then
```

```
      control_de_colas.sincroniza.envia(…);
   else
      Chequeo de excepciones:
         raise tipos_de_colas.descriptor_no_activo;
         raise tipos_de_colas.acceso_no_permitido;
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end envia;
```

Procedure *colas de mensajes.recibe (…)*

```
procedure recibe (…) is

begin
   if operación local then
      loop
         loop
            exit when no suspendido;
         end loop;
         control_de_colas.sincroniza.recibe(…);
         exit when hay mensaje;
      end loop;
   else
      Chequeo de excepciones:
         raise tipos_de_colas.descriptor_no_activo;
         raise tipos_de_colas.acceso_no_permitido;
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Establece los parámetros del descriptor;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
      Suspende la tarea;
      Interpreta la respuesta;
      Libera el mensaje: memoria.libera_espacio_mensaje(…);
      if respuesta correcta then
         Establece los parámetros que devuelve;
      else
         Eleva las excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.cola_en_proceso_de_destruccion;
            raise tipos_de_colas.problemas_de_comunicacion;
      end if;
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end recibe;
```

***Procedure** colas_de_mensajes.recibe_si_hay (…)*

```
procedure recibe_si_hay (…) is

begin
   if operación local then
      control_de_colas.sincroniza.recibe_si_hay(…);
   else
      Chequeo de excepciones:
         raise tipos_de_colas.descriptor_no_activo;
         raise tipos_de_colas.acceso_no_permitido;
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Establece los parámetros del descriptor;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
      Suspende la tarea;
      Interpreta la respuesta;
      Libera el mensaje: memoria.libera_espacio_mensaje(…);
      if respuesta correcta then
         Establece los parámetros que devuelve;
      else
         Eleva las excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.cola_en_proceso_de_destruccion;
            raise tipos_de_colas.problemas_de_comunicacion;
      end if;
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end recibe_si_hay;
```

***Procedure** colas_de_mensajes.coge_atributos (…)*

```
procedure coge_atributos (…) is

begin
   if operación local then
      control_de_colas.sincroniza.coge_atributos(…);
   else
      Chequeo de excepciones:
         raise tipos_de_colas.descriptor_no_activo;
      Reserva un mensaje: memoria.reserva_espacio_mensaje;
      Establece los parámetros del descriptor;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
      Suspende la tarea;
      Interpreta la respuesta;
      Libera el mensaje: memoria.libera_espacio_mensaje(…);
      if respuesta correcta then
         Establece los parámetros que devuelve;
      else
         Eleva las excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.problemas_de_comunicacion;
```

```
      end if;
   end if;
   exception
      when memoria.no_hay_espacio_de_mensajes =>
         raise tipos_de_colas.problemas_de_comunicacion;
      when red_transmision.no_hay_espacio_en_la_cola =>
         Libera el mensaje: memoria.libera_espacio_mensaje(…);
         raise tipos_de_colas.problemas_de_comunicacion;
end coge_atributos;
```

2.- Tarea de control del espacio de mensajes.

Los procedimientos que se ofrecen para utilizar el espacio de mensajes llaman a los puntos de entrada de esta tarea, localizada en el cuerpo del paquete *memoria*, que garantiza la exclusión mutua en el acceso a los datos que gestiona.

Task *maneja_recursos*

```
task maneja_recursos is
   pragma priority(24);
   entry libera_espacio_mensaje (mensaje : in positive);
   entry reserva_espacio_mensaje (mensaje : out positive);
end maneja_recursos;

task body maneja_recursos is

begin
   Inicializa lista de mensajes;
   loop
      select
         accept libera_espacio_mensaje (…) do
            Añade el elemento a la lista de libres;
         end libera_espacio_mensaje;
      or
         accept reserva_espacio_mensaje (…) do
            Chequeo de excepciones:
               raise no_hay_espacio_de_mensajes;
            Extrae mensaje de lista de libres;
         end reserva_espacio_mensaje;
      end select;
   end loop;
end maneja_recursos;
```

3.- Tarea de control de operaciones sobre recursos locales.

La especificación concreta de esta tarea se encuentra en el paquete *control_de_colas* del apartado A.1 de este mismo apéndice.

***Task** control_de_colas.sincroniza*

```
task body sincroniza is

begin
   Inicializa los descriptores locales;
   Inicializa las colas de mensajes;
   Inicializa las colas de espera;
   loop
      select
         accept ejecuta_remoto (…) do
            case operación is
               when abre =>
                  abre_remoto(…);
               when cierra =>
                  cierra_remoto(…);
               when envia =>
                  envia_remoto(…);
               when recibe =>
                  recibe_remoto(…);
               when recibe_si_hay =>
                  recibe_si_hay_remoto(…);
               when coge_atributos =>
                  coge_atributos_remoto(…);
            end case;
            exception
               when red_transmision.no_hay_espacio_en_la_cola =>
                  raise tipos_de_colas.problemas_de_comunicacion;
         end ejecuta_remoto;
      or
         accept crea (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.cola_ya_creada;
               raise tipos_de_colas.no_hay_colas_libres;
               raise tipos_de_colas.no_hay_descriptores_libres;
            Cálculo y reserva de memoria;
            Coge una cola de las libres y establece sus parámetros;
            Asigna el descriptor y establece sus parámetros;
            exception
               when memoria.no_hay_espacio_de_memoria =>
                  raise tipos_de_colas.problemas_de_comunicacion;
            end;
         end crea;
      or
         accept abre (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.cola_no_existente;
               raise tipos_de_colas.
                     cola_en_proceso_de_destruccion;
               raise tipos_de_colas.no_hay_descriptores_libres;
            Asigna el descriptor y establece sus parámetros;
         end abre;
      or
         accept cierra (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.descriptor_no_activo;
            Desactiva el descriptor y lo añade a los libres;
            if cola en proceso de destrucción y
               no hay descriptores abiertos then
               Libera la memoria;
               Destruye la cola y la añade a la lista de libres;
            end if;
         end cierra;
      or
```

```
      accept destruye (…) do
         Chequeo de excepciones:
            raise tipos_de_colas.cola_no_existente;
            raise tipos_de_colas.
                  cola_en_proceso_de_destruccion;
         if no hay descriptores abiertos then
            Libera la memoria;
            Destruye la cola y la añade a la lista de libres;
         else
            Anota el proceso de destrucción;
         end if;
      end destruye;
   or
      accept envia (…) do
         Chequeo de excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.
                  cola_en_proceso_de_destruccion;
            raise tipos_de_colas.
                        longitud_de_mensaje_fuera_de_rango;
            raise tipos_de_colas.acceso_no_permitido;
         if no hay operaciones en cola de espera remota then
            -- Se realiza la copia del mensaje.
            Chequeo de excepciones:
               raise tipos_de_colas.
                           insuficiente_espacio_en_la_cola;
            Introduce el mensaje en la cola en su prioridad;
         end if;
         if hay operaciones en cola de espera remota then
            Busca la primera operación en espera;
            Elabora el mensaje(ya estaba reservado);
            Establece los parámetros de transmisión;
            Transmite el mensaje: red_transmision.transmite(…);
         elsif hay operaciones en cola de espera local then
            Despierta a la tarea que corresponda;
         end if;
         exception
            when red_transmision.no_hay_espacio_en_la_cola =>
               raise tipos_de_colas.problemas_de_comunicacion;
      end envia;
   or
      accept recibe (…) do
         if suspendido then
            return;
         end if;
         Chequeo de excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.
                  cola_en_proceso_de_destruccion;
            raise tipos_de_colas.acceso_no_permitido;
         if no hay mensajes en la cola then
            Añade la tarea a la cola de espera local;
            Suspende la tarea;
         else
            Busca el mensaje de más prioridad;
            Lo devuelve junto con los demás parámetros;
         end if;
      end recibe;
   or
      accept recibe_si_hay (…) do
         Chequeo de excepciones:
            raise tipos_de_colas.descriptor_no_activo;
            raise tipos_de_colas.
                  cola_en_proceso_de_destruccion;
            raise tipos_de_colas.acceso_no_permitido;
```

```
            if no hay mensajes en la cola then
               Devuelve que no hay;
            else
               Busca el mensaje de más prioridad;
               Lo devuelve junto con los demás parámetros;
            end if;
         end recibe_si_hay;
      or
         accept coge_atributos (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.descriptor_no_activo;
            Devuelve los atributos de la cola;
         end coge_atributos;
      end select;
   end loop;
end sincroniza;
```

4.- Tarea de control de operaciones sobre recursos remotos.

La especificación concreta de esta tarea se encuentra en el paquete *control_de_colas* del apartado A.1 de este mismo apéndice.

Task *control_de_colas.recursos_remotos*

```
task body recursos_remotos is

begin
   Inicializa los descriptores remotos;
   loop
      select
         accept abre (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.no_hay_descriptores_libres;
            Establece el descriptor y sus parámetros;
         end abre;
      or
         accept cierra (…) do
            Chequeo de excepciones:
               raise tipos_de_colas.descriptor_no_activo;
            Añade el descriptor a la lista de libres;
         end cierra;
      end select;
   end loop;
end recursos_remotos;
```

5.- Procedimiento de encaminamiento de mensajes remotos.

La especificación concreta de este procedimiento se encuentra en el paquete *control_de_colas* del apartado A.1 de este mismo apéndice.

Procedure *control_de_colas.recibe (…)*

```
procedure recibe (…) is

begin
   if es una orden de operación then
      Ejecuta el mensaje:
         sincroniza.ejecuta_remoto(…);
   elsif respuesta a operación envia o cierra con error then
      Descifra y anota el error producido;
      Libera el mensaje: memoria.libera_espacio_mensaje(…);
   else
      Despierta a la tarea suspendida en
      ejecución de operación remota;
   end if;
   exception
      when tipos_de_colas.problemas_de_comunicacion =>
         Anota el error como interno;
end recibe;
```

6.- Procedimientos de ejecución de órdenes remotas.

Estos procedimientos se encuentran en el cuerpo del paquete *control_de_colas* cuya especificación está en el apartado A.1 de este mismo apéndice. Son todos los procedimientos encargados de la realización de las operaciones ordenadas por tareas de otros nudos sobre una cola de mensajes del propio nudo. Su ejecución la lleva a cabo la tarea *control_de_colas.sincroniza* a través del punto de entrada *ejecuta_remoto*.

Procedure *control_de_colas.abre_remoto (…)*

```
procedure abre_remoto (red     : in configuracion.tipo_red;
                       mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.cola_en_proceso_de_destruccion;
      raise tipos_de_colas.no_hay_descriptores_libres;
   Establece los parámetros del descriptor y lo activa;
   Elabora la respuesta;
   Transmite el mensaje: red_transmision.transmite(…);
   exception
      when tipos_de_colas.no_hay_descriptores_libres =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.cola_no_existente =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
```

```
      when tipos_de_colas.cola_en_proceso_de_destruccion =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end abre_remoto;
```

Procedure *control_de_colas.cierra_remoto (…)*

```
procedure cierra_remoto (red     : in configuracion.tipo_red;
                         mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.descriptor_no_activo;
   Desactiva el descriptor y lo añade a los libres;
   if cola en proceso de destrucción y
      no hay descriptores abiertos then
      Libera la memoria;
      Destruye la cola y la añade a la lista de libres;
   end if;
   Libera el mensaje: memoria.libera_espacio_mensaje(…);
   exception
      when tipos_de_colas.descriptor_no_activo =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end cierra_remoto;
```

Procedure *control_de_colas.envia_remoto (…)*

```
procedure envia_remoto (red     : in configuracion.tipo_red;
                        mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.descriptor_no_activo;
      raise tipos_de_colas.cola_en_proceso_de_destruccion;
      raise tipos_de_colas.longitud_de_mensaje_fuera_de_rango;
   if no hay operaciones en cola de espera remota then
      -- Se realiza la copia del mensaje.
      Chequeo de excepciones:
         raise tipos_de_colas.insuficiente_espacio_en_la_cola;
      Introduce el mensaje en la cola en su prioridad;
   end if;
   if hay operaciones en cola de espera remota then
      Busca la primera operación en espera;
      Elabora el mensaje:
         intercambia el mensaje con el recibe remoto en espera;
      Establece los parámetros de transmisión;
      Transmite el mensaje: red_transmision.transmite(…);
   elsif hay operaciones en cola de espera local then
      Despierta a la tarea que corresponda;
   end if;
   Libera el mensaje: memoria.libera_espacio_mensaje(…);
```

```
   exception
      when tipos_de_colas.descriptor_no_activo =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.cola_en_proceso_de_destruccion =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.longitud_de_mensaje_fuera_de_rango =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.insuficiente_espacio_en_la_cola =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end envia_remoto;
```

Procedure *control_de_colas.recibe_remoto (…)*

```
procedure recibe_remoto (red     : in configuracion.tipo_red;
                         mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.descriptor_no_activo;
      raise tipos_de_colas.cola_en_proceso_de_destruccion;
   if no hay mensajes en la cola then
      Añade la operación a la cola de espera remota;
   else
      Busca el mensaje de más prioridad;
      Elabora el mensaje;
      Transmite el mensaje: red_transmision.transmite(…);
   end if;
   exception
      when tipos_de_colas.descriptor_no_activo =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.cola_en_proceso_de_destruccion =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when cola_de_espera_completa =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end recibe_remoto;
```

Procedure *control_de_colas.recibe_si_hay_remoto (…)*

```
procedure recibe_si_hay_remoto (red    : in configuracion.tipo_red;
                                mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.descriptor_no_activo;
      raise tipos_de_colas.cola_en_proceso_de_destruccion;
```

```
   if no hay mensajes en la cola then
      Elabora el mensaje con no hay;
   else
      Busca el mensaje de más prioridad;
      Elabora el mensaje;
   end if;
   Transmite el mensaje: red_transmision.transmite(…);
   exception
      when tipos_de_colas.descriptor_no_activo =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
      when tipos_de_colas.cola_en_proceso_de_destruccion =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end recibe_si_hay_remoto;
```

Procedure *control_de_colas.coge_atributos_remoto (…)*

```
procedure coge_atributos_remoto
            (red     : in configuracion.tipo_red;
             mensaje : in positive) is

begin
   Establece parámetros de transmisión;
   Chequeo de excepciones:
      raise tipos_de_colas.descriptor_no_activo;
   Elabora el mensaje con los atributos;
   Transmite el mensaje: red_transmision.transmite(…);
   exception
      when tipos_de_colas.descriptor_no_activo =>
         Establece parámetros de transmisión;
         Transmite el mensaje: red_transmision.transmite(…);
end coge_atributos_remoto;
```

7.- Funciones de chequeo de error.

Funciones de chequeo de los errores producidos en las operaciones remotas *cierra* y *envía*, y en la recepción y ejecución de las operaciones remotas. Estas operaciones no gestionan mediante excepciones los errores producidos en el nudo remoto, sino que lo hacen a través de estas funciones. Estas funciones se encuentran en los cuerpos de los paquetes *colas_de_mensajes* (función de interfase al usuario), y *control_de_colas* (acceso a la información de error), cuyas especificaciones están en el apartado A.1 de este mismo apéndice.

Function *colas_de_mensajes.obten_error*

```
function obten_error return tipos_de_colas.tipo_error is


begin
   return control_de_colas.errores_remotos;
end obten_error;
```

Function *control_de_colas.errores_remotos*

```
function errores_remotos return tipos_de_colas.tipo_error is

begin
    Reinicializa el error;
    return errores producidos;
end errores_remotos;
```

A.4. Implementación de las comunicaciones

En este apartado del apéndice vamos a mostrar la estructura de datos y los pseudocódigos, de los procedimientos y tareas que forman parte de la implementación de las redes de comunicación. Las especificaciones de los procedimientos de interfase se recogen en el apartado A.1 de este apéndice. Teniendo en cuenta la definición de paquetes Ada realizada en la implementación de las comunicaciones, vamos a proceder a su descripción de acuerdo con los siguientes apartados:

1.- ***Red_transmision.***

Dispone del procedimiento que se encarga de encaminar los mensajes que se quieren transmitir hacia la red adecuada.

Procedure *red_transmision.transmite(…)*

```
procedure transmite (…) is

begin
   Establece la cabeza de red y los parámetros de transmisión;
   Encamina el mensaje a la red adecuada:
   case red is
      when xxx =>
```

```
            if configuracion.uso_servidor_esporadico then
               comunicaciones_transmision_xxx_ss.transmite(…);
            else
               comunicaciones_transmision_xxx.transmite(…);
            end if;
         ...
         when others => null;
      end case;
      exception
         when comunicaciones_transmision_xxx.
                              no_hay_espacio_en_la_cola =>
            raise no_hay_espacio_en_la_cola;
         ...
         when comunicaciones_transmision_xxx_ss.
                              no_hay_espacio_en_la_cola =>
            raise no_hay_espacio_en_la_cola;
         ...
   end transmite;
```

2.- *Red_recepcion*.

En este paquete Ada tenemos la tarea de control de mensajes de comunicación recibidos (internamente define una cola de prioridad), y la tarea que inicia la ejecución de estos mensajes. Antes de describir las tareas definimos la variable de control de ejecución de mensajes recibidos: *estado_ejecucion_activo* : boolean;

Task *red_recepcion.control_recibe*

```
task control_recibe is
   pragma priority(24);
   entry encola (red       : in configuracion.tipo_red;
                 mensaje   : in positive;
                 prioridad : in tipos_de_colas.tipo_prioridad);
   entry desencola (red     : out configuracion.tipo_red;
                    mensaje : out positive;
                    stop    : out boolean);
end control_recibe;

task body control_recibe is

Definición de la cola de mensajes de comunicación recibidos;

begin
   Inicializa cola de recepcion;
   loop
      select
         accept encola (…) do
            Chequeo de excepciones:
               raise no_hay_espacio_en_la_cola;
            Introduce el mensaje en la cola;
            if not estado_ejecucion_activo then
               estado_ejecucion_activo := true;
               ordena_recibe.ejecuta;
```

```
            end if;
         end encola;
      or
         accept desencola (…) do
            if hay que cambiar el estado then
               estado_ejecucion_activo := false;
               stop := true;
            else
               stop := false;
               Devuelve el mensaje de mas prioridad;
            end if;
         end desencola;
      end select;
   end loop;
end control_recibe;
```

Task *ordena_recibe*

```
task ordena_recibe is
   pragma priority(23);
   entry ejecuta;
end ordena_recibe;

task body ordena_recibe is
begin
   loop
      accept ejecuta;
      loop
         control_recibe.desencola(…);
         exit when stop;
         control_de_colas.recibe(…);
      end loop;
   end loop;
end ordena_recibe;
```

3.- *Comunicaciones_transmision_232/485.*

Vemos ahora la tarea de control de la cola de paquetes de comunicación para transmisión en líneas dedicadas punto a punto. La cola de prioridad se implementa dentro de esta tarea.

Task *sincroniza_transmite*

```
task sincroniza_transmite is
   pragma priority (24);
   entry transmite
            (mensaje   : in positive;
             prioridad : in tipos_de_colas.tipo_prioridad);
   entry orden_transmision;
end sincroniza_transmite;
```

```
task body sincroniza_transmite is

estado_transmitiendo : boolean := false;
Definición de la cola de paquetes de comunicación transmitidos;

begin
   Inicializa cola de transmisión;
   loop
      select
         accept transmite (…) do
            Chequeo de excepciones:
               raise no_hay_espacio_en_la_cola;
            if mensaje con más de un paquete
               or estado_transmitiendo then
               Encola el mensaje (partido en paquetes);
            end if;
            if not estado_transmitiendo then
               estado_transmitiendo := true;
               Copia el paquete en el buffer de transmisión;
               Establece nuevos parámetros del paquete o
                  libera el mensaje de comunicaciones;
               Inicia la transmisión (puerta serie);
            end if;
         end transmite;
      or
         accept orden_transmision do
            if no hay paquetes then
               estado_transmitiendo := false;
            else
               Establece el paquete de más prioridad y lo copia en
               el buffer de transmisión;
               Establece nuevos parámetros del paquete o
                  libera el mensaje de comunicaciones;
               Inicia la transmisión (puerta serie);
            end if;
         end orden_transmision;
      end select;
   end loop;
end sincroniza_transmite;
```

4.- *Comunicaciones_transmision_bus*.

Vemos aquí la tarea de control de la cola de paquetes de comunicación para transmisión en el bus VME, mostrando explícitamente la parte correspondiente al mecanismo de control de la comunicación a través del bus. La cola se implementa como una cola de prioridad y se declara en el interior de la tarea que la controla.

Task *sincroniza_transmite*

```
task sincroniza_transmite is
   pragma priority (24);
   entry transmite
            (mensaje   : in positive;
```

```
                  prioridad : in tipos_de_colas.tipo_prioridad);
      entry orden_transmision;
   end sincroniza_transmite;

   task body sincroniza_transmite is

   Definición de la cola de paquetes de comunicación transmitidos;
   function semaforo_libre (…) return boolean;

   begin
      Inicializa cola de transmisión;
      loop
         select
            accept transmite (…) do
               Chequeo de excepciones:
                  raise no_hay_espacio_en_la_cola;
               Bloquea el semáforo:
                  loop
                     exit when semaforo_libre(…);
                  end loop;
               if mensaje con más de un paquete
                  or hardware.cola_bus.estado_transmitiendo then
                  Encola el mensaje (partido en paquetes);
               end if;
               if not hardware.cola_bus.estado_transmitiendo then
                  Copia el paquete en el buffer compartido;
                  Establece nuevos parámetros del paquete o
                     libera el mensaje de comunicaciones;
                  Actualiza parámetros de hardware.cola_bus:
                     Actualiza si la cola del nudo actual está vacía;
                     hardware.cola_bus.estado_transmitiendo := true;
                     hardware.cola_bus.disparo_transmision := false;
                  Interrumpe al nudo adecuado;
                  Libera el semáforo;
               else
                  Libera el semáforo;
               end if;
            end transmite;
         or
            accept orden_transmision do
               Bloquea el semáforo:
                  loop
                     exit when semaforo_libre(…);
                  end loop;
               if hay_paquetes en algún nudo then
                  if el paquete pertenece al nudo actual then
                     Establece el paquete de más prioridad y lo copia
                        en el buffer compartido;
                     Establece nuevos parámetros del paquete o
                        libera el mensaje de comunicaciones;
                     Actualiza parámetros de hardware.cola_bus:
                        if cola nudo actual vacía then
                           Actualiza cola nudo actual vacía;
                        else
                           Actualiza cola nudo actual no vacía;
                           Establece paquete de más prioridad;
                        end if;
                       hardware.cola_bus.disparo_transmision := false;
                     Interrumpe al nudo adecuado;
                     Libera el semáforo;
                  else
                     hardware.cola_bus.disparo_transmision := true;
                     Pasa el control al nudo adecuado;
                     Libera el semáforo;
                  end if;
```

```
            else
               hardware.cola_bus.estado_transmitiendo := false;
               Libera el semáforo;
            end if;
         end orden_transmision;
      end select;
   end loop;
end sincroniza_transmite;
```

5.- *Comunicaciones_transmision_232/485_ss.*

En este paquete tenemos la tarea de control de la cola de paquetes de comunicación para transmisión en líneas dedicadas punto a punto, cuando se utiliza el servidor esporádico de comunicaciones. La cola de prioridad se encuentra en el paquete Ada que contiene al servidor esporádico.

Task *sincroniza_transmite*

```
task sincroniza_transmite is
   pragma priority (24);
   entry transmite
            (mensaje   : in positive;
             prioridad : in tipos_de_colas.tipo_prioridad);
   entry orden_transmision;
end sincroniza_transmite;

task body sincroniza_transmite is

estado_transmitiendo : boolean := false;

begin
   Inicializa cola de transmisión;
   loop
      select
         accept transmite (…) do
            Chequeo de excepciones:
               if no hay espacio disponible en la cola del SEC then
                  raise no_hay_espacio_en_la_cola;
               end if;
            Encola el mensaje:
               for i in número de paquetes
               loop
                  ss_comunicaciones_232/485.introduce(…);
               end loop;
            if not estado_transmitiendo then
               estado_transmitiendo := true;
               ss_comunicaciones_232/485.extrae(…);
               Copia el paquete en el buffer de transmisión;
               Establece nuevos parámetros del paquete o
                  libera el mensaje de comunicaciones;
               Inicia la transmisión (puerta serie);
            end if;
         end transmite;
      or
         accept orden_transmision do
```

```
            if ss_comunicaciones_232/485.cola_vacia then
               estado_transmitiendo := false;
            else
               ss_comunicaciones_232/485.extrae(…);
               Copia el paquete en el buffer de transmisión;
               Establece nuevos parámetros del paquete o
                  libera el mensaje de comunicaciones;
               Inicia la transmisión (puerta serie);
            end if;
         end orden_transmision;
      end select;
   end loop;
end sincroniza_transmite;
```

6.- *Comunicaciones_transmision_bus_ss.*

Este paquete contiene la tarea de control de la transmisión de paquetes de comunicación en el bus VME cuando se utiliza el servidor esporádico para comunicaciones. Al igual que en las líneas punto a punto, la cola de prioridad se encuentra en el paquete Ada que contiene al servidor esporádico.

Task *sincroniza_transmite*

```
task sincroniza_transmite is
   pragma priority (24);
   entry transmite
              (mensaje   : in positive;
               prioridad : in tipos_de_colas.tipo_prioridad);
   entry orden_transmision;
end sincroniza_transmite;

task body sincroniza_transmite is

function semaforo_libre (…) return boolean;

begin
   Inicializa cola de transmisión;
   loop
      select
         accept transmite (…) do
            Chequeo de excepciones:
               if no hay espacio disponible en la cola del SEC then
                  raise no_hay_espacio_en_la_cola;
               end if;
            Encola el mensaje:
               for i in número de paquetes
               loop
                  ss_comunicaciones_bus.introduce(…);
               end loop;
            Bloquea el semáforo:
               loop
                  exit when semaforo_libre(…);
               end loop;
            if not hardware.cola_bus.estado_transmitiendo then
               ss_comunicaciones_bus.extrae(…);
```

```
                Copia el paquete en el buffer compartido;
                Establece nuevos parámetros del paquete o
                   libera el mensaje de comunicaciones;
                Actualiza parámetros de hardware.cola_bus:
                   Actualiza si la cola del nudo actual está vacía;
                   Establece nueva prioridad y modo a partir de
                      ss_comunicaciones_bus.estado(…);
                   hardware.cola_bus.estado_transmitiendo := true;
                   hardware.cola_bus.disparo_transmision := false;
                Interrumpe al nudo adecuado;
                Libera el semáforo;
             else
                Establece nueva prioridad y modo a partir de
                   ss_comunicaciones_bus.estado(…);
                Libera el semáforo;
             end if;
          end transmite;
       or
          accept orden_transmision do
             Bloquea el semáforo:
                loop
                   exit when semaforo_libre(…);
                end loop;
             if hay_paquetes en algún nudo then
                if el paquete pertenece al nudo actual then
                   ss_comunicaciones_bus.extrae(…);
                   Copia el paquete en el buffer compartido;
                   Establece nuevos parámetros del paquete o
                      libera el mensaje de comunicaciones;
                   Actualiza parámetros de hardware.cola_bus:
                      if cola nudo actual vacía then
                         Actualiza cola nudo actual vacía;
                      else
                         Actualiza cola nudo actual no vacía;
                         Establece nueva prioridad y modo a partir
                            de ss_comunicaciones_bus.estado(…);
                         Establece paquete de más prioridad;
                      end if;
                      hardware.cola_bus.disparo_transmision := false;
                   Interrumpe al nudo adecuado;
                   Libera el semáforo;
                else
                   hardware.cola_bus.disparo_transmision := true;
                   Pasa el control al nudo adecuado;
                   Libera el semáforo;
                end if;
             else
                hardware.cola_bus.estado_transmitiendo := false;
                Libera el semáforo;
             end if;
          end orden_transmision;
       end select;
    end loop;
 end sincroniza_transmite;
```

7.- *Comunicaciones_recepcion_xxx.*

Los paquetes Ada en los que se implementa la recepción en las comunicaciones mantienen la misma estructura de datos y de gestión de estos

datos independientemente de la red a la que sirvan. A continuación mostramos los datos que se utilizan y el pseudocódigo del procedimiento que realiza la recepción de paquetes de comunicación.

identificador_de_red : **constant** *configuracion.tipo_red*;

Añade este identificador al mensaje recibido.

mapa_de_recepcion : **array** (*configuracion.tipo_nudo*,

1..*memoria.max_espacio_de_mensajes*) **of** positive;

Localización de los mensajes recibidos para su reconstrucción a partir de los paquetes de comunicación.

buffer : dependiente de la red y de la implementación;

Espacio para la recepción de un paquete de comunicación.

Procedure *comunicaciones_recepcion_xxx.recibe(…)*

```
procedure recibe is

begin
   Establece el mensaje al que añade el paquete recibido;
   if se ha completado un mensaje then
      Se lanza a la red de recepción para procesarlo:
         red_recepcion.encola(…);
   end if;
   exception
      when red_recepcion.no_hay_espacio_en_la_cola =>
         Anota el error como interno;
end recibe;
```

8.- *Comunicaciones*.

Vamos a mostrar las tareas de interrupción, que se encargan de canalizar los paquetes de comunicación recibidos hacia los procedimientos de las comunicaciones de recepción.

Task *interrupcion_nivel_5*

```
task interrupcion_nivel_5 is
   pragma priority (24);
   entry interrupcion (estado : in short_integer);
   for interrupcion use at dependiente de la implementación;
end interrupcion_nivel_5;
```

```
task body interrupcion_nivel_5 is
begin
   loop
      accept interrupcion (estado : in short_integer) do
         if recepción RS-485 then
            comunicaciones_recepcion_485.recibe;
         end if;
         if recepción RS-232 then
            comunicaciones_recepcion_232.recibe;
         end if;
         if transmisión RS-485 then
            if configuracion.uso_servidor_esporadico then
               comunicaciones_transmision_485_ss.orden_transmision;
            else
               comunicaciones_transmision_485.orden_transmision;
            end if;
         end if;
         if transmisión RS-232 then
            if configuracion.uso_servidor_esporadico then
               comunicaciones_transmision_232_ss.orden_transmision;
            else
               comunicaciones_transmision_232.orden_transmision;
            end if;
         end if;
      end interrupcion;
   end loop;
end interrupcion_nivel_5;
```

Task *interrupcion_nivel_7*

```
task interrupcion_nivel_7 is
   pragma priority (24);
   entry interrupcion;
   for interrupcion use at dependiente de la implementación;
end interrupcion_nivel_7;

task body interrupcion_nivel_7 is

begin
   loop
      accept interrupcion  do
         if not hardware.cola_bus.disparo_transmision then
            comunicaciones_recepcion_bus.recibe;
         end if;
         if configuracion.uso_servidor_esporadico then
            comunicaciones_transmision_bus_ss.orden_transmision;
         else
            comunicaciones_transmision_bus.orden_transmision;
         end if;
      end interrupcion;
   end loop;
end interrupcion_nivel_7;
```

9.- *Hardware*.

En este paquete Ada se encuentra el procedimiento que se suministra como parámetro genérico *cambio_a_normal(…)* al paquete Ada genérico *SS_distribuido* en su instanciación para crear *ss_comunicaciones_bus*.

Procedure *hardware.marca_cambio_a_normal (…)*

```
procedure marca_cambio_a_normal
          (prioridad : in tipos_de_colas.tipo_prioridad) is

begin
   Establece cambio a normal en la cola del bus:
      if hardware.cola_bus.lista_procesadores
         (nudo actual).prioridad_normal then
         if hardware.cola_bus.lista_procesadores
            (nudo actual).prioridad < prioridad then
            hardware.cola_bus.lista_procesadores
               (nudo actual).prioridad := prioridad;
         end if;
      else
         hardware.cola_bus.lista_procesadores
            (nudo actual).prioridad := prioridad;
         hardware.cola_bus.lista_procesadores
            (nudo actual).prioridad_normal := true;
      end if;
end marca_cambio_a_normal;
```

B. Terminología del modelado de las redes de comunicación

B.1. Lista de términos y significados

l_{atrib}	Longitud del mensaje correspondiente a la codificación de los atributos de las colas de mensajes para su transmisión.
l_{cab}	Longitud del protocolo de comunicaciones para la partición en paquetes de los mensajes de comunicación.
l_m	Longitud del mensaje propio de las colas de mensajes, o de la información adicional que necesitan algunas operaciones (como *abre* y *coge atributos*).
l_{nom}	Longitud del mensaje correspondiente a la codificación del nombre de las colas de mensajes para su transmisión.
l_p	Longitud del paquete de comunicaciones.
l_{pro}	Longitud del protocolo de comunicaciones en el que se codifican las operaciones sobre las colas de mensajes.
N_{busca_cola}	Número de colas recorridas en la búsqueda de la cola sobre la que se quiere abrir un descriptor hasta encontrar la que se busca. El peor caso es N_{colas}.

$N_{busca_mensaje}$	Número de prioridades recorridas (desde la máxima) hasta encontrar el mensaje de más prioridad en la cola de mensajes sobre la que se realiza una operación *recibe*.
N_{busca_tarea}	Número de prioridades recorridas (desde la máxima) hasta encontrar la tarea de más prioridad en la cola de espera local.
N_{colas}	Número máximo de colas que admite el sistema.
$N_{desencola}$	Número de rellenos desencolados y ejecutados.
N_{elem_conf}	Número de elementos configurados recorridos hasta encontrar el que se busca.
$N_{intentos}$	Número de intentos de recibir mensaje en una operación local.
$N_{mensajes}$	Número de mensajes con el que se crea la cola.
N_n	Número de nudos conectados al bus VME.
N_p	Número de prioridades recorridas (desde la máxima) hasta la del paquete en disposición de ser transmitido.
$N_{paquetes}$	Número de paquetes en los que se divide un mensaje para su transmisión.
N_r	Número de rellenos encolados en el instante de la llamada a una operación sobre el tipo de cola con el que se implementa la cola de rellenos.
$N_{seg_liberados}$	Número de segmentos de memoria liberados.
$N_{seg_reservados}$	Número de segmentos de memoria reservados.

p_{intro}	Número de paquetes que se introducen en la cola del servidor esporádico para comunicaciones.
t_{abre_rem}	Tiempo de ejecución de la acción *abre_rem*.
t_{busca}	Tiempo de búsqueda del paquete de mayor prioridad para el bus VME cuando se utiliza el servidor esporádico para comunicaciones.
t_{cierra_rem}	Tiempo de ejecución de la acción *cierra_rem*.
t_{colas_x}	Tiempos de ejecución de los tramos de la acción *colas* para las distintas operaciones sobre las colas de mensajes (x = i, f, m, m1, m2, m3).
t_{com_tr}	Tiempo de ejecución de la acción *com_tr* cuando no se utiliza el servidor esporádico para comunicaciones.
$t_{com_tr_ss}$	Tiempo de ejecución de la acción *com_tr* cuando se utiliza el servidor esporádico para comunicaciones.
$t_{consulta}$	Tiempo total de consulta de elemento a desencolar de la cola de rellenos.
$t_{control}$	Tiempo de ejecución de la acción *control* completa para las distintas operaciones sobre las colas de mensajes.
$t_{control_x}$	Tiempos de ejecución de los tramos de la acción *control* para las distintas operaciones sobre las colas de mensajes (x = i, f).
$t_{dequeue}$	Tiempo de desencolado del elemento de mayor prioridad para el tipo de cola con el que se implementa la cola de rellenos.

$t_{desencola}$	Tiempo total de la operación de desencolado y ejecución de un elemento de la cola de rellenos.
$t_{desencola_rec}$	Tiempo de ejecución de la acción *desencola_rec*.
$t_{ejec_rellenos}$	Tiempo de ejecución de rellenos en el servidor esporádico para comunicaciones.
$t_{ejec_rellenos_temp}$	Tiempo de ejecución de rellenos por control temporal en el servidor esporádico para comunicaciones.
t_{em}	Tiempo consumido en el encolado del mensaje partido en paquetes en la cola de transmisión. Se calcula a partir de los siguientes tiempos: $t_{red_tr_i}$, $t_{red_tr_f}$ y t_{com_tr}.
t_{em_ss}	Tiempo consumido en el encolado del mensaje partido en paquetes en la cola del servidor. Se calcula a partir de los siguientes tiempos: $t_{red_tr_i}$, $t_{red_tr_f}$ y $t_{com_tr_ss}$.
t_{encola_rec}	Tiempo de ejecución de la acción *encola_rec*.
$t_{enqueue}$	Tiempo de encolado de un elemento en el tipo de cola con el que se implementa la cola de rellenos.
t_{estado}	Tiempo de consulta del estado para el servidor esporádico de comunicaciones distribuido.
t_{extrae}	Tiempo de extracción de un elemento de la cola del servidor esporádico para comunicaciones.
t_{int_x}	Tiempos de ejecución de los tramos correspondientes a las acciones *interrupción nivel 5* e *interrupción nivel 7* (x = i, f).

t_{intro}	Tiempo de introducción de un elemento en la cola del servidor esporádico para comunicaciones.
t_{iter}	Tiempo de iteración para la búsqueda del elemento de mayor prioridad de la cola del servidor esporádico para comunicaciones.
t_{libera_men}	Tiempo de ejecución de la acción *libera_men*.
t_{libera_seg}	Tiempo de liberación de segmentos de memoria (procedimiento *libera_segmentos* del paquete Ada *memoria*).
t_{nudo}	Tiempo de consulta del nudo asociado a una cola de mensajes en la configuración del sistema (función *obten_nudo* del paquete Ada *configuracion*).
t_{nudo_actual}	Tiempo de consulta del nudo actual (función *obten_nudo_actual* del paquete Ada *configuracion*).
$t_{ordena_rec_x}$	Tiempos de ejecución de los tramos correspondientes a la acción *ordena_rec* (x = i, m, f).
t_{ordena_tr}	Tiempo de ejecución de la acción *ordena_tr* cuando no se utiliza el servidor esporádico para comunicaciones.
$t_{ordena_tr_ss}$	Tiempo de ejecución de la acción *ordena_tr* cuando se utiliza el servidor esporádico para comunicaciones.
$t_{paquete}$	Tiempo de transmisión de un paquete de comunicaciones.
$t_{plan_relleno}$	Tiempo de planificación de rellenos en el servidor esporádico para comunicaciones.

t_{read} Tiempo de lectura del elemento de mayor prioridad para el tipo de cola con el que se implementa la cola de rellenos.

t_{red} Tiempo de consulta de la red asociada a una cola de mensajes en la configuración del sistema (función *obten_red* del paquete Ada *configuracion*).

$t_{red_tr_x}$ Tiempos de ejecución de los tramos de acción correspondientes a *red_tr* (x = i, f).

$t_{reserva_men}$ Tiempo de ejecución de la acción *reserva_men*.

$t_{reserva_seg}$ Reserva de segmentos de memoria (procedimiento *reserva_segmentos* del paquete Ada *memoria*).

$t_{sin_ejec_x}$ Tiempos de ejecución de los tramos de acción correspondientes a la llamada y retorno de la tarea *sincroniza* (paquete Ada *control_de_colas*) para la ejecución de operaciones remotas en el nudo remoto (x = i, f).

$t_{tarea_int_x}$ Tiempos de ejecución de los tramos correspondientes a las acciones *tarea interrupción 5 / 5* - y *tarea interrupción 7 / 7* - (x = i, f).

t_{tick} Tiempo de tick utilizado por el planificador del Ada.

t_{tp} Tiempo empleado para la transmisión de los paquetes que componen el mensaje. Se calcula a partir de los tiempos siguientes: t_{int_i}, t_{int_f}, $t_{tarea_int_i}$, $t_{tarea_int_f}$ y t_{ord_tr}.

t_{tp_ss} Tiempo empleado para la transmisión de los paquetes que componen el mensaje. Se calcula a partir de los tiempos siguientes: t_{int_i}, t_{int_f}, $t_{tarea_int_i}$, $t_{tarea_int_f}$ y $t_{ord_tr_ss}$.

t_{trans} Tiempo consumido en la transmisión de un mensaje sin servidor esporádico.

t_{trans_ss} Tiempo consumido en la transmisión de un mensaje con servidor esporádico.

B.2. Correspondencias de las acciones

Abre_rem Acción de apertura de un descriptor remoto realizada por la tarea *recursos_remotos* del paquete Ada *control_de_colas*.

Cierra_rem Acción de cierre de un descriptor remoto realizada por la tarea *recursos_remotos* del paquete Ada *control_de_colas*.

Colas Tramo de acción correspondiente a los procedimientos del paquete Ada *colas_de_mensajes* ejecutados por la tarea de usuario al realizar una operación sobre las colas de mensajes.

Com_tr Acción de encolado de paquetes en la cola de transmisión. La lleva a cabo la tarea *sincroniza_transmite* de los paquetes Ada *comunicaciones_transmision_xxx* o *comunicaciones_transmision_xxx_ss*.

Conecta_red_tr Secuencia de acciones derivadas del uso del procedimiento de transmisión de mensajes por la red.

Control Acción que se corresponde con el código que posee la tarea *sincroniza* del paquete Ada *control_de_colas* para cada operación sobre las colas de mensajes.

Control (ejecuta_remoto) Acción *control* en ejecución de una operación remota en el nudo remoto.

Desencola_rec	Acción de sacar un mensaje de la cola de recepción realizada por la tarea *control_recibe* del paquete *red_recepcion*.
Ejecuta	Secuencia de acciones correspondientes a la ejecución de las operaciones remotas en el nudo remoto.
Encola_rec	Acción de encolado de un mensaje de comunicaciones (cuando está completo) en la cola de recepción. Esta acción la realiza la tarea *control_recibe* del paquete Ada *red_recepcion*.
Interrupción nivel 5	Acción del procedimiento de atención de interrupción de las líneas serie para la transmisión o recepción de un caracter o de un paquete de comunicaciones.
Interrupción nivel 7	Acción del procedimiento de atención de interrupción del bus VME para la transmisión o recepción de un nuevo paquete de comunicaciones o para el cambio en el control de la transmisión.
Libera_men	Acción de liberación de un mensaje, controlada por la tarea *maneja_recursos* del paquete Ada *memoria*.
Ordena_rec	Acción de ejecución de las operaciones remotas o de sus respuestas en el nudo remoto. Se corresponde con la tarea *ordena_recibe* del paquete Ada *red_recepcion*.
Ordena_tr	Acción de extracción de un paquete de comunicaciones de la cola de transmisión efectuada por la tarea *sincroniza_transmite* de los paquetes Ada *comunicaciones_transmision_xxx* y *comunicaciones_transmision_xxx_ss*. En el bus VME puede realizar el cambio en el control de la transmisión.

Recepción — Acciones correspondientes a la recepción de paquetes de comunicación a través de las redes.

Red_tr — Tramo de acción correspondiente al procedimiento *transmite* del paquete *red_transmision*. Este código puede ser ejecutado por *colas*, es decir, por la tarea de usuario, o por *control* (en la generación de la respuesta a la operación).

Reserva_men — Acción de reserva de un mensaje, controlada por la tarea *maneja_recursos* del paquete Ada *memoria*.

Tarea interrupción 5 — Acción que realiza la tarea de interrupción *interrupcion_nivel_5* del paquete Ada *comunicaciones* cuando es llamada para la transmisión de un nuevo paquete de comunicaciones.

Tarea interrupción 5 - — *Procedimiento recibe.* Acción que realiza la tarea de interrupción *interrupcion_nivel_5* del paquete Ada *comunicaciones* cuando es llamada a causa de la recepción de un paquete de comunicaciones completo. Esta tarea ejecuta el procedimiento *recibe* de los paquetes Ada *comunicaciones_recepcion_xxx.*

Tarea interrupción 7 — Acción que realiza la tarea de interrupción *interrupcion_nivel_7* del paquete Ada *comunicaciones* cuando es llamada para la transmisión de un nuevo paquete de comunicaciones o la transferencia del control de transmisión en el bus VME.

Tarea interrupción 7 - — *Procedimiento recibe.* Acción que realiza la tarea de interrupción *interrupcion_nivel_7* del paquete Ada *comunicaciones* cuando es llamada a causa de la recepción de un paquete de comunicaciones completo. Esta tarea, al igual que en las líneas serie, ejecuta el procedimiento *recibe* de los paquetes Ada *comunicaciones_recepcion_xxx.*

Transmisión	Acciones correspondientes a la transmisión de paquetes de comunicación a través de las redes.

Bibliografía

[ADA83] "Reference Manual for the ADA Programming Language". ANSI/MIL-STD 1815 A.

[ADA93] "Ada 9X Rationale". Ada 9X Project, Department of Defense, USA, Draft 4.0, September 1993.

[ADA95] International Standard ISO/IEC 8652:1995(E): "Information technology - Programming languajes - Ada. Ada Reference Manual".1995.

[AGR91] AGRAWAL G., CHEN B., ZHAO W., and DAVARI S.: "Architecture Impact of FDDI Network on Scheduling Hard Real-Time Traffic". Workshop on Architectural Aspects of Real-Time Systems, December 1991.

[ALM90] AL-MOUHAMED M.A.: "Lower Bound on the Number of Processors and Time for Scheduling Precedence Graphs With Communication Costs". IEEE Transactions on Software Engineering, vol. 16, no. 12, pp.1390-1401, December 1990.

[ARA94] ARAS C.M., KUROSE J.F., REEVES D.S. y SCHULZRINNE H.: "Real-Time Communications in Packet-Switched Networks". Proceedings of the IEEE, Vol. 82, No. 1, January 1994.

[AUD91A] AUDSLEY N.C.: "Deadline Monotonic Scheduling". Department of Computer Science, University of York, Technical Report YCS_146, September 1991.

[AUD91B] AUDSLEY N.C.: "Optimal Priority Assignment and Feasibility of Static Priority Tasks With Arbitrary Start Times". Department of Computer Science, University of York, Technical Report YCS_164, December 1991.

[AUD91C] AUDSLEY N.C., BURNS A., RICHARDSON M.F. y WELLINGS A.J.: "Hard Real-Time Scheduling: The Deadline Monotonic Approach". Proceedings 8th IEEE Workshop on Real-Time Operating Systems and Software, Atlanta, May 1991.

[AUD93] AUDSLEY N.C., BURNS A., RICHARDSON M., TINDELL K. y WELLINGS A.J.: "Applying New Scheduling Theory to Static Priority Pre-emptive Scheduling". Software Engineering Journal, Vol. 8, No. 5, pp. 284-292, September 1993.

[AUD94A] AUDSLEY N., BURNS A., RICHARDSON M. t WELLINGS A.: "STRESS: A Simulator for Hard Real-Time Systems". Software Practice and Experience, July 1994.

[AUD94B] AUDSLEY N.C., BURNS A., DAVIS R.I. y WELLINGS A.J.: "Integrating Best Effort and Fixed Priority Scheduling". Proceedings of the 1994 Workshop on Real-Time Programming, Lake Constance, Germany, June 1994.

[BAI93] BAILEY C.M., FYFE E., VARDANEGA T. y WELLINGS A.J.: "The Use of Preemptive Priority-Based Scheduling for Space Applications". Proceedings Real-Time Systems Symposium, December 1993.

[BAI94] BAILEY C.M., BURNS A., WELLINGS A.J. y FORSYTH C. H.: "A Performance Analysis of a Hard Real-Time System". Department of Computer Science, University of York, Technical Report YCS_224, February 1994.

[BAK86] BAKER T.P. y SCALLON G.M.: "An Architecture for Real-Time Software Systems". IEEE Software, vol 3, no. 3, pp. 50-59, May 1986.

[BAK88] BAKER T.P., SHAW A.: "The Cyclic Executive Model and Ada". Proceedings of the IEEE Real-Time Systems Symposium, December 1988.

[BAK90] BAKER T.P.: "A Stack-Based Resource Allocation Policy for Realtime Processes". IEEE, CH2933-0, pp. 191-200, 1990.

[BAK91A] BAKER T.P. y RICCARDI G.A.: "Ada Tasking: From Semantics to Efficient Implementation". IEEE Software, pp. 34-46, March 1991.

[BAK91B] BAKER T.P.: "Stack-Based Scheduling of Realtime Processes". The Journal of Real-Time Systems, Kluwer Academic Publishers, 3, pp. 67-99, 1991.

[BLA91] BLAKE B.A. y SCHWAN K.: "Experimental Evaluation of Real-Time Scheduler for a Multiprocessor System". IEEE Transaction on Software Engineering, vol. 17, no. 1, pp. 34-44, January 1991.

[BOO87] BOOCH G.: "Software Components with Ada. Structures, Tools and Subsystems". The Benjamin/Cummings Publishing Company, Inc, 1987.

[BOR89A] BORGER M.W., KLEIN M.H. y VELTRE R.A.: "Real-Time Software Engineering in Ada: Observations and Guidelines". Carnegie Mellon University, Sofware Engineering Institute, Tech. Rept. CMU/SEI-89-TR-22, ESD-89-TR-30, September 1989.

[BOR89B] BORGER M.W. y RAJKUMAR R.: "Implementing Priority Inheritance Algorithms in an Ada Runtime System". Carnegie Mellon University, Sofware Engineering Institute, Tech. Rept. CMU/SEI-89-TR-15, ESD-89-TR-23, April 1989.

[BUR89] BURNS A. y WELLINGS A.J.: "Real-Time Ada: Outstanding Problem Areas". Third International Real Time Ada Workshop, Conference Report. Ada User Vol.10, no. 3, pp.141-142, July 1989.

[BUR90] BURNS A. y WELLING A.J.: "Real-Time Systems and Their Programming Languages". Addison Wesley Publishing Company, 1990.

[BUR91] BURNS A. y WELLINGS A.J.: "Priority Inheritance and Message Passing Communicaton: A Formal Treatment". The Journal of Real-Time Systems, Kluwer Academic Publishers, 3, pp. 19-44, 1991.

[BUR92] BURNS A. y WELLINGS A.J.: "Designing Hard Real-Time Systems". Proceedings 11th Ada Europe conference (1992), Springer-Verlag Lecture Notes in Computer Science 603.

[BUR93A] BURNS A.: "Preemptive Priority Based Scheduling: An Appropriate Engineering Approach". Department of Computer Science, University of York, Technical Report YCS_214, November 1993.

[BUR93B] BURNS A., NICHOLSON M., TINDELL K. y ZHANG N.: "Allocating and Scheduling Hard Real-Time Tasks on a Point-to-Point Distributed System". Proceedings of the Workshop on Parallel and Distributed Real-Time Systems, pp. 11-20, April 13-15, 1993.

[BUR93C] BURNS A. y WELLINGS A.J.: "Implementing Analysable Hard Real-time Sporadic Tasks in Ada 9X". Department of Computer Science, University of York, Technical Report YCS_209, September 1993.

[BUR94A] BURNS A. y WELLINGS A.J.: "HRT-HOOD: A Structured Design Method for Hard Real-Time Systems". Real-Time Systems, 6, pp. 73-114, 1994.

[BUR94B] BURNS A., TINDELL K. y WELLINGS A.J.: "Fixed Priority Scheduling with Deadlines Prior to Completion". Proceedings Sixth Euromicro workshop on Real-Time Systems, Vaesteraas, Sweden, pp. 138-142, June 1994.

[COO83] COOLAHAN J.E. y ROUSSOPOULOS N.: "Timing Requirements for Time-Driven Systems Using Augmented Petri Nets". IEEE Trans. on Software Eng. Vol. SE-9, No. 5, pp. 603-616 , September 1983.

[COO85] COOLAHAN J.E. y ROUSSOPOULOS N.: "A Timed Petri Net Methodology for Specifying Real-Time System Timing Requirements". International Workshop on Timed Petri Nets. IEEE Catalog number 85CH2187-3, pp. 24-31, July 1985.

[CHA93] CHAPMAN R., BURNS A. y WELLINGS A.J.: "Worst-case Timing Analysis of Exception Handling in Ada". Proceedings Ada UK conference, London, 1993.

[CHA94] CHAPMAN R., BURNS A. y WELLINGS A.J.: "Integrated Program Proof and Worst-Case Timing Analysis of SPARK Ada". Proceedings 1994 ACM Workshop on Language, Compiler, and Tool Support for Real-Time Systems, Florida, June 1994.

[CHE90] CHEN M. y LIN K.: "Dynamic Priority Ceilings: A Concurrency Control Protocol for Real-Time Systems". Real-Time Systems, Kluwer Academic Publishers, 2, pp. 325-346, 1990.

[CHT90] CHETTO H., SILLY M. y BOUCHENTOUF T.: "Dynamic Scheduling of Real-Time Tasks under Precedence Constraints". Real-Time Systems, vol. 2, no. 3, pp. 181-194, September 1990.

[DAV86] DAVARI S. y DHALL S.K.: "An on Line Algorithm for Real-Time Tasks Allocation". IEEE, CH2351-5, 1986. Procedente: Real-Time System Symphosium, December 2-4, 1980.

[DVI93A] DAVIS R.I.: "Scheduling Slack Time in Fixed Priority Pre-emptive Systems". Department of Computer Science, University of York, Technical Report YCS_216, November 1993.

[DVI93B] DAVIS R.I.: "Approximate Slack Stealing Algorithms for Fixed Priority Pre-emptive Systems". Department of Computer Science, University of York, Technical Report YCS_217, November 1993.

[DVI94A] DAVIS R.I.: "Dual Priority Scheduling: A Means of Providing Flexibility in Hard Real-time Systems". Department of Computer Science, University of York, Technical Report YCS_230, May 1994.

[DVI94B] DAVIS R.I.: "Guaranteeing X in Y: On-line Acceptance Tests for Hard Aperiodic Tasks Scheduled by the Slack Stealing Algorithm". Department of Computer Science, University of York, Technical Report YCS_231, May 1994.

[GAF91] GAFFORD J.D.: "Rate Monotonic Scheduling". IEEE Micro, pp. 34-38 y 102, June 1991.

[GEH91] GEHANI N.: "Ada Concurrent Programming". Prentice Hall, 1991.

[GIE89] GIERING III E.W. y BAKER T.P.: "Toward the Deterministic Scheduling of Ada Tasks". IEEE, CH2803-5, 1989.

[GOM84] GOMAA H.: "A Software Design Method for Real-Time Systems". Communications of the ACM, Vol. 27, No. 9, September 1984.

[GON91A] GONZALEZ HARBOUR M., KLEIN M.H. y LEHOCZKY J.P.: "Fixed Priority Scheduling of Periodic Tasks with Varying Execution Priority". IEEE Real-Time Systems Symposium, 1991.

[GON91B] GONZALEZ HARBOUR M. y SHA L.: "An Application-Level Implementation of the Sporadic Server". Technical Report CMU/SEI-91-TR-26, ESD-91-TR-26, September 1991.

[GON94] GONZALEZ HARBOUR M., KLEIN M.H. y LEHOCZKY J.P.: "Timing Analysis for Fixed-Priority Scheduling of Hard Real-Time Systems". IEEE Transaction on Software Engineering, Vol. 20, no. 1, January 1994.

[GOO88] GOODENOUGH J.B. y SHA L.: "The Priority Ceiling Protocol: A Method for Minimizing the Blocking of High Priority Ada Task". Proceeding of the 2th International Workshop on Real-Time Ada Issues, June 1988.

[GRA89] GRANDA M. y DRAKE J.M. (Director): "Métodos para la Evaluación de Sistemas Multiprocesadores y su Aplicación a la Programación de Sistemas de Tiempo Real Mediante Prototipos Rápidos". Tesis Doctoral, Universidad de Cantabria, 1989.

[GUT91] GUTIERREZ GARCIA J.J., GREGORIO MONASTERIO J.A. y GONZALEZ HARBOUR M.: "Sistema de Control de un Robot para la Medida de Excentricidades de Tubos en el Interior de una Central Nuclear". Tesina de Licenciatura, Junio de 1991.

[GUT94] GUTIERREZ GARCIA J.J., GONZALEZ HARBOUR M. y PALENCIA GUTIERREZ J.C.: "Planificación y análisis de sistemas distribuidos de tiempo real estricto". Primeras Jornadas de Sistemas de Tiempo Real, Málaga, Octubre de 1994.

[GUT95A] GUTIERREZ GARCIA J.J. y GONZALEZ HARBOUR M.: "Optimized Priority Assignment for Tasks and Messages in Distributed Real-Time Systems". Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 124-132, April 1995.

[GUT95B] GUTIERREZ GARCIA J.J. y GONZALEZ HARBOUR M.: "Increasing Schedulability in Distributed Hard Real-Time Systems". Proceedings of the 7th Euromicro Workshop on Real-Time Systems, Odense, Denmark, pp. 99-106, June 1995.

[GUT95C] GUTIERREZ GARCIA J.J. y GONZALEZ HARBOUR M.: "Aumentando la Planificabilidad de Sistemas Distribuidos de Tiempo Real Estricto". IV Jornadas de Concurrencia, El Escorial, Junio de 1995.

[HAB90] HABAN D. y SHIN K.G.: "Application of Real-Time Monitoring to Scheduling Tasks with Random Execution Times". IEEE Transactions on Software Engineering, vol. 16, no. 12, pp.1374-1389, December 1990.

[JOS86] JOSEPH M. y PANDYA P.: "Finding Response Times in a Real-Time System". The Computer Journal (British Computing Society) 29, 5, pp. 390-395, October 1986.

[KIR83] KIRKPATRICK S., GELATT C.D. y VECCHI M.P.: "Optimization by Simulated Annealing". Science, vol. 220, no. 4598, pp. 671-680, May 1983.

[KLE90] KLEIN M. y RALYA T.: "An Analysis of Input/Output Paradigms for Real-Time Systems". Software Engineering Institute, Tech. Rept. CMU/SEI-90-TR-19, July 1990.

[KLE93] KLEIN M., RALYA T., POLLAK B., OBENZA R. y GONZALEZ HARBOUR M.: "A Practitioner's handbook for Real-Time Analisys". Kluwer Academic Pub., 1993.

[LAI86] LAIRD J.D., BURTON B.A. y KOPPES M.R.: "Implementation of an Ada Real-Time Executive - A Case Study". Annual National Conference on Ada Technology 1986, pp. 114-124.

[LEH86] LEHOCZKY J.P. y SHA L.: "Performance of Real-Time Bus Scheduling Algorithms". ACM Performance Evaluation Review, Special Issue 14, 1, May 1986.

[LEH87] LEHOCZKY J.P., SHA L. y STROSNIDER J.K.: "Enhanced Aperiodic Scheduling in Hard Real-Time Environments". Proceedings of the IEEE Real-Time Systems Symposium, Los Alamitos, CA, IEEE Computer Society Press, pp. 261-270, 1987.

[LEH89] LEHOCZKY J., SHA L. y DING Y.: "The Rate Monotonic Scheduling Algorithm: Exact Characterization and Average Case Behavior". IEEE Real-Time Systems Symp., CS Press, Los Alamitos, CA, pp. 166-171, 1989.

[LEH90] LEHOCZKY J.P.: "Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines". IEEE Real-Time Systems Symposium, 1990.

[LEI82] LEINBAUGH D.W. y YAMINI MR.: "Guaranteed Response Times in a Distributed Hard-Real-Time Environment". IEEE, 1982. Procedente: Real-Time System Symphosium, December 7-9, 1982.

[LEU82] LEUNG J. y WHITEHEAD J.: "On Complexity of Fixed-Priority Scheduling of Periodic Real-Time Tasks". Performance Evaluation, 2, pp. 237-250, 1982.

[LEV87] LEVESON N.G. y STOLZY J.L.: "Safety Analysis Using Petri Nets". IEEE Trans. on Software Eng. Vol. SE-13, No. 3, pp. 386-397 , March 1987.

[LIU73] LIU C.L. y LAYLAND J.W.: "Scheduling Algorithms for Multi-Programming in a Hard Real-Time Environment". Journal of the Association for Computing Machinery, vol. 20, no. 1, pp. 46-61, January 1973.

[LIU91] LIU J.W.S., LIN KJ., SHIH WK. y YU A.C.: "Algorithms for Scheduling Imprecise Computations". IEEE Computer, pp. 58-68, May 1991.

[LVI88] LEVINE G.: "The Control of Priority Inversion in Ada". Ada Letters, Vol. VIII, No. 6, November/December 1988.

[MAR89] MARINESCU D.y STANSIFER. R.: "Verification of Timing Correctness of Real-Time Programs". Software Engineering Research Center report no. SERC-TR-57-P, Purdue University, November 1989.

[MCE94] McELHONE C. : "Adapting and Evaluating Algorithms for Dynamic Schedulability Testing". Department of Computer Science, University of York, Technical Report YCS_225, February 1994.

[MED94] MEDIAVILLA M.E., DRAKE J.M. y GRANDA M. (Codirectores): "Modelado y Evaluación de Programas Paralelos con Distintas Estrategias de Planificación de Procesos". Tesis Doctoral, Universidad de Cantabria, 1994.

[MEN93] MENENDEZ DEL LLANO R. y DRAKE J.M. (Director): "Diseño y Evaluación de un Método de Programación en Ada para Sistemas Multicomputadores".Tesis Doctoral, Universidad de Cantabria, 1993.

[MIL91] MILLARD B.R., MILLER D.S. y WU C.: "Support for Ada Intertask Communication in a Message-Based Distributed Operating System". IEEE, CH2859-5, 1991.

[NIE88] NIELSEN K. y SHUMATE K.: "Designing Large Real-Time Systems with Ada". Mc. Graw Hill, 1988.

[NIE90] NIELSEN K.: "Ada in Distributed Real-Time Systems". Intertext Publications, Mc. Graw Hill Book Company, 1990.

[PAR91] PARK C.Y. y SHAW A.C.: "Experiments with a Program Timing Tool Based on Source-Level Timing Schema". IEEE Computer, pp. 48-56, May 1991.

[PLE92] PLEINEVAUX P.: "An Improved Hard Real-Time Scheduling for the IEEE 802.5". The Journal of Real-Time Systems, 4, pp. 99-112, 1992.

[POS93] IEEE Std. 1003.1b: "POSIX (Portable Operating System Interface)". IEEE Standard for Information Technology, 1993.

[PUS89] PUSCHNER P. y KOZA CH.: "Calculating the Maximum Execution Time of Real-Time Programs". The Journal of Real-Time Systems, Kluwer Academic Publishers, 1, pp. 159-176, 1989.

[RAJ88] RAJKUMAR R., SHA L. y LEHOCZKY J.P.: "Real-Time Synchronization Protocols for Multiprocessors". IEEE Real-Time Systems Symposium, CS Press, Los Alamitos, Calif., December 1988.

[RAJ89] RAJKUMAR R: "Task Synchronization in Real-Time Systems". Ph.D. Dissertation, Department of Electrical and Computer Engineering, Carnegie Mellon University. August 1989.

[RAJ90] RAJKUMAR R.: "Real-Time Synchronization Protocols for Shared Memory Multiprocessors". Proceedings of The 10th International Conference on Distributed Computing Systems, 1990.

[RAM80] RAMAMOROORTHY C.V. y GARY S.H.: "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets". IEEE Trans. on Software Eng. Vol. SE-6, No. 5, pp. 440-449 , September 1980.

[RMM90] RAMAMRITHAM K., STANKOVIC J.A. y SHIAH P.: "Efficient Scheduling Algorithms for Real-Time Multiprocessor Systems". IEEE Transactions on Parallel and Distributed Systems, vol. 1, no. 2, pp. 184-194, April 1990.

[SAV85] SAVITZKY S.R.: "Real-Time Microprocessor Systems". Van Nostrand Reinhold Company, 1985.

[SCM94] SCHMID U.: "Monitoring Distributed Real-Time Systems". Real-Time Systems, 7, pp. 33-56, 1994.

[SCH94] SCHOLEFIELD D.J.: "Proving Properties of Real-Time Semaphores". Department of Computer Science, University of York, Technical Report SCP_S_94, 1994.

[SCH94] SCHOLEFIELD D.J., ZEDAN H.S.M. y HE J.: "A Specification Oriented Semantics for The Refinement of Real-Time Systems". Department of Computer Science, University of York, Technical Report TCS_SZH, 1994.

[SHA89] SHA L., RAJKUMAR R., LEHOCZKY J.P. y RAMAMRITHAM K.: "Mode Change Protocols for Priority-Driven Preemptive Scheduling". Real-Time Systems, Kluwer Academic Publishers, vol. 1, no. 3, pp. 243-264, December 1989.

[SHA90A] SHA L., RAJKUMAR R. y LEHOCZKY J.P.: "Priority Inheritance Protocols: An Approach to Real-Time Synchronization". IEEE Transaction on Computers, vol. 39, no. 9, pp. 1175-1185, September 1990.

[SHA90B] SHA L. y GOODENOUGH B.: "Real-Time Scheduling Theory and Ada". IEEE Computer, pp. 53-62, April 1990.

[SHA91A] SHA L., RAJKUMAR R. y LEHOCZKY J.P.: "Real-Time Computing with IEEE Futurebus+". IEEE Micro, pp. 30-33 y 95-100, June 1991.

[SHA91B] SHA L., KLEIN M.H. y GOODENOUGH J.B.: "Rate Monotinic Analisys for Real-Time Systems". In Foundations of Real-Time Computing: Scheduling and Resource Management. Van Tilborg, Andre and Koob, Gary M., Ed. Kluwer Academic Piblishers, pp. 129-155, 1991.

[SHA92] SHA L. y SATHAYE S.S.: "Distributed Real-Time System Design using Generalized Rate Monotonic Theory". 2nd ICARCV International Conference, Vol. 3, pp. 5.5.1-7, 1992.

[SHA93] SHA L. y SATHAYE S.S.: "A Systematic Approach to Designing Distributed Real-Time Systems". IEEE Computer, 0018-9162, pp. 68-78, 1993.

[SHT88] SHATZ S.M. y CHENG. W.K.: "A Petri Net Framework for Automated Static Analysis of Ada Tasking Behavior". The Journal of Systems and Software, no. 8, pp. 343-359, 1988.

[SPI90] SPIVEY J.M.: "Specifying a Real-Time Kernel". IEEE Software, pp. 21-28, September 1990.

[SPR88] SPRUNT B., LEHOCZKY J. y SHA L.: "Exploiting Unused Periodic Time For Aperiodic Service Using The Extended Priority Exchange Algorithm". IEEE, pp 251-258, 1988.

[SPR89A] SPRUNT B., SHA L. y LEHOCZKY J.P.: "Aperiodic Task Scheduling for Hard-Real-Time Systems". The Journal of Real-Time Systems, Kluwer Academic Publishers, 1, pp. 27-60, 1989.

[SPR89B] SPRUNT B., SHA L. y LEHOCZKY J.P.: "Scheduling Sporadic and Aperiodic Events in a Hard Real-Time System". Carnegie Mellon University, Sofware Engineering Institute, Tech. Rept. CMU/SEI-89-TR-11, ESD-89-TR-19, April 1989.

[SPR90A] SPRUNT B. y SHA L.: "Implementing Sporadic Servers in Ada". Technical Report CMU/SEI-90-TR-6, 1990.

[SPR90B] SPRUNT B.: "Aperiodic Task Scheduling for Real-Time Systems". Ph.D. Thesis, Carnegie Mellon University, August 1990.

[STA87] STANKOVIC J.A. y RAMAMRITHAM K.: "The Design of the Spring Kernel". IEEE, pp. 371-384, 1987.

[STA88] STANKOVIC J.A. y RAMAMRITHAM K.: "Hard Real-Time Systems". IEEE Computer Society, catalog no. EH0276-6, 1988.

[STA90] STANKOVIC J.A. y RAMAMRITHAM K.: "What is Predictability for Real-Time Systems?". Real-Time Systems, Kluwer Academic Publishers, 2, pp. 247-254, 1990.

[STA91] STANKOVIC J.A. y RAMAMRITHAM K.: "The Spring Kernel: A New Paradigm for Real-Time Systems". IEEE, pp. 62-72, May 1991.

[STO77] STONE H.S.: "Multiprocessor Scheduling with the Aid of Network Flow Algorithms". IEEE Transaction on Software Engineering, vol. SE-3, no. 1, pp. 85-93, January 1977.

[STR88] STROSNIDER J.K., MARCHOK T., LEHOCZKY J.P.: "Advanced real-time scheduling using the IEEE 802.5 Token Ring", Proceedings of the IEEE Real-Time Systems Symposium, Huntsville, Alabama, USA, 1988, pp. 42-52.

[STT85] STOTTS P.D.: "Hierarchical Modeling of Software Systems with Timed Petri Nets". IEEE Catalog number 85CH2187-3, pp. 32-39, july 1985.

[STY94] STOYENKO A.D. y BAKER T.P.: "Real-Time Schedulability-Analizable Mechanisms in Ada9X". Proceedings of the IEEE, Vol. 82, No. 1, January 1994.

[SUN95] SUN J. y LIU J.W.S.: "Bounding the End-to-End Response Time in Multiprocessor Real-Time Systems". Proceedings of the 3rd Workshop on Parallel and Distributed Real-Time Systems, Santa Barbara, California, pp. 91-98, April 1995.

[TIN91] TINDELL K., BURNS A. y WELLINGS A.J.: "Guaranteeing Hard Real Time End-to-End Communications Deadlines". Department of Computer Science, University of York, Technical Report RTRG_91_107, December 1991.

[TIN92] TINDELL K.W., BURNS A., and WELLINGS A.J., "Allocating Real-Time Tasks. An NP-Hard Problem Made Easy". Real-Time Systems Journal, Vol.4, no.2, May 1992, pp. 145-166.

[TIN93A] TINDELL K. y BURNS A.: "Scheduling Hard Real-Time Multi-Media Disk Traffic". Department of Computer Science, University of York, Technical Report YCS_204, 1993.

[TIN93B] TINDELL K.: "Fixed Priority Scheduling for Hard Real-Time Systems". DPhil Thesis, Department of Computer Science, University of York, 1993.

[TIN93C] TINDELL K., BURNS A. y WELLINGS A.J.: "On Limitations of the Analysis Presented in the Paper `Engineering and Analysis of Fixed Priority Schedulers' by Katcher et al". Department of Computer Science, University of York, Technical Report RTRG_93_101, December 1993.

[TIN93D] TINDELL K.: "An Extendible Approach for Analysing Fixed Priority Hard Real-Time Tasks". Journal of Real-Time Systems, Vol. 6, No. 2, March 1994.

[TIN94A] TINDELL K. y BURNS A.: "Guaranteed Message Latencies for Distributed Safety-Critical Hard Real-Time Control Networks". Department of Computer Science, University of York, Technical Report YCS_229, May 1994.

[TIN94B] TINDELL K.: "Adding Time-Offsets To Schedulability Analysis". Department of Computer Science, University of York, Technical Report YCS_221, January 1994.

[TIN94C] TINDELL K., BURNS A. y WELLINGS A.J.: "Analysis of Hard Real-Time Communications". Department of Computer Science, University of York, Technical Report RTS_TBW_Comms, 1994.

[TIN94D] TINDELL K. y BURNS A.: "Guaranteeing Message Latencies on Controller Area Network (CAN)". Proceedings 1st International CAN Conference, Mainz, Germany, September 1994.

[TIN94E] TINDELL K., BURNS A. y WELLINGS A.J.: "Calculating Controller Area Network (CAN) Message Response Times". Proceedings 1994 IFAC workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain, September 1994.

[TIN94F] TINDELL K. y CLARK J.: "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems". Microprocessing & Microprogramming, Vol. 50, Nos. 2-3, pp. 117-134, April 1994.

[TOR92] TORON ANTONS B.: "Diseño e Implementación de una Herramienta para el Análisis de Sistemas de Tiempo Real Estricto". Tesina de Licenciatura, Septiembre de 1992.

[VAL90] VALLEJO F., GREGORIO J.A., GONZALEZ HARBOUR M. y DRAKE J.M.: "Shared Memory Multimicroprocessor Operating System with an Extended Petri Net Model". IEEE Trans. on Parallel and Distributed Systems, Vol. 5, No. 7, July 1994.

[VAL91] VALLEJO F. y GREGORIO J.A.(Director): "Diseño y Evaluación de un Entorno de Programación para Sistemas Multicomputadores Fuertemente Acoplados". Tesis Doctoral, Universidad de Cantabria, 1991.

[VOL85] VOLZ R.A., MUDGE T.N., NAYLOR A.W. y MAYER J.H.: "Some Problems in Distributing Real Time Ada Programs Across Machine". Ada in Use, Proceeding of the Ada International Conference, pp. 72-84, Paris, 1985.

[WAN93] WANG J.P. y STEIDLEY C.: "Task Allocation Model for Minimization of Completion Time in Distributed Computer Systems". SPIE Vol. 2056 Intelligent Robots and Computer Vision XII, pp. 430-436, 1993.

[WED94] WEDDE H.F., KOREL B. y HUIZINGA D.M.: "Formal Timing Analysis for Distributed Real-Time Programs". Real-Times Systems, 7, pp. 57-90, 1994.

[XUJ90] XU J. y PARNAS D.L.: "Scheduling Processes with Release Times, Deadlines, Precedence, and Exclusion Relations". IEEE Transaction on Software Engineering, vol. 16, no. 3, pp. 360-369, March 1990.

[ZHA94] ZHANG S. y BURNS A.: "A Study of Timing Properties With The Timed Token Protocol". Department of Computer Science, University of York, Technical Report YCS_226, April 1994.