UNIVERSIDAD NACIONAL DEL SUR

Tesis Doctor en Ingeniería

Desarrollo de una Herramienta de Diseño de Software de Tiempo Real para Sistemas Embebidos

Leonardo Damián Ordinez

BAHIA BLANCA ARGENTINA

2010

UNIVERSIDAD NACIONAL DEL SUR

Tesis Doctor en Ingeniería

Desarrollo de una Herramienta de Diseño de Software de Tiempo Real para Sistemas Embebidos

Leonardo Damián Ordinez

BAHIA BLANCA ARGENTINA

2010

a Mari, cada día, todos los días...

Prefacio

Esta tesis se presenta como parte de los requisitos para optar al grado Académico de Doctor en Ingeniería, de la Universidad Nacional del Sur y no ha sido presentada previamente para la obtención de otro título en esta Universidad u otra. La misma contiene los resultados obtenidos en investigaciones llevadas a cabo en el Laboratorio de Sistemas Digitales dependiente del Departamento de Ingeniería Eléctrica y de Computadoras durante el período comprendido entre el 21 de noviembre de 2006 y el 20 de mayo de 2010, bajo la dirección de los profesores Dr. Javier Orozco, profesor titular de Diseño Lógico y del Dr. Rodrigo Santos, profesor adjunto de Introducción a las Computadoras Digitales.

Leonardo D. Ordinez

Agradecimientos

Las páginas que siguen se llevan consigo un gran número de secretos. Una extraña clase de secretos que solo ellas y un par de personas más conocemos. Son los secretos que no se pueden contar ni mostrar. Son horas y horas de esfuerzo, de frustraciones, de alegrías y de amor. Y ese par de personas estuvieron siempre, a cambio de nada. Estas líneas las escribo cuando todo eso ya pasó y para quede aquí, grabado. No me dejarían que les agradeciera, así es que no lo voy a hacer, porque estuvieron a mi lado por amor y amistad. Vayan estos sentimientos a Mari, mi compañera de vida, la que prendió la luz cuando se me hizo de noche y abrió la ventana para que entre la alegría. Te amo con el alma. Y a David, el gran amigo, que *es* y *está*. ¡Por tantas historias y tantos motivos, salud!

En el camino hubo varias personas que por acción, omisión o contra ejemplo merecen un lugar acá. El agradecimiento eterno a Alfredo Olivero y Sergio Yovine. A los compañeros del Laboratorio de Sistemas Digitales. Una línea especial para Omar Alimenti, a quien descubrí sobre el final de esta tesis y me ha mostrado ser una gran persona. A Ricardo Cayssials, por su apoyo. A mis directores, Javier Orozco y Rodrigo Santos, por haber confiado en mí dándome libertad.

Finalmente, quiero agradecer y dedicar esta tesis a mis padres, quienes sin entender aún a qué se dedica su hijo, me apoyaron y se alegraron; y a mis abuelos, los de acá y los de allá, porque en la ausencia o el sombrío ocaso, nuestros ojos brillan cuando se cruzan.

Resumen

Los sistemas embebidos se han convertido en una parte importante de la vida cotidiana. Desde pequeños dispositivos de entretenimiento como reproductores de MP3, a enormes plantas industriales o satélites están aprovechando la continua investigación en este campo. Los sistemas de tiempo real son sistemas que tienen estrictas restricciones de tiempo en su especificación. En particular, el software para sistemas embebidos suele estar sujeto a restricciones de tiempo que se derivan de la definición misma del problema.

A partir de lo anterior, en esta tesis, se plantean los siguientes aportes: 1) la elaboración de una política de planificación dinámica que contemple, al momento de establecer las prioridades de las tareas, el comportamiento de éstas, lo cual implica ampliar el modelo clásico de las mismas; 2) la construcción de una política de manejo de recursos compartidos segura y de eficiente implementación, que pueda adaptarse y coexistir junto con la formulada como objetivo en el punto anterior, y que además sea utilizable durante todo el proceso de desarrollo de la aplicación; y finalmente 3) el desarrollo de un marco de software que brinde las bases para una correcta y rápida implementación de las políticas propuestas en los puntos anteriores, junto con todas aquellas entidades participantes en el sistema.

Abstract

Embedded systems have become an important part of everyday life. From small entertainment devices like MP3 players to huge industrial plants or satellites, are taking advantage of the ongoing research in this field. Real-time systems are systems that have strict time constraints in their specifications. In particular, the software for embedded systems is often subject to time constraints arising from the very definition of the problem.

From the above, in this thesis, the following contributions are made: 1) the development of a dynamic scheduling policy that takes into account the behavior of the tasks, when setting their priorities. This involves extending the classic model of tasks. 2) The construction of a policy for the management of shared resources which is safe and efficient. This policy can also adapt and coexist with the objective formulated in the previous point. In addition, the proposed policy can be used throughout the development process of an application. And finally 3) the development of a software framework that provides the foundation for a correct and rapid implementation of the policies proposed in the above, along with all entities participating in the system.

Índice general

Capítulo 1

Introducción

Este capítulo introduce el marco de referencia de la investigación realizada en esta tesis. Como el objeto de estudio principal se centra en sistemas de tiempo real, la mayor parte del capítulo está dedicada a exponer las definiciones básicas, los conceptos introductorios y los enfoques teóricos existentes. Asimismo, se manifiestan también las motivaciones que estimularon las investigaciones realizadas y su relación con los resultados alcanzados. En particular, este capítulo trata los siguientes temas:

- *Las características generales de un sistema embebido de tiempo real.*
- *Los conceptos de predecibilidad, planificación y factibilidad.*
- *Los aportes realizados hasta el momento en materia de planificación, tanto para sistemas con tareas independientes como para aquellos en que las tareas comparten recursos.*
- *La motivación y objetivos que tiene esta tesis.*

1.1. Sistemas Embebidos de Tiempo Real

Hoy en día, la vida de cualquier persona está influenciada y, en algunos casos, condicionada por una gran diversidad de dispositivos electrónicos. Estos dispositivos permiten comunicarse, obtener energía, detectar y curar enfermedades, posibilitan el funcionamiento de vehículos, de electrodomésticos, de plantas industriales e incluso de satélites. Gran parte de estos sistemas son denominados *sistemas embebidos*. Aunque no hay una única definición de sistema embebido, la mayoría de los autores [17, 37, 63, 68, 98], con sutiles diferencias, concuerdan que es un sistema desarrollado para uno o varios propósitos específicos, que forma parte de un sistema más grande, que interactúa fuertemente con el entorno físico mediante sensores y actuadores y que generalmente opera en forma autónoma, sin intervención de personas. Asimismo, sobre estos sistemas recae una restricción muy fuerte que no se suele encontrar en otro tipo de sistema computacional: la inclusión del tiempo como un requerimiento más dentro del sistema. Este hecho vuelve al sistema embebido uno de *tiempo real*.

Los *sistemas de tiempo real* son sistemas computacionales que involucran en su definición y caracterización restricciones temporales asociadas a cada entidad computacional, las cuales deben ser tenidas en cuenta no sólo en la etapa de diseño, sino que condicionan los procesos de codificación, compilación, testeo y entorno de ejecución. El parámetro temporal más sensible de estos sistemas es el *vencimiento*, a partir del cual el sistema se degrada y su desempeño puede pasar a ser inaceptable. Una definición ya clásica propuesta por John Stankovic en 1988 [93], determina que "en una computación de tiempo real la correctitud del sistema depende no sólo de los resultados lógicos de la computación, sino también del tiempo en que los resultados son producidos". La importancia de contemplar, en el desarrollo de un sistema, las restricciones temporales se puede ver claramente en el siguiente ejemplo, tomado del libro *Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications* [24], de Giorgio Buttazzo.

> El 25 de febrero de 1991, el radar de visión detectó un misil Scud dirigido a Arabia Saudita, y la computadora de a bordo predijo su trayectoria, realizó la verificación, pero clasificó el caso como una falsa alarma. Unos minutos más tarde, el Scud cayó sobre la ciudad de Dhahran, causando víctimas y daños económicos enormes. Posteriormente, se descubrió que, debido a un sutil error de software, el reloj de tiempo real de la computadora de a bordo acumulaba un retraso de cerca de 57 microsegundos por minuto. El día del accidente, el equipo había estado trabajando por cerca de 100 horas (una condición excepcional que nunca se experimentó antes), acumulando así un retraso total de 343 milisegundos. ¡Este retraso provocó un error de predicción en la fase de verificación de 687 metros! El error se corrigió el 26 de febrero, el día después del accidente.

Este ejemplo presentado por Buttazzo muestra, en un caso extremo y contundente, las consecuencias de una falla en el desarrollo de un sistema de tiempo real. En este sentido, las técnicas tradicionales que se aplican al desarrollo de software, deben adaptarse a fin de contemplar las necesidades propias de estos sistemas.

Desde un punto de vista más concreto, un sistema de tiempo real está conformado por un conjunto de componentes ejecutables que realizan funciones particulares dentro del sistema. Estos componentes son nombrados *tareas* y son los encargados de cumplir los propósitos para los que fue desarrollado el sistema. Las tareas se asemejan conceptualmente al modelo de *proceso* presente en los sistemas UNIX, aunque su definición es más amplia, ya que no está supeditada a un modelo de programación particular. Las tareas presentes en el sistema pueden ser tanto *duras* como *blandas*. En el primer caso, el no cumplimiento de sus restricciones temporales es inaceptable; mientras que en el segundo, una cantidad acotada de no cumplimientos se considera tolerable. Esta distinción entre tipos de tareas ya es clásica en la literatura [92], aunque continúa siendo objeto de investigaciones y discusiones dentro de la comunidad académica. Más allá de la diferentes variantes de esta definición, lo importante de destacar en este punto, es que tanto sobre tareas duras como sobre tareas blandas existen

restricciones temporales que deben cumplirse. Lo que difiere entre unas y otras es el grado de aceptación de los posibles incumplimientos, que se establece sobre cada una. Así, éste suele ser más estricto para las tareas duras que para las blandas, aunque en ambos casos dicho criterio es conocido y está claramente determinado.

La caracterización, desde un punto de vista temporal, de una tarea de tiempo real involucra una gran diversidad de parámetros que, a la vez, suelen presentarse de variadas formas [65, 66, 36, 6]. A continuación, se expone el modelo básico, con el que se caracteriza a un sistema de tiempo real y a sus tareas. En general, este modelo suele ser compartido por la mayoría de los autores en la literatura de sistemas de tiempo real. A partir de él es de donde se realizan nuevas abstracciones e hipótesis para las investigaciones. Como se dijo, un sistema de tiempo real, denotado $\mathbf{\Gamma}(n)$, se compone de n tareas, simbolizadas τ_i. Esto es, $\mathbf{\Gamma}(n) = \{\tau_i | i = 1, \ldots, n\}$. Asimismo, cada tarea (sea dura o blanda), está caracterizada temporalmente por los siguientes parámetros: su peor caso de tiempo de ejecución, C_i; su período de ejecución o mínimo tiempo entre arribos, T_i; y su vencimiento relativo, D_i, el cual se utiliza para calcular el vencimiento absoluto, $d_i = a_i + D_i$, donde a_i es el instante en que la tarea arriba al sistema. El vencimiento es el parámetro más sensible dentro del desarrollo de sistemas de tiempo real, ya que se debe garantizar que todas las tareas del sistemas se ejecuten antes de sus correspondientes vencimientos. Como las tareas son periódicas, pueden verse como una secuencia de *trabajos* o *instancias*, J_{ij}, donde el primer subíndice se refiere a la tarea y el segundo a la instancia. Cabe remarcar, que el modelo de sistema adoptado en esta tesis y el desarrollo de la misma se realiza para sistemas de tiempo real monoprocesador. Es decir, todas las tareas del sistema compiten por un único elemento de procesamiento. El área de investigación de sistemas de tiempo real para multiprocesadores está fuera del alcance de esta tesis.

El modelo de sistema presentado en el párrafo anterior se denomina sistema de *tareas independientes*, ya que las tareas no tienen relación entre sí y sólo comparten (compiten por) el procesador. De otra manera, existen situaciones en las que las tareas del sistema compar-

ten recursos unas con otras. Estos *recursos compartidos*, que pueden ser tanto de hardware (*e.g.*, sensores, actuadores, interfaces de comunicación) como de software (*e.g.*, variables, señales, mensajes), introducen mayor complejidad al tratamiento de las restricciones temporales [87, 12]. El acceso a los recursos involucra generalmente una exclusión mutua entre quienes comparten dichos recursos a fin de evitar *condiciones de carrera* [95]. La utilización de mecanismos de exclusión mutua (*e.g.*, semáforos) trae aparejada un potencial bloqueo. Esto es, una tarea de menor prioridad accede a un recurso compartido, en forma mutuamente excluyente, evitando que una de mayor prioridad pueda hacerlo. Esta situación también se conoce como *inversión de prioridad* y representa un aspecto determinante al momento de diseñar un sistema de tiempo real, ya que la misma puede conducir directamente a la pérdida de vencimientos. La aparición de recursos compartidos en un sistema de tiempo real lleva a ampliar la caracterización expuesta anteriormente. Formalmente, el sistema $\mathbf{\Gamma}(n)$ ahora, a la vez, cuenta con un conjunto $\mathbf{R}(m) = \{R_1, R_2, \ldots, R_m\}$ de m recursos compartidos. Asimismo, cada tarea τ_i se caracteriza también por un tiempo de bloqueo máximo B_i, que representa el tiempo máximo que puede sufrir una inversión de prioridad.

De la definición planteada anteriormente por Stankovic y la caracterización temporal de un sistema de tiempo real, se puede ver que a la correctitud requerida para todo sistema de software[1], se le impone una nueva dimensión: la temporal. Esto es, el comportamiento a lo largo del tiempo de estos sistemas debe ser *predecible*. El concepto de predecibilidad extiende el concepto de correctitud de la siguiente manera [94]: predecibilidad significa que "debería ser posible mostrar, demostrar o probar que se cumplen los requerimientos sujetos a las hipótesis formuladas". Con lo cual, un sistema de tiempo real es predecible si respeta las restricciones temporales impuestas en su definición. Nótese que lo anterior no implica que el sistema deba ser rápido. Este concepto, y otros, fueron abordados en [93] con el objetivo de prevenir malas interpretaciones de ciertos aspectos básicos de los sistemas de tiempo real. A fin de aclarar este punto, se presentan dos sistemas concretos que serán luego retomados

[1]Correctitud definida como la resolución, en un tiempo finito, del problema para el cual fue diseñado el sistema.

a lo largo de la tesis.

Alerta Temprana de Incendios Forestales. El inicio de los incendios forestales está sujeto a la presencia de una fuente de ignición. Una vez que la fuente de ignición está presente, varios factores ambientales determinan la ocurrencia de incendios. Características de la vegetación, el estado del tiempo y la topografía son aspectos fundamentales a fin de predecir el comportamiento del fuego. Diferentes sistemas han sido desarrollados para evaluar los principales factores que afectan a la ocurrencia, el comportamiento y las consecuencias del fuego [44, 96, 53]. En general, se componen de varios módulos, cada uno de los cuales indica la contribución de un aspecto específico de las características del fuego. Estos son sistemas complejos que incorporan las relaciones entre las variables climáticas, los estados del combustible presente y el comportamiento del fuego para generar indicadores que proporcionan una medida cuantitativa de las dificultades para el control y el daño potencial que podría causar un incendio.

Estos sistemas se caracterizan por estar expuestos a condiciones ambientales adversas, tener que operar con escaso consumo de energía, ya que suelen ser alimentados por baterías, y manejar restricciones temporales con magnitudes relativamente grandes (*i.e.*, minutos u horas). Este último punto se debe a que los cambios en las variables ambientales que monitorean son lentos. ■

Alerta de Colisiones en Automóviles Los sistemas de evasión de colisiones se encuentran en muchos de los sistemas electrónicos de vehículos modernos [19, 59, 5]. Éstos vigilan el camino delante del vehículo y advierten al conductor cuando existe un potencial riesgo de colisión. Por ejemplo, los radares disponibles en la actualidad utilizan algoritmos complejos para determinar la distancia, acimut y la velocidad relativa entre el vehículo portador del sistema y el vehículo u objeto delante de él. Cuando el vehículo portador del sistema está viajando a lo largo del camino, éste le puede avisar al conductor en que momento su vehículo o el objeto está en su carril dentro de un umbral predeterminado. Actualmente, la mayoría

de estos sistemas no toman ninguna acción automática para evitar una colisión o de control del vehículo, por lo tanto, los conductores son responsables de la operación segura de sus vehículos utilizando la dirección y el frenado, si es seguro hacerlo, para evitar un accidente. A medida que el intervalo de tiempo con el vehículo de adelante se reduce, el sistema debe emitir una advertencia cada vez más grave. Asimismo, cuando un objeto se acerca a la parte delantera del vehículo en un ángulo diferente, el sistema debe emitir un tipo diferente de alarma. Los fabricantes de sistemas establecen así diversos umbrales de alerta.

Esta clase de sistemas se caracterizan por estar en continuo funcionamiento mientras el automóvil se encuentra en marcha. En este caso, no hay una gran restricción en cuanto a consumo de energía. Aunque, sí la hay, más fuerte que el caso anterior, en relación a las magnitudes temporales. En estos sistemas, las tiempos que se manejan son en unidades inferiores a los segundos (milisegundos). ■

Cabe destacar que los dos casos propuestos anteriormente de sistemas con restricciones temporales, junto con el ejemplo tomado de Buttazzo, muestran claramente que un sistema de tiempo real no es aquel que provee respuestas rápidas; sino que lo hace en forma predecible. Es decir, se puede conocer previamente la manera en que se va a comportar el sistema.

La rama teórica encargada de estudiar y proveer políticas que aseguren el comportamiento predecible de los sistemas de tiempo real es la *planificación*. El objetivo principal que se persigue en dicho estudio es encontrar políticas óptimas para organizar la ejecución de dichos sistemas ante diferentes restricciones temporales impuestas sobre los mismos en su definición. Esa organización presupone la imposición de prioridades sobre los componentes del sistema. En general, la prioridades se pueden caracterizar en dos grupos: *prioridades fijas*, las cuales no varían para cada componente a lo largo del tiempo; y *prioridades dinámicas*, que son dependientes del instante de tiempo en que se encuentre el sistema actualmente. Una vez asignadas las prioridades a las tareas mediante la política de planificación, aquella tarea de mayor prioridad será la que ejecute en cada instante de tiempo. Un aspecto distintivo a considerar a lo largo de los más de 30 años de estudio sobre planificación, es que la asignación

de prioridades de acuerdo a las múltiples y variadas políticas de planificación no tiene en consideración otros parámetros que los temporales del sistema. Con lo cual, las políticas se desentienden de la funcionalidad de los componentes para únicamente tener en cuenta, al momento de asignarles prioridades, sus características temporales.

Por otro lado, ante la posibilidad de que el sistema incurra en un comportamiento anormal en tiempo de ejecución, se requiere de mecanismos que eviten o acoten el daño que se pueda sufrir. Resulta necesario que el mal comportamiento temporal de un componente no afecte al resto de ellos, a fin de limitar los perjuicios. En este sentido, una técnica comúnmente utilizada es aquella basada en *servidores* que implementan mecanismos de reserva de recursos. Los servidores son una abstracción de software que encapsula uno o varios componentes del sistema y que es tomado como entidad planificable. En este sentido, los servidores aislan a los componentes que encapsulan de otros componentes y otros servidores presentes en el sistema, con lo cual el mal funcionamiento de un componente dentro de un servidor no daña a otros dentro del sistema. Esta propiedad es conocida como *aislamiento temporal*[2] [78] e involucra el concepto abstracto de poder ver al servidor como un procesador virtual que funciona a una frecuencia menor a la del procesador real [6]. Esto refuerza lo dicho anteriormente de que dentro de un servidor, la predecibilidad depende únicamente de los componentes encapsulados. En el caso de servidores que encapsulan un sólo componente, el mismo no se verá afectado por ningún otro componente presente en el sistema.

Toda política de planificación debe proveer la forma de verificar bajo qué condiciones un sistema planificado por esa política respetará sus restricciones temporales. Esto se lleva a cabo mediante un *test de factibilidad*, el cual determina si un dado sistema, bajo las condiciones de la política en cuestión, cumplirá con todas sus restricciones temporales.

Finalmente, cabe señalar que las políticas de planificación y los diversos mecanismos que ellas utilizan, se implementan generalmente mediante un Sistema Operativo de Tiempo Real (SOTR), el cual provee las ventajas comunes a los sistemas operativos [95], junto con

[2]La palabra *temporal* hace referencia a que el aislamiento es referido a la dimensión tiempo y no a la duración de un intervalo de tiempo.

aquellas necesarias para trabajar en un ambiente de tiempo real. En particular, el SOTR es fundamental en la construcción de un sistema de tiempo real, ya que permite, mediante mecanismos seguros, un manejo eficiente del tiempo durante el que las tareas acceden al procesador. En este sentido, el SOTR es el encargado de manejar las apropiaciones, es decir, los cambios de contexto que involucran que una u otra tarea ejecuten y que, cuando la que fue interrumpida retome su ejecución ésta siga siendo consistente.

1.2. Fundamentos de Planificación Dinámica

Una política de planificación de tiempo real define la manera en que las tareas acceden a los recursos del sistema. Estos recursos pueden ser físicos, *i.e.* procesador, dispositivos de hardware; o lógicos, *i.e.* variables, señales, mensajes. A la vez, las tareas dentro del sistema pueden ser duras o blandas. Un sistema que admite ambos tipos de tareas se denominará *mixto*. En esta tesis se aborda el área de sistemas de tiempo real mixtos, en un ambiente multiprogramado de tiempo compartido [95], donde ambos tipos de tareas pueden compartir diversos recursos. Es decir, las entidades del sistema se ejecutan concurrentemente y en un mismo instante de tiempo puede haber dos o más de esas entidades compitiendo por el acceso a un mismo recurso. Para poder decidir cuál de esas entidades debe ejecutarse, a éstas se les asignan prioridades. La asignación de prioridades suele estar establecida por la política de planificación. Así, se tienen esquemas de asignación basados en la proximidad del vencimiento de la tarea con respecto al tiempo actual, en la duración del período de la tarea, en la duración del vencimiento relativo, por prioridades fijas arbitrariamente elegidas por el desarrollador, entre otras. En general, el planificador del SOTR es quien luego resuelve la concurrencia sobre el recurso de acuerdo a dicha política. En particular, el planificador que implementa una política dinámica se activa en ciertos puntos específicos de planificación [100]. Estos incluyen:

- Creación de una tarea.

- Finalización de una tarea (ya sea que fue abortada o terminó normalmente).
- Acceso a un recurso compartido.
- Liberación de un recurso compartido.

El trabajo fundacional en cuanto a planificación de tiempo real es, sin dudas, el de Liu y Layland de 1973, denominado "*Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*" [65]. En ese trabajo, se establece la caracterización temporal de tareas presentada en la sección anterior. Una de las condiciones impuestas en el trabajo de Liu y Layland es la concordancia entre vencimiento y período de una tarea, esto es, $D_i = T_i$ para toda tarea τ_i. Además, dicho trabajo considera intolerable la pérdida de vencimientos, con lo que el sistema que trata es de tareas duras. Allí, se presentan las dos principales políticas de planificación desarrolladas hasta el momento para sistemas de tiempo real duros: 1) *Rate Monotonic* (RM), de prioridades fijas, la cual establece las prioridades en relación inversa a la duración del período; y 2) *Earliest Deadline First* (EDF), para prioridades dinámicas, que establece las prioridades en relación al tiempo restante al vencimiento absoluto de la tarea, es decir, a medida que una tarea se acerca a su vencimiento, su prioridad aumenta, con lo cual la tarea más próxima a su vencimiento absoluto será la de mayor prioridad. Por otro lado, Mok [72] realiza una caracterización más fina de las características temporales de una tarea, introduciendo el concepto de tarea blanda. Este tipo de tarea admite la pérdida "ocasional" de vencimientos. En cuanto a la utilización de servidores para planificar tareas blandas, el primer trabajo completo sobre este tema es el de Sprunt [90]. Allí, el autor introduce el mecanismo de servidores para planificar tareas blandas junto a otras duras, en un esquema de prioridades fijas basado en RM.

Con todo, los tres trabajos citados anteriormente ([65, 72, 90]) representan estudios fundacionales a partir de los cuales se ha desarrollado la mayor parte de la teoría de planificación en tiempo real de la actualidad. En este sentido, a continuación se expone una exploración de las tres líneas de investigación inauguradas por esos trabajos: 1) sistemas de tareas indepen-

dientes; 2) sistemas basados en servidores; y 3) sistemas de tareas con recursos compartidos. Cabe destacar, que el repaso se hace sobre sistemas basados en prioridades dinámicas, que es el área de estudio de esta tesis. Por otro lado, es importante resaltar que en la historia de la teoría de sistemas de tiempo, la investigación en el área de planificación con prioridades dinámicas tuvo su advenimiento masivo un tiempo después que aquella sobre prioridades fijas, ya que la segunda presentaba menores dificultades de implementación que la primera en detrimento de mejoras en el desempeño [46].

1.2.1. Sistemas Independientes

En esta sección, se hará un repaso general por los principales trabajos referidos a políticas de planificación de tareas que no comparten otro recurso que el procesador. Asimismo, se asumirá que las tareas son periódicas y están constituidas por una secuencia infinita de instancias de sí mismas. Como las políticas a analizar son dinámicas, una manera interesante de ver la asignación de prioridades es sobre las instancias particulares y no sobre cada tarea. Más aún, la prioridad de una instancia en un determinado período de ejecución no necesariamente será constante.

Como se dijo anteriormente, bajo la política EDF se asignan prioridades en forma inversamente proporcional al vencimiento absoluto de las instancias de tareas activas en el sistema. En [65] se estableció la condición de factibilidad de un conjunto de n tareas planificadas por EDF, como

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1$$

En este sentido, Dertouzos [45] probó formalmente que la política EDF es óptima entre todas aquella políticas de planificación apropiativas. Esto es, que si existe una planificación factible para un determinado conjunto de tareas, entonces la planificación generada por EDF también es factible.

Aloysius Mok presentó en [72], la política de planificación *Least Laxity First* (LLF), que

también es óptima en el mismo sentido que EDF. El autor define el parámetro *laxitud* l, como $l = d - t - c$. La política LLF asigna el procesador a la tarea activa con menor laxitud. La definición del parámetro l hace que su variación en tiempo de ejecución sea muy dinámica, lo que ha hecho que la mayor parte de la investigación en la comunidad de tiempo real se haya concentrado más en EDF que en LLF [88].

En la línea de investigación de EDF, dos trabajos realizan aportes fundamentales que amplían el original de Liu y Layland. En primer lugar, Chetto y Chetto, en [35], presentan dos implementaciones llamadas EDS (por *Earliest Deadline as Soon as possible*) y EDL (por *Earliest Deadline as Late as possible*). La implementación EDS es similar a EDF y la asignación de prioridades se hace en tiempo de ejecución. Por otro lado, EDL impone que previo a la ejecución se determinen los tiempos de arribo de las tareas en el intervalo $[0, T]$, lo cual hace que el algoritmo se deba correr antes del tiempo de ejecución.

En segundo lugar, Baruah *et ál* [16] consideran tareas cuyos vencimientos relativos son menores que sus períodos. Con este propósito introducen el criterio de *demanda de procesador*. Básicamente, la demanda de procesador en un intervalo $[t_1, t_2]$ es la cantidad de tiempo de procesamiento $g(t_1, t_2)$ solicitado por aquellas instancias de tareas que se activaron en $[t_1, t_2]$ y que deben completarse en ese mismo intervalo. En consecuencia, la factibilidad de un conjunto de tareas se da si y sólo si en cualquier intervalo de tiempo la demanda total de procesador no excede el tiempo disponible.

1.2.2. Sistemas basados en Servidores

Los sistemas tratados en la sección anterior estaban compuestos por tareas duras, periódicas e independientes bajo políticas de planificación dinámica. Aquí, se incluirán en el sistema (y en la planificación) tareas blandas aperiódicas. Esta inclusión incrementa la complejidad de la política de planificación y lleva a tomar decisiones particulares para que la ejecución de las tareas blandas no perjudique la de las duras. En este sentido, a continuación se presentan los esquemas de planificación fundamentales basados en servidores para prioridades

dinámicas, en particular, para EDF.

Ghazalie y Baker, en [52], consideran un conjunto de tareas periódicas y aperiódicas ejecutando sobre un único procesador. Las tareas se asumen independientes, excepto en casos donde se comparten recursos. A la vez, estas tareas pueden tener un vencimiento duro, uno blando, ambos o ninguno. Los autores presentan tres tipos de servidores para atacar este problema:

Deadline Deferrable Server (DDS) : el DDS es una tarea periódica cuyo vencimiento relativo es igual a su período. Este servidor puede atender arribos de tareas que ocurran en el medio de su período.

Deadline Sporadic Server (DSS) : el DSS tiene un período T_s y un tiempo de ejecución C_s. El algoritmo intenta presupuestar el tiempo de ejecución del servidor de modo que el efecto de éste en la planificación de las tareas duras no sea peor que el de una tarea dura con período T_s y tiempo de ejecución C_s.

Deadline Exchange Server (DXS) : este servidor es una modificación y simplificación del algoritmo DSS. La idea del servidor DXS es descartar su tiempo de ejecución remanente tan pronto como no tenga más tareas que atender, a cambio de recargar su tiempo de ejecución por completo más temprano.

Otro método eficiente para planificar tareas aperiódicas bajo EDF es el *Total Bandwidth Server* (TBS) [91]. Este servidor asigna a cada instancia de una tarea aperiódica que arriba al sistema un vencimiento, de manera que la carga total de las tareas aperiódicas nunca exceda un valor máximo U_s. A pesar de que el TBS puede manejar eficientemente ráfagas de instancias de ejecución, no puede ser usado para servir instancias con tiempo de ejecución variable o desconocido.

Un algoritmo similar al del TBS fue propuesto por Deng *et ál* en [43]. Este enfoque se denomina *Constant Utilization Server* (CUS) y fue desarrollado como parte de un esquema de planificación jerárquico para sistemas abiertos. Esta clase de sistemas no impone ninguna

restricción en cuanto a la creación y destrucción de tareas en tiempo de ejecución, por lo cual su manejo es altamente dinámico.

Como se mencionó anteriormente, el TBS no es adecuado para manejar tareas con tiempos de ejecución indeterminados. Para atender esa situación se necesita un mecanismo de manejo del presupuesto del servidor, a fin de evitar que una tarea ejecute más tiempo del permitido y perjudique a las otras. Este problema fue solucionado por Abeni y Buttazzo [6], mediante el *Constant Bandwidth Server* (CBS). En este esquema cada instancia atendida por el CBS tiene asignado un vencimiento igual al de su servidor. Esto permite mantener la demanda dentro del ancho de banda reservado. Cada vez que el presupuesto del servidor se agota, éste se recarga al valor máximo y el vencimiento del servidor se pospone por un período, para reducir las interferencias con otras tareas. Se debe notar que al posponer el vencimiento, la tarea continua estando lista para ejecutar.

Finalmente, Maryline Silly presentó, en [89], el servidor EDL. Este esquema se enfoca en tareas duras periódicas y blandas aperiódicas, donde sólo entre las primeras puede haber recursos compartidos. La idea fundamental del servidor EDL consiste en ejecutar las tareas periódicas lo antes posible, hasta el arribo de la primer tarea blanda aperiódica. Y mientras haya tareas blandas aperiódicas que ejecutar, las tareas duras periódicas se ejecutan lo más tarde posible.

1.2.3. Sistemas con Recursos Compartidos

En un sistema de tiempo real, comúnmente el acceso exclusivo a un recurso compartido se hace mediante operaciones sobre semáforos. El uso de estos mecanismos debe ser estrictamente controlado por un protocolo que, además de asegurar un acceso mutuamente exclusivo, garantice la ausencia de *deadlocks* y acote las inversiones de prioridad. Una inversión de prioridad aparece cuando una tarea de mayor prioridad debe esperar que una tarea de menor prioridad libere un recurso, para poder continuar su ejecución. En este caso, se dice que la tarea de mayor prioridad es *bloqueada* por la de menor prioridad. Cuando estas

inversiones de prioridad no están acotadas, el sistema de tiempo real puede pasar a tener un comportamiento temporal impredecible.

El trabajo fundacional sobre manejo de recursos compartidos en sistemas de tiempo real es el de Sha, Rajkumar y Lehoczky [87], cuya versión original es de 1987 [86]. Allí, los autores introducen los conceptos básicos para manejo de recursos compartidos en sistemas basados en prioridades fijas. A continuación se explican estos conceptos:

Herencia de Prioridad: cuando una tarea τ bloquea una o más tareas de mayor prioridad, ignora su prioridad asignada y ejecuta su sección crítica con la mayor prioridad de todas las tareas que bloqueó. Luego de ejecutar la sección crítica, la tarea recupera su prioridad original.

Techo de Prioridad: se define sobre cada semáforo y se calcula como la prioridad de la tarea de mayor prioridad que accede a ese semáforo. Asimismo, para que una tarea pueda acceder al recurso debe tener una prioridad mayor que la del semáforo de mayor prioridad actualmente en el sistema.

Chen y Lin [33] extendieron el trabajo de Sha *et ál*, construyendo el *Dynamic Priority Ceiling Protocol* (DPCP), para sistemas planificados por EDF. La idea básica de DPCP es la siguiente: un techo de prioridad se define para cada sección crítica y su valor en cualquier tiempo t es la prioridad de la tarea de mayor prioridad (*i.e.*, la tarea con vencimiento más próximo) que pueda entrar a la sección crítica en o después de t. El protocolo consiste de dos mecanismos llamados herencia de prioridad y techo de prioridad, respectivamente. En el primer mecanismo, una tarea de baja prioridad τ, dentro de una sección crítica, temporariamente hereda la prioridad de la de mayor prioridad que espera que τ salga de la sección crítica. En el mecanismo techo de prioridad, a una tarea τ se le permite entrar a una sección crítica sólo si su prioridad es mayor que los techos de prioridad de todas las secciones críticas actualmente en uso por otras tareas.

En [12], Baker presenta la Política de Recurso Pila (SRP, por *Stack Resource Policy*, en

inglés), la cual es una de las más utilizadas junto con EDF, por sus propiedades y eficiencia [88]. Baker introduce la noción de *nivel de apropiación*, de forma separada de los niveles de prioridad bajo EDF. En consecuencia, cada instancia de una tarea τ_i tiene asignada una prioridad $p(\tau_i)$, de acuerdo a su vencimiento absoluto d_i; y un nivel de apropiación $\pi(\tau_i)$ inversamente proporcional a su vencimiento relativo D_i. Cada recurso compartido tiene un techo, el cual es el máximo nivel de apropiación de todas las tareas que requieran ese recurso. A la vez, se define un *techo del sistema* $\overline{\pi}$, que es el mayor techo actual de todos los recursos tomados. La regla de planificación de SRP establece que una tarea no puede comenzar su ejecución hasta que su prioridad sea mayor que la de todas las tareas activas y su nivel de apropiación sea mayor que el techo del sistema. Con esto se garantiza que una vez que una tarea comienza su ejecución no podrá ser bloqueada.

Finalmente, en [56], Jeffay presenta el algoritmo EDF/DDM (EDF con Modificación Dinámica de Vencimiento). Bajo esta política, las tareas que requieren acceso exclusivo a recursos tienen dos tipos de vencimientos: un vencimiento de *competencia*, para la adquisición inicial del procesador; y uno de *ejecución* para la ejecución subsiguiente. La política EDF/DDM asegura que las tareas bloqueadas son reanudadas lo más pronto posible. El mismo autor asegura que esta política es pesimista en el sentido que asume que la planificación de una tarea que va a acceder a un recurso siempre derivará en que otra tarea sea bloqueada.

1.3. Motivación

Los trabajos analizados en la sección previa representan puntos fundacionales de diversas líneas de investigación en el área de planificación de sistemas de tiempo real. En particular, como ya se mencionó, los trabajos se centran en planificación dinámica. Este tipo de políticas asigna prioridades a las tareas en tiempo de ejecución de acuerdo a alguna regla preestablecida. En general, estas reglas se basan en algún parámetro temporal de las tareas. Esto ha derivado en que la investigación en políticas de planificación para sistemas de tiempo real

se focalice, casi enteramente, en la búsqueda de nuevos parámetros temporales o combinaciones de los mismos que garanticen un óptimo desempeño del sistema con respecto a algún criterio[3].

Por otro lado, como se explicó y ejemplificó en la Sección 1.1, los sistemas de tiempo real generalmente están fuertemente ligados al mundo físico. Su dominio de aplicación puede encontrarse en sistemas de control, procesos químicos, automóviles, monitoreo de eventos ambientales y meteorológicos, entre otros. Estas aplicaciones tan diversas comparten una estructura de funcionamiento básica que es la toma de información del mundo físico, su procesamiento y la consecuente realización de acciones que lleven al sistema a un punto deseado. De esto se desprende que la importancia de las tareas varía en función del estado del sistema. A la vez, cada tarea realiza acciones a favor del sistema cuyos resultados influyen determinantemente en el estado del mismo. Estas acciones representan aspectos *funcionales* de las tareas y su conjunto, en una instancia de ejecución, se define como el *comportamiento* de la tarea.

Como se dijo, la mayoría de las políticas de planificación basan sus reglas y formulación en aspectos *no funcionales* de las tareas (*e.g.*, parámetros temporales). Sin embargo, el comportamiento de las mismas, también determina su importancia dentro del sistema. Esta importancia generalmente se manifiesta en términos de la prioridad de la tarea. Y esa prioridad se expresa como función de los parámetros temporales, sin tener en cuenta el comportamiento. De esto surge la necesidad de elaborar políticas de planificación que no sólo tengan en cuenta los aspectos no funcionales de las tareas, sino también los funcionales, al tiempo de determinar la importancia de las mismas en el sistema y asignar las correspondientes prioridades. Asimismo, el hecho de que el comportamiento de una tarea varíe entre instancias de la misma, lleva a que la política que tenga en cuenta este aspecto deba ser inherentemente dinámica, ya que de otra manera sería imposible reflejar ese hecho en la

[3]Una muestra clara de esto se puede ver en artículo "*Real-Time Scheduling Theory: A Historical Perspective*" [88], que contiene un compendio de los aportes más trascendentales realizados entre 1973 y 2004, en planificación de sistemas de tiempo real.

prioridad. Del mismo modo, la naturaleza crítica de los sistemas de tiempo real determina que haya que tomar los recaudos suficientes para evitar que el mal funcionamiento de una tarea perjudique al resto del sistema. En este sentido, los mecanismos basados en servidores son una excelente alternativa para evitar dichas interferencias y eliminar los potenciales riesgos de manera simple y a bajo costo de complejidad computacional. Por otro lado, una característica común a cualquier sistema multiprogramado es que las entidades del sistema compartan recursos durante su ejecución. En sistemas de tiempo real, la inclusión de recursos compartidos agrega nuevas restricciones a la planificación, ya que los recursos compartidos pueden involucrar no sólo *deadlocks*, sino también inversiones de prioridad y tiempos de bloqueo no acotados. Estas últimas situaciones pueden llevar a un mal desempeño del sistema y a la pérdida de vencimientos. Para evitar dichas situaciones indeseadas, se requiere de políticas de manejo de recursos compartidos que garanticen ejecuciones libres de *deadlocks* y acoten el número de inversiones de prioridad, a fin de mantener la predecibilidad del sistema. A la vez, otras dos características requeridas para una política de este tipo son: 1) el cálculo preciso del tiempo máximo que una tarea puede ser bloqueada; y 2) la minimización del número de apropiaciones que puede sufrir una tarea.

Finalmente, un tercer aspecto que motivó esta tesis (y que se suma a la inclusión del comportamiento en la planificación y al uso seguro y eficiente de recursos compartidos), es el acercamiento de los conceptos teóricos de planificación a la implementación práctica. Durante muchos años, la investigación en sistemas de tiempo real se ha centrado en la elaboración de políticas y algoritmos de planificación que garanticen formalmente una ejecución predecible y factible del sistema. Se han construido métodos muy avanzados para brindar un óptimo desempeño del sistema con respecto a los más diversos objetivos. Sin embargo, la realidad continúa siendo la misma. Los fabricantes de sistemas siguen confiando en políticas que, aunque efectivas, distan de ser eficientes. Más aún, la mayoría de los sistemas industriales, satelitales y de automóviles están basados en prioridades fijas. Esta distancia entre la investigación académica y la utilización práctica, se debe en parte a la carencia de ar-

quitecturas, marcos o estructuras de software para estas nuevas políticas. De este modo, un marco de software lo suficientemente específico para incluir todas las características propias de una política y lo suficientemente genérico para permitirle libertad al desarrollador para adaptarlo a su aplicación particular, contribuiría favorablemente en el acercamiento entre teoría y práctica.

1.4. Objetivos

A partir de lo expresado como motivación de esta tesis, se puede inferir que la misma persigue tres objetivos bien claros y sumamente relacionados:

I. La elaboración de una política de planificación dinámica que contemple, al momento de establecer las prioridades de las tareas, el comportamiento de éstas. Esto implica naturalmente la ampliación del modelo clásico de tarea, para tener en cuenta dicho comportamiento y poder ponderarlo.

II. La construcción de una política de manejo de recursos compartidos segura y de eficiente implementación, que pueda adaptarse y coexistir junto con la formulada como objetivo en el punto anterior.

III. El desarrollo de un marco de software que brinde las bases para una correcta y rápida implementación de las políticas propuestas en los puntos anteriores, junto con todas aquellas entidades participantes en el sistema.

1.5. Aportes y Resultados Alcanzados

Esta tesis ha sido desarrollada en el Laboratorio de Sistemas Digitales del Instituto de Investigaciones en Ingeniería Eléctrica, bajo las normas internas sobre estudios de posgrado en dicho laboratorio. Estas normas expresan que toda tesis debe tener como base una cantidad suficiente de artículos publicados en revistas o congresos con referato. Y que en

dichos artículos el tesista debe ser primer o segundo autor, siendo inaceptable la utilización de un mismo resultado por más de un tesista. A continuación se enumeran las principales publicaciones desarrolladas en el marco de esta tesis:

- **Leo Ordinez**, David Donari, Rodrigo Santos, Javier Orozco: "Resource sharing in behavioral based scheduling", *24th ACM Symposium on Applied Computing*, Honolulu, Hawaii, USA, 2009.

- **Leo Ordinez**, David Donari, Rodrigo Santos, Javier Orozco: "A behavior priority driven approach for resource reservation scheduling", *23rd ACM Symposium on Applied Computing*, Fortaleza, Ceará, Brazil, 2008.

- **Leo Ordinez**, David Donari, Rodrigo Santos, Javier Orozco: "The BIDS Software Framework", *X Workshop on Real-Time Systems*, Río de Janeiro, Brasil, 2008.

- **Leo Ordinez**, David Donari, Rodrigo Santos, Javier Orozco: "An Application-Based Real-Time Scheduler for Wireless Sensor Networks", *XXXIV Conferencia Latinoamericana de Informática*, Santa Fe, Argentina, 2008.

- David Donari, **Leo Ordinez**, Rodrigo Santos, Javier Orozco: "Real-Time Server Oriented Operating System for Embedded Applications", *XXXIV Conferencia Latinoamericana de Informática*, Santa Fe, Argentina, 2008.

- **Leo Ordinez**, David Donari, Diana Sánchez, Martín Duval: "A Semantic-Based Scheduling Approach for Soft Real-Time Systems", *XXXIII Conferencia Latinoamericana de Informática*, San José, Costa Rica, 2007.

- Rodrigo Santos, Pedro Doñate, María Belén D'Amico, **Leo Ordinez**, David Donari: "Adaptive Resource Reservation Scheduler",*XII Reunión de trabajo en Procesamiento de la Información y Control*, Río Gallegos, Argentina, 2007.

1.6. Organización

Luego de este capítulo introductorio, donde se presentó el marco general y las motivaciones de esta tesis, el resto de la misma se organiza de la siguiente manera: en el Capítulo 2, se introduce la Política de Planificación Comportamental para sistemas de tiempo real. Esta política contempla en su definición la asignación de prioridades no sólo en base a las características temporales de las tareas, sino también de acuerdo al comportamiento de las mismas y su aporte al funcionamiento general del sistema. En el Capítulo 3, se propone una extensión a una política de manejo de recursos compartidos clásica, que facilita su implementación y provee una nueva visión sobre el tratamiento de recursos compartidos entre tareas de tiempo real. El Capítulo 4 presenta un marco de software genérico para la implementación de las políticas propuestas anteriormente. Este diseño es útil desde el punto de vista práctico, ya que acerca los conceptos teóricos con la aplicación práctica, mediante una herramienta de modelado. Finalmente, el Capítulo 5, expone las conclusiones de esta tesis y plantea las futuras líneas de trabajo que ésta puede tener.

Capítulo 2

Planificación Comportamental

Como se explicó en el capítulo anterior, la planificación de sistemas de tiempo real, consiste en el análisis e implementación de políticas que permitan garantizar que las tareas que componen un sistema respeten y cumplan con sus restricciones temporales. En este sentido, se ha avanzado mucho en el desarrollo de esas políticas. Sin embargo, la gran mayoría de esos enfoques únicamente se centran en aspectos temporales para determinar la asignación de prioridades. En este capítulo, se presenta una nueva política de planificación que, además de considerar las características temporales de las tareas, tiene en cuenta su comportamiento, la función y relevancia dentro del sistema. De esta manera, se logra una planificación más justa y flexible que aquella que se obtiene de sólo contemplar las restricciones temporales. En particular, este capítulo aborda los siguientes temas:

- *Se introduce un nuevo modelo de tarea independiente y de servidor para contemplar aspectos comportamentales.*
- *Se propone la Política de Planificación Comportamental, que planifica tareas y servidores basándose en sus restricciones temporales y sus comportamientos.*
- *Se demuestran propiedades y se brinda un test de factibilidad simple para la política.*
- *Se realiza un exhaustivo análisis comparativo entre la política de planificación presentada y otros enfoques.*

2.1. Motivación

Los sistemas embebidos se han convertido en una parte importante de la vida cotidiana. Desde pequeños dispositivos de entretenimiento como reproductores de MP3, a enormes plantas industriales o satélites están aprovechando la continua investigación en este campo. Los sistemas de tiempo real son sistemas que tienen estrictas restricciones de tiempo en su especificación. En particular, el software para sistemas embebidos suele estar sujeto a restricciones de tiempo que se derivan de la definición misma del problema. Por lo tanto, el software para estos sistemas sufre algunas restricciones adicionales a las que se encuentran en cualquier otro tipo de software. Esto es, no sólo tienen que ser funcionalmente correctos, sino que también tienen que cumplir con ciertas restricciones intrínsecas, que surgen de la definición del problema en sí. Así, el estudio de este tipo de sistemas implica una atención cuidadosa de los requerimientos funcionales y no funcionales a lo largo de toda la vida del sistema. Como se presentó en la introducción de esta tesis, dentro de la teoría de los sistemas de tiempo real hay una rama particular de que se ocupa especialmente del cumplimiento de las restricciones temporales: la teoría de planificación en tiempo real [88]. La piedra angular de esta teoría es el concepto de tarea. De esta manera, un sistema de tiempo real se dice que está compuesto por tareas, que realizan las funciones necesarias para cumplir con los requerimientos del usuario. En términos de Jackson [55], la oración anterior puede ser reescrita como: *una tarea es una especificación de un requerimiento de software extraído de un dominio de aplicación.* De una manera simplista, el conjunto completo de tareas de un sistema responde a la pregunta: ¿Qué hace el sistema?

A menudo, los sistemas de tiempo real están compuestos por tareas duras y blandas. Las tareas duras están sujetas a una planificación que no debe permitir que ningún vencimiento se pierda. Mientras que las blandas pueden perder una cierta cantidad acotada de sus vencimientos. Sin embargo, la perdida de un vencimiento de una tarea blanda no debe afectar el desempeño de las otras, tanto duras como blandas. Con el fin de alcanzar este objetivo, el uso de mecanismos de reserva de recursos (servidores), es una opción conveniente para

planificar estas tareas.

Los mecanismos de reserva de recursos se han implementado con diferentes clases de servidores. Algunos destacados son: el Servidor de Utilización Constante (CUS) [43] y el Servidor de Ancho de Banda Constante (CBS) [6]. Esta idea de servidores se ha utilizado, a la vez, en diferentes aplicaciones específicas: multimedia [6], sistemas de control [31], comunicaciones en tiempo real [74] y otras más generales como en [69]. Sin embargo, en los enfoques basados en servidores mencionados antes, cada tarea se trata sin tener en cuenta los resultados que produce en instancias sucesivas. En todas estas alternativas, la prioridad del servidor que contiene la tarea es estrictamente definida por el conjunto de parámetros del servidor (o la tarea): período, presupuesto o vencimiento. En este sentido, la política de planificación y la asignación de prioridades son funciones exclusivamente del tiempo. Sin embargo, en muchos casos, los resultados producidos por la ejecución de una tarea en una o más de sus instancias pueden ser convenientemente utilizados para cambiar la prioridad de la tarea en instancias siguientes. Así, se reflejaría en la prioridad de modo más fiel la importancia de la tarea en el sistema. Además, este comportamiento de una tarea puede ser aprovechado en el momento de la planificación. Algunos ejemplos en los que algún tipo de planificación basada en el comportamiento puede ser utilizada son: a) La detección temprana de incendios forestales: por lo general, una red de sensores se despliega en el medio ambiente y diferentes variables se sensan para la construcción de un índice de fuego [53]; cuanto mayor es el índice, mayor es la probabilidad de que un incendio se pueda desarrollar en el sitio. Por lo tanto, los incrementos relativos del índice, en un área determinada, conducen a aumentar la prioridad de los sensores en la zona. b) La predicción de tormentas, donde una mayor reducción en la presión barométrica es un claro indicio de una tormenta de viento en desarrollo. Una vez más, los sensores en la zona deben incrementar su prioridad de acuerdo al comportamiento actual. c) Los sistemas de alerta de colisiones en un vehículo: en función de la proximidad a los obstáculos que se detectan, la importancia de la información proporcionada se vuelve más relevante y los sensores involucrados deben aumentar sus prioridades. Con mayor o menor

sofisticación, hay una vasta cantidad de ejemplos en los que la importancia de una variable se incrementa con el valor o la diferencia entre dos medidas sucesivas.

2.1.1. Contribución

Con el objetivo de tomar en cuenta, además de los parámetros temporales, otros referidos al comportamiento de los componentes del sistema de tiempo real y aprovechando las ventajas que ofrecen los mecanismos de reserva de recursos, se propone la Política de Planificación Comportamental. La idea central de esta política es priorizar a aquellas entidades planificables cuyo comportamiento anterior requiera mayor atención. De lo anterior se desprende que al momento de planificar el sistema, se tienen en cuenta no solamente los parámetros temporales sino también una ponderación, establecida por el desarrollador, sobre el comportamiento hasta el momento actual de las entidades. Asimismo, dicha ponderación también puede ser utilizada para estimar el comportamiento futuro de la tarea y, en base a ello, tomar la decisión que corresponda. Desde otro punto de vista, la Planificación Comportamental establece que se debe posponer o adelantar la ejecución de una determinada tarea, en base a su comportamiento y su efecto dentro del sistema.

2.2. Modelo del Sistema bajo la Planificación Comportamental

Bajo la Política de Planificación Comportamental, presentada en este capítulo, se distinguen dos entidades en el sistema: tareas y servidores. Las tareas son definidas por el desarrollador en base a los requerimientos de su aplicación; mientras que los servidores son una entidad abstracta, transparente, que se utiliza para encapsular tareas. Estos servidores son, de hecho, las entidades planificables en el esquema propuesto. A continuación se describen, en detalle, ambas entidades.

2.2.1. Modelo de Tareas

Formalmente, un sistema de tiempo real es un conjunto de tareas periódicas y apropiativas[1] $\mathbf{\Gamma} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. Las tareas presentes en el sistema pueden ser tanto duras como blandas. Ambos tipos de tareas están caracterizadas por los siguientes parámetros temporales: C_i, denotando el tiempo de ejecución de la tarea; T_i, su período de ejecución; a_i, el instante en el cual será reactivada; y el vencimiento relativo D_i, el cual se usa para computar el vencimiento absoluto $d_i = a_i + D_i$. Este último parámetro es el más sensible dentro de la planificación de sistemas de tiempo real, ya que dicha planificación debe garantizar que todas las tareas del sistemas se ejecuten antes de sus correspondientes vencimientos. Como las tareas son periódicas, pueden verse como una secuencia de trabajos o instancias, J_{ij}, donde el primer subíndice se refiere a la tarea y el segundo a la instancia

2.2.1.1. Ponderación del Comportamiento

La característica fundamental, de la política de planificación presentada en este capítulo, es la consideración del comportamiento de las tareas al momento de planificarlas. Como se mencionó anteriormente, las tareas cumplen una función dentro del sistema, tienen un propósito. En este sentido, se entiende al comportamiento de la tarea como aquellas acciones necesarias para cumplir dicho propósito. Asimismo, este comportamiento y su propósito final pueden ser evaluados por el desarrollador desde diferentes perspectivas. Un caso simple consiste en considerar si el propósito fue cumplido o no. Otro más complejo (y asumiendo que el propósito se cumple siempre; es decir, la tarea siempre alcanza su objetivo), es aquel en que se evalúa cualitativamente el comportamiento de la tarea y, por consiguiente, su propósito alcanzado. De este modo, se cuenta con una medida dinámica de la importancia del comportamiento de la tarea en el sistema. Nótese que esto no excluye a aquellas tareas, cuyo comportamiento no puede ser ponderado, ya que el mismo puede verse como constante

[1] En este capítulo de la tesis, se considerará que las tareas son independientes unas de otras. Esto es, no comparten otro recurso que el procesador. Este modelo será ampliado a uno donde se comparten recursos en el capítulo siguiente.

a lo largo del tiempo.

Con el objetivo de ponderar el comportamiento de una tarea, se introduce un nuevo elemento a los descriptos anteriormente, el cual se simboliza como δ_i y es una función del comportamiento de la tarea. La definición de la función δ_i está dada por el desarrollador en base a los comportamientos esperados de las tareas. En un caso particular, se puede pensar a la función como binaria y contemplando únicamente la última instancia de la tarea. En este caso, reflejaría, por ejemplo, una ejecución exitosa o bien una errónea. Desde un punto de vista más general, δ_i podría contemplar más alternativas de ejecución, así como tener en cuenta otras instancias anteriores a la última para realizar su cálculo. En este sentido, la función δ_i puede representar, entre otras, la calidad de la salida producida, la precisión en un cálculo o una estimación del comportamiento futuro. Por ejemplo, podría realizar un promedio dinámico de un cierto número de instancias anteriores, o computar alguna función más elaborada que involucrara una combinación de derivaciones e integraciones de resultados pasados. La computación de la función δ_i es llevada a cabo por la tarea, ya que es la única con capacidad para administrar los datos requeridos.

En base a lo anterior, la función δ_i se puede definir de la siguiente manera:

Definición 2.2.1. Sea $\mathcal{E} = \{\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_p\}$ el conjunto de todos los posibles comportamientos de una tarea, llamado Conjunto Dominio de Ejecución.

Definición 2.2.2. Sea $\mathcal{I} = \{\sigma_1, \sigma_2, \ldots, \sigma_q\}$ un conjunto de valores pertenecientes a $\mathcal{R}$, tales que $\sigma_j \neq \sigma_k$, $\forall j, k$. Este conjunto se denomina Conjunto Imagen de Ejecución.

Definición 2.2.3. Sea $\delta : \mathcal{E} \rightarrowtail \mathcal{I}$ la función que asocia a cada comportamiento posible de ejecución de una tarea, uno y sólo un valor del Conjunto Imagen de Ejecución.

Nota 2.2.1. La definición de la función δ se ha expresado de la forma más general posible, a fin de dejar absoluta libertad al desarrollador para que la pueda definir de acuerdo a las necesidades y particularidades propias de la aplicación que desarrolla.

En sistemas encargados de monitorear y controlar variables físicas, el comportamiento particular de cada tarea está sujeto a las regiones de trabajo *normales* y *anormales* establecidas por el desarrollador. Del estudio previo del sistema a desarrollar, comúnmente se extraen niveles y medidas que son útiles para examinar el desempeño de una tarea en pos del sistema. Esto es, dicho de otro modo, se realiza una ponderación del comportamiento de la tarea en base a los propósitos particulares que persigue. De esta forma, se puede obtener una medida de la importancia que tiene dicha tarea, en el momento actual, dentro del sistema. A continuación se presentan dos ejemplos ilustrativos, a fin de aclarar la definición de la función δ_i en un sistema concreto.

Ejemplo: *Muchos automóviles modernos están equipados con un sistema de alerta de colisiones [19, 59, 5]. En general, estos dispositivos están compuestos por varios sensores y actuadores. El sistema debe encuestar los sensores, procesar la información y actualizar los dispositivos de visualización para el conductor o accionar algún actuador (por ejemplo, unos airbags). En caso de contar con un control crucero, el mismo deberá ajustar sus parámetros, también. A fin de ejemplificar las características de la función δ_i, este ejemplo se centrará únicamente en el análisis de una tarea, que tiene como propósito tomar la información de un sensor de distancia (e.g., radar, lidar, etc.) y procesarla.*

En una ruta, la distancia entre dos autos es un punto crítico para la alerta de colisiones. En Argentina, la ley establece una distancia temporal, entre dos autos, de dos segundos, tomados desde un punto fijo (e.g., un poste de luz). Como se puede inferir, a medida que la distancia visible al objeto detectado por el sensor disminuye (esto es, los dos objetos se acercan), la tarea asociada a dicho sensor cobra mayor importancia dentro del sistema y debe ser atendida más frecuentemente. La ponderación del comportamiento de la tarea y su importancia se hace entonces en base a la distancia al objeto detectado. Y es aquí, donde juega un papel fundamental el criterio del desarrollador, al tener que definir la escala sobre el comportamiento.

En particular, supóngase un auto desplazándose sobre una ruta y otro auto detenido completamente, sobre la ruta, a 200m. Se asume que el sensor utilizado tiene un rango máximo de alcance de 120m. En el Cuadro 2.1 se muestra una posible definición de la función δ_i, en base a la distancia computada por la tarea.

Distancia (m)	δ_i
0 a 19	7
20 a 39	6
40 a 59	5
60 a 79	4
80 a 99	3
100 a 119	2
Mayor a 120	1

Cuadro 2.1: Relación entre distancia al objeto y valor de δ_i

Ejemplo: *Los incendios forestales representan un problema que involucra riesgos de vidas humanas, daños ecológicos y pérdidas económicas de enorme magnitud. El comienzo de un incendio forestal está sujeto a la presencia de una fuente de ignición. Una vez que esa fuente se hace presente, diversos factores ambientales determinan la aparición o no del fuego. Las características de la vegetación, el estado del clima y la topografía son aspectos fundamentales para predecir el inicio y comportamiento (si se hubiera iniciado) del fuego.*

Desde hace algunas décadas, se han elaborado diferentes sistemas para evaluar el riesgo de inicio de fuego, su comportamiento y posibles consecuencias. En particular, el Índice Meteorológico de Incendios (FWI, por la sigla en inglés) Canadiense [29] toma en cuenta la probabilidad de ignición, las características comportamentales (en caso que se desarrolle), las dificultades para controlarlo y los daños que podría causar. En la actualidad, este sistema está siendo portado y adaptado a las características geográficas argentinas [44], [96].

El índice canadiense FWI se compone de seis módulos, cada uno de los cuales indica la contribución de un aspecto específico, a las características del fuego. Este sistema es muy complejo, e incorpora relaciones entre variables meteorológicas, estado de combustibles en el suelo y comportamiento del fuego, para generar distintos indicadores. Estos indicadores

proveen una medida cuantitativa de las dificultades para controlar y el potencial daño que pudiera provocar el fuego.

A fin de ilustrar la definición de la función δ_i, se considera una tarea encargada de computar el índice FWI. Dicha definición se muestra en el Cuadro 2.2.[2]

Clase FWI	**Rango**	**Tipo de Fuego**	**Peligro Potencial**	δ_i
Bajo	0 a 4	Fuego superficial progresivo	Fuego auto-extinguible	1
Moderado	5 a 9	Fuego superficial de bajo vigor	Apagable con herramientas de mano	2
Alto	10 a 19	Fuego superficial ligeramente superficial	Necesidad de bombas y mangueras	3
Muy Alto	20 a 29	Fuego superficial muy intenso	Difícil de controlar	4
Extremo	Mayor a 30	Fuego activo en desarrollo	Situación crítica	5

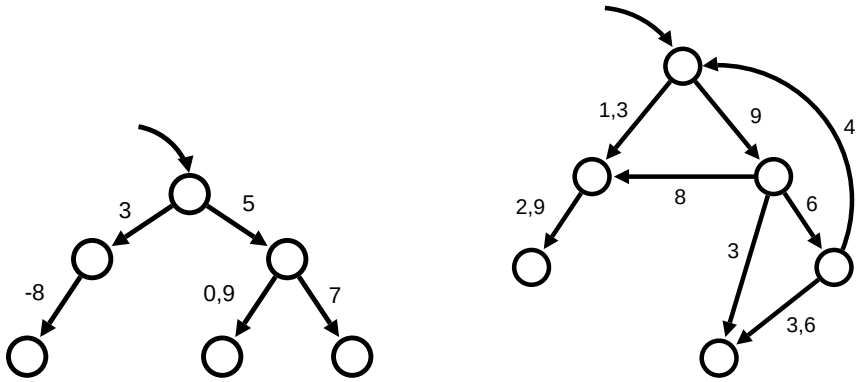
Cuadro 2.2: Definición de δ_i en base al FWI

Una alternativa más concreta y, a la vez, general para ponderar el comportamiento de una tarea consiste en evaluar su ejecución mediante un *grafo de ejecución*. Este tipo de estructura es similar a las presentadas en [14] y [10], aunque aquí se utilizan con otro sentido: el de ponderar el comportamiento. Mientras que en esos trabajos se utiliza para obtener una medida más precisa del tiempo de ejecución de la tarea. De este modo, una tarea es una estructura consistente de vértices o nodos y aristas que los unen, donde cada vértice representa una *computación trascendental*.

Definición 2.2.4. Se define a una computación trascendental de una tarea τ_i como el conjunto de computaciones atómicas, que es plausible de otorgar una medida parcial de la importancia del comportamiento de la tarea τ_i.

En base a lo anterior, se define a una tarea como un grafo dirigido (que puede o no contener ciclos), cuyos vértices son computaciones trascendentales y cuyas aristas están etiquetadas con un valor que representa la ponderación del comportamiento del vértice origen de cada arista. Asimismo, se destacan dos tipos de vértices particulares, presentes en todo grafo: 1) un único *vértice inicial*, por el que comienza la tarea cuando es despachada; y 2) uno o más *vértices finales*, en los que finaliza la ejecución de la tarea. Formalmente:

[2]La clasificación del FWI se tomó de reportes del Servicio Forestal Canadiense [29].



(a) Grafo de Ejecución Acíclico. (b) Grafo de Ejecución Cíclico.

Figura 2.1: Grafos de Ejecución de Dos Tareas Genéricas.

Definición 2.2.5 (Grafo de Ejecución)**.** Un grafo de ejecución Ω_i de una tarea τ_i es una tupla $\langle V, v^o, V^F, E, \rho \rangle$, donde

- V es un conjunto de vértices (computaciones trascendentales)
- v^o es el nodo inicial
- $V^F \subseteq V$ es el conjunto de vértices finales
- $E \subseteq V \times V$ es un conjunto de transiciones entre vértices (aristas)
- $\rho : E \longrightarrow \mathcal{R}$ es una función de peso, que asocia a cada arista un valor real

La Figura 2.1 muestra dos ejemplos de dos tareas τ_i y τ_j y sus correspondientes grafos de ejecución Ω_i y Ω_j.

Se debe remarcar, que el hecho de representar a las tareas como grafos cíclicos no implica de ninguna manera que se contemplen ciclos infinitos en sus ejecuciones. Este comportamiento es inaceptable en sistemas de tiempo real, ya que vuelve al sistema impredecible.

Informalmente, la semántica de ejecución del modelo Ω_i de una tarea τ_i, se describe de la siguiente manera. La ejecución comienza en el nodo v^o, luego que la tarea es enviada a ejecutar. Tras ejecutar la primera computación trascendental en el nodo v^o, la tarea continúa su ejecución mediante una transición no-determinista a uno de los nodos adyacentes a v^o.

Esto es, para un nodo v_p cualquiera, $(v^o, v_p) \in E$. Este proceso continúa desde el nodo v_p hasta alcanzar un nodo $v_q \in V^F$.

A continuación se definen los caminos de ejecución de una tarea y sus comportamientos parciales.

Definición 2.2.6 (Camino de Ejecución)**.** Un camino de ejecución $\omega_k = \langle V', v^o, v_f, E', \rho' \rangle$ es un subgrafo de $\Omega_i = \langle V, v^o, V^F, E, \rho \rangle$ tal que $V' \subseteq V$, $v_f \in V^F$, $\rho' \subseteq \rho$ y $E' = \langle (v^o, v_p), (v_p, v_{p+1}), \ldots, (v_{p+l}, v_f) \rangle \subseteq V' \times V' \subseteq E$.

Definición 2.2.7 (Comportamiento Parcial Ponderado)**.** Un comportamiento parcial ponderado $\eta_k(\tau_i)$ de una tarea τ_i es la sumatoria de los pesos de las aristas que componen un camino de ejecución ω_k. Formalmente, $\eta_k(\tau_i) = \sum_{v^o}^{v_f} \rho((v_p, v_q))$, donde $(v_p, v_q) \in E' \; \forall v$.

En base a las definiciones anteriores, la Definición 2.2.3 se puede reformular de la siguiente manera:

Definición 2.2.8 (Función de Comportamiento)**.** La función de comportamiento de una tarea τ_i, cuyo grafo de ejecución es $\Omega_i = \{\omega_1, \omega_2, \ldots, \omega_u\}$ es el conjunto $\delta_i = \{\eta_1, \eta_2, \ldots, \eta_k\}$ de todos los comportamientos parciales ponderados de la tarea τ_i.

A partir de los ejemplos de la Figura 2.1, las funciones $\delta_{(a)}$ y $\delta_{(b)}$ para cada tarea se muestran en la Figura 2.2(a) y en la Figura 2.2(b), respectivamente. Nótese que los valores de k_1, k_2 y k_3 representan la cantidad de veces que se itera el ciclo correspondiente.

Finalmente, una tarea τ_i bajo el modelo de la Política Planificación Comportamental es una tupla $\tau_i = \langle C_i, D_i, T_i, \delta_i, \Omega_i \rangle$, donde C_i es peor caso de tiempo de ejecución, D_i es el vencimiento relativo, T_i es el período de la tarea o mínimo tiempo entre arribos, δ_i es la función de comportamiento y Ω_i es el grafo de ejecución. Este modelo de tarea, como se puede ver, claramente amplía el modelo clásico presentado en el Capítulo 1.

$$\delta_{(a)} = \begin{cases} \eta_1 = 3 + (-8) = 11 \\ \eta_2 = 5 + 7 = 12 \\ \eta_3 = 5 + 0,9 = 5,9 \end{cases}$$

(a)

$$\delta_{(b)} = \begin{cases} \eta_1 = 1,3 + 2,9 = 4,2 \\ \eta_2 = 9 + 8 + 2,9 = 19,9 \\ \eta_3 = 9 + 3 = 12 \\ \eta_4 = 9 + 6 + 3,6 = 18,6 \\ \eta_5 = k_1(9 + 6 + 4) + 3,6 \\ \eta_6 = k_2(9 + 6 + 4) + \underbrace{9 + 8 + 2,9}_{\eta_2} \\ \eta_7 = k_2(9 + 6 + 4) + \underbrace{9 + 3}_{\eta_3} \end{cases}$$

(b)

Figura 2.2: Definiciones de δ correspondientes a la Figura 2.1

2.2.1.2. Consideraciones de Implementación

A fin de poder implementar de forma efectiva la Política de Planificación Comportamental en un sistema operativo de tiempo real, éste debería proveer, al menos, dos primitivas fundamentales para poder computar la importancia comportamental de la tarea. En primer lugar, es necesario una primitiva que sea capaz de tomar el valor de una computación trascendental, sumarlo a otros previos (que componen el camino de ejecución actual) y almacenarlo. Esto es, permita la construcción del comportamiento parcial ponderado de la instancia actual de la tarea, $\eta_k(\tau_i)$. Por otro lado, es necesario contar con una primitiva que sea capaz de computar el comportamiento parcial ponderado actual de la tarea. Esto es, compute la ponderación del comportamiento de la instancia de ejecución que acaba de terminar la tarea (*i.e.*, su importancia). En el Cuadro 2.3, se muestran dos prototipos de las primitivas descriptas anteriormente.

```
sumarComputaciónTranscendental(r: real)
computarImportancia()
```

Cuadro 2.3: Prototipos de primitivas para ponderar el comportamiento.

2.2.2. Modelo del Servidor Comportamental

El modelo del Servidor Comportamental (BIPS, por la sigla en inglés) comparte los aspectos básicos, comunes a otros mecanismos de reservas. Un BIPS S_s está caracterizado

por un presupuesto o reserva Q_s, que es su cuota de tiempo de procesador disponible por instancia de ejecución. El hecho de nombrar instancia de ejecución establece que el servidor es periódico, siendo P_s su período. Así como las tareas, los BIPS tienen un vencimiento relativo D_s y uno absoluto d_s, que se calcula en base al instante de reactivación r_s, como $d_s = r_s + P_s$. Cada tarea blanda en el sistema, se asume asociada a un servidor comportamental. Mientras que las tareas duras, pueden o no estar asociadas a un servidor. La reserva Q_s del servidor debe estar relacionada al peor caso de tiempo de ejecución de la tarea. Del mismo modo, el período P_s debe ser igual al de la tarea encapsulada. Esto es, $Q_s \geq C_i$ y $P_s = T_i$. El cociente entre los dos parámetros anteriores determina el *ancho de banda* U_s disponible para el servidor. Así, $U_s = \frac{Q_s}{P_s}$ establece la fracción de tiempo del procesador que el servidor posee para ejecutar su tarea. Finalmente, entre los parámetros comunes a varios enfoques basados en servidores, se encuentra q_s, que es el valor actual del presupuesto del BIPS, tal que $0 \leq q_s \leq Q_s$.

Como se mencionó anteriormente, la idea distintiva de la Política de Planificación Comportamental es adelantar o posponer la ejecución de una tarea, de acuerdo al comportamiento e importancia de la misma, dentro del sistema. En esta política, dicho manejo de la frecuencia de ejecución, se realiza mediante una Función de Postergación α_s que caracteriza a cada servidor y es definida de acuerdo al criterio del desarrollador y las particularidades de la aplicación. El resultado de la función afecta a la siguiente instancia de ejecución del servidor y, por consiguiente, de su tarea asociada (esto es, J_{ij+1}). Por definición, se tiene que $\alpha_s(\delta_{ij}) \geq 1 \ \ \forall \delta_{ij}$. A continuación se describe con mayores detalles esta definición.

2.2.2.1. La Función de Postergación

Las tareas duras usualmente están asociadas a partes críticas del sistema, en el sentido de que perder un vencimiento o posponer la ejecución de una de ellas puede traer severas consecuencias a todo el sistema. Por otro lado, las tareas blandas, a pesar de tener un vencimiento que cumplir, están más relacionadas a partes no tan críticas del sistema y

pueden admitir una frecuencia de ejecución variable basada en el estado del sistema. Sobre este hecho, es que la utilización de BIPS se vuelve atractiva. Este mecanismo, basado en la Función de Postergación, permite manejar la frecuencia de ejecución de una tarea de forma sumamente simple y a la vez poderosa.

Como se mencionó anteriormente, la modificación dinámica (en tiempo de ejecución) de la frecuencia con la cual una tarea se ejecuta, se realiza mediante la función α_s. Esta función se define en base a los requerimientos particulares de la aplicación y, en especial, de cada tarea. Sin embargo, puede notarse que la función α_s se aplica sobre los servidores BIPS. Más aún, es un parámetro característico de cada servidor. En este punto, es donde se puede ver el carácter jerárquico que naturalmente impone una planificación basada en servidores, ya que el planificador general (de mayor nivel) es quien elige al servidor con mayor prioridad, y éste es quien despacha a su tarea. De este modo, se refuerza el concepto de aislamiento en el sentido de que la tarea está supeditada a las ordenes de su servidor BIPS asociado. Así, es la tarea quien pondera su comportamiento, retornando el valor δ_{ij}, y es el servidor BIPS quien, en base a dicho valor, calcula el valor de la función α_s. En consecuencia, es el mismo servidor quien decide, una vez finalizada la ejecución de su tarea, cuándo se va a volver a reactivar.

Con todo lo anterior, la función α_s de cada servidor BIPS se define formalmente como:

Definición 2.2.9. Sea $\mathcal{R}^*$ el conjunto de los números reales mayores o iguales a 1. La Función de Postergación se define como $\alpha : \mathcal{I} \rightarrowtail \mathcal{R}^*$, tal que $\exists \sigma_H \in \mathcal{I} \wedge \alpha(\sigma_H) = 1$

Nota 2.2.2. Al igual que la definición de la función δ (de ponderación del comportamiento), la Función de Postergación se ha definido de forma tal de no condicionar al desarrollador y de proveer la generalidad suficiente para su adaptación a distintas aplicaciones.

Es importante destacar, que la definición genérica de la función α_s para cada servidor, provee un mecanismo altamente flexible para determinar la postergación o el adelantamiento de una ejecución. Este mecanismo permite la construcción de una función tan compleja o simple como se requiera en cada caso. Así, por ejemplo, en base al comportamiento actual y

algunos pasados, una función de seguimiento asociada a un radar, puede estimar la posición futura de un objetivo y, en base a ello, determinar la frecuencia de muestreo adecuada [76].

Ejemplo: *A fin de ilustrar la definición de la Función de Postergación, se retomará aquí el ejemplo planteado en la Sección 2.2.1.1, referido a un Sistema de Alerta de Choques. En este caso, se analizará una tarea τ asociada a un servidor BIPS S. En particular, esta tarea toma información proveniente de sensores de distancia y la procesa. Este proceso se repite con un período de 15ms. Nótese que los valores de α_s y del período de la tarea fueron tomados considerando un tiempo máximo necesario para accionar un par de airbags de 50ms. En el Cuadro 2.4 se muestra la definición de la función α_s para este ejemplo. En el cuadro, se puede ver la definición de δ presentada anteriormente, la definición de α_s, el período P original de la tarea y el nuevo período P', calculado a partir de α_s.*

Distancia (m)	δ	$\alpha_\mathbf{s}$	**P (ms)**	**P′ (ms)**
0 a 19	7	1	15	15
20 a 39	6	1,4	15	21
40 a 59	5	1,8	15	27
60 a 79	4	2,3	15	34,5
80 a 99	3	3	15	45
100 a 119	2	4	15	60
Mayor a 120	1	6	15	90

Cuadro 2.4: Definición de α_s

2.2.2.2. Calidad de Servicio

La implementación de un algoritmo de planificación como el que aquí se propone tiene influencia sobre la *calidad de servicio* y la *calidad de aplicación* que tendrán las aplicaciones. Es conveniente entonces distinguir los dos conceptos que son complementarios en la teoría de tiempo real y se refieren a cuestiones diferentes. El primero, está relacionado con la cantidad de vencimientos que es admisible perder. El segundo, en cambio, se refiere a la calidad con la cual un observador aprecia el resultado de la aplicación.

Calidad de Aplicación. La calidad de aplicación es antes que nada una cuestión subjetiva que el observador aprecia como buena o mala en términos generales. Sin embargo, desde el punto de vista de la ingeniería se la puede cuantificar en distintos aspectos que, más allá de lo subjetivo, son tangibles. Por ejemplo, la digitalización de una señal sonora se puede hacer con distintas calidades, de 64 Kbps a 44Mbps [85]. Quien escucha la reproducción luego podrá determinar o no diferencias basado en su propia percepción, sin embargo una mejor digitalización requerirá mayor poder de computo y espacio de memoria para almacenarla y reproducirla. En el mismo sentido, la codificación JPEG de imágenes admite diferentes grados de calidad representado sobre todo en los factores de cuantificación y de compresión. Cuanto mayor es el grado de cuantificación y de compresión, menor es el espacio requerido por la imagen para su almacenamiento y más simple resulta su visualización. Quien observa la imagen luego podrá o no apreciar la diferencia basado en su propia percepción. Diversos ejemplos se podrían agregar en otras áreas de la ingeniería pero se han elegido estos dos pues se comprenden fácilmente.

La calidad de la aplicación está relacionada con el algoritmo de planificación en cuanto la misma determina la complejidad de cálculo involucrada en el procesamiento de la misma. En un esquema jerárquico como el propuesto en esta tesis, como se mencionó anteriormente, un servidor puede verse como un procesador virtual de menor velocidad que el real, donde corren las aplicaciones que encapsula. En base a los parámetros de un servidor (un BIPS, en particular), esa velocidad se calcula como: $U_s = \frac{Q_s}{P_s}$ y se denomina ancho de banda del servidor. De este modo, un incremento (o decremento) del presupuesto Q_s o del período P_s, traerían aparejado una modificación en la velocidad del procesador virtual, sobre el que corre la tarea. Esta variación tiene entonces impacto sobre la potencia de cálculo destinada a las tareas que se ejecutan dentro del servidor. La selección adecuada de los ancho de banda de cada uno de los servidores que componen el sistema está entonces directamente ligada a la calidad de la aplicación que se espera de cada uno de ellos. En la literatura se plantean distintos mecanismos [7, 39, 81, 82] a fin de controlar el ancho de banda asignado a cada

uno de los servidores. Este control se puede hacer tanto sobre la capacidad como sobre el período. En el caso particular de BIPS, el control se realiza sobre el periodo y en base a los resultados que la misma aplicación va entregando.

Calidad de Servicio. La calidad de servicio, como se mencionó antes, está relacionada con la cantidad de vencimientos que la aplicación admite perder sin que por esto deje de funcionar. Se trata entonces de la caracterización tradicional de sistemas de tareas duras o blandas ya presentada en el Capitulo 1. Este concepto está relacionado con la implementación de un esquema de planificación jerárquico como el propuesto en esta tesis en dos niveles. En el primero, el propio servidor que contiene a la aplicación, tiene asociada una calidad de servicio dura. Es decir, el planificador debe garantizar que el servidor cumple con todos sus vencimientos. Si se admitiese la pérdida de vencimientos por parte del servidor, entonces prácticamente nada se podría garantizar de las aplicaciones contenidas en él. En el segundo nivel, el servidor debe garantizar el cumplimiento de los vencimientos de las tareas asociadas a él con cierta calidad de servicio (que podrá ser dura o blanda), dependiendo del tipo de aplicación que se trate. Para este requerimiento la selección del presupuesto y el período del servidor resultan cruciales. En el caso de ser dura, el presupuesto Q_s debe ser igual al peor tiempo de ejecución posible de la tarea mientras que el período P_s debe ser igual al mínimo tiempo entre dos activaciones sucesivas de la tarea.

La utilización de un BIPS está ligada a tareas y/o aplicaciones en cuyo funcionamiento se contemplen variaciones en el período de activación de la misma. En este sentido la función de postergación del servidor, α_s, define el siguiente momento de activación de la tarea en base a los resultados obtenidos o el comportamiento alcanzado. Esto supone que la tarea admite una frecuencia variable en su ejecución sin que por esto signifique que se pierden vencimientos. Por ejemplo, una tarea de control que trabaja sobre una señal analógica que es muestreada con una determinada frecuencia, puede variar esta frecuencia en base a diferentes factores tanto internos como externos. Dentro de los primeros se pueden considerar condiciones de

estabilidad del sistema que tornan los cálculos repetitivos. Entre los segundos, se puede mencionar el caso del ejemplo presentado anteriormente sobre un sistema de Alerta de Choques, en él si el vehículo se encuentra detenido o a muy baja velocidad, el sistema de monitoreo de la distancia podría ser desactivado o atendido con una baja frecuencia independientemente de la distancia al obstáculo detectado.

Elegibilidad. En la misma línea de razonamiento que se plantea en el párrafo anterior, la funcíon α_s puede verse como un condicionante de la elegibilidad de la tarea. En este sentido, las tareas duras requieren atención siempre y en cada período de activación, mientras que las blandas pueden saltearse algunas instancias cuando la dinámica del sistema lo determine. El factor de postergación establecido por la función α_s actúa entonces como una condición de elegibilidad para las instancias subsiguientes de la tarea. Esto es, cuando se incrementa el período del servidor asociado a una tarea, la tarea pierde importancia dentro del sistema, ya que tanto su período de activación como su vencimiento se extienden. Con esto, la tarea se vuelve no elegible dentro del sistema. Este comportamiento puede ser aprovechado en sistemas que involucran magnitudes con un alto grado de inercia y donde no se dan cambios abruptos, con lo cual se puede reducir la frecuencia de muestreo sin afectar la calidad del control. Un ejemplo claro de esta clase de sistemas lo constituye el de Alerta Temprana de Incendios, presentado anteriormente.

2.3. Algoritmo de Planificación Comportamental

La Política de Planificación Comportamental introduce, mediante el uso de servidores BIPS, un esquema jerárquico para planificar las tareas del sistema. Así, el SOTR planifica los servidores BIPS en EDF y los servidores despachan a las tareas. Por otro lado, a fin de conseguir una distribución justa del ancho de banda total del procesador entre los diferentes servidores, se impone el esquema de *reserva dura* [78]. Éste esquema establece que una vez que el presupuesto de un servidor se agotó, dicho servidor no puede volver a planificarse

hasta tanto su presupuesto no se haya recargado. Además, dicho tiempo entre recargas debe tener una duración de, al menos, el tiempo restante al vencimiento del servidor. En el caso del BIPS, la reserva dura se introduce por medio de una espera diferenciada y calculada dinámicamente para la recarga del presupuesto. Con esto en mente, en cada instante de tiempo, un servidor BIPS puede estar en sólo uno de los estados mostrados en la Figura 2.3. A continuación se detalla el significado de cada uno de esos estados.

ACTIVO: Hay al menos una instancia de una tarea lista para ser ejecutada y $q_s > 0$.

INACTIVO: No hay instancias de ejecución pendientes.

ESPERANDO: El presupuesto del servidor está agotado y hay al menos una instancia de ejecución pendiente. El tiempo de espera se define en base al valor retornado por la función α_s asociada al servidor y a la tarea.

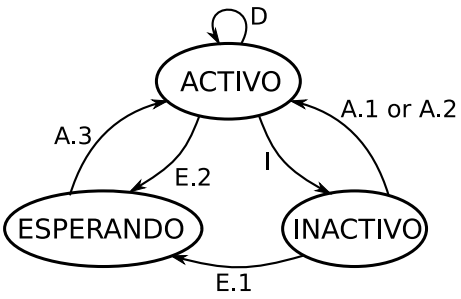


Figura 2.3: Estados del BIPS bajo la Planificación Comportamental.

El servidor BIPS se basa en una serie de reglas simples, las cuales se describen siguiendo la siguiente convención: **A** representa una transición al estado ACTIVO; **E** una al estado ESPERANDO; **I** al INACTIVO y **D** indica un decremento en el presupuesto del servidor. En este sentido, las reglas están también numeradas para distinguir la situación en que son aplicadas. A fin de presentar estas reglas en la forma más clara posible, se introduce la variable *num* que indica que hay instancias de ejecución pendientes y el servidor BIPS está en el estado ESPERANDO. En consecuencia, el servidor estará en el estado ESPERANDO si $num > 0$. Como en lo que sigue todo se refiere a una tarea τ_i, la notación será simplificada

eliminando el doble subíndice a las instancias de ejecución de dicha tarea. Entonces, la j-ésima instanciación de la i-ésima tarea τ_i se denotará J_j, en lugar de J_{ij}.

A: El servidor BIPS tiene un presupuesto actual q_s suficiente para ejecutar al menos una instancia de ejecución. Se realiza una transición al estado ACTIVO, bajo las siguientes condiciones:

A.1: Si una instancia J_j arriba en el tiempo $t = a_j$ y el BIPS está en el estado INACTIVO y $(t \geq d_s - q_s\alpha_s\frac{P_s}{Q_s})$, luego $q_s \leftarrow Q_s$, $d_s \leftarrow t + \alpha_s P_s$ y $r_s \leftarrow t$

A.2 Si una instancia J_j arriba en el tiempo $t = a_j$ y el BIPS está en el estado INACTIVO y $(t < d_s - q_s\alpha_s\frac{P_s}{Q_s})$ y $d_s \geq t$ y $q_s \neq 0$, luego la instancia es atendida con el presupuesto del servidor disponible actualmente, el vencimiento no se modifica y $r_s \leftarrow t$

A.3: Si $(num > 0)$ y $(t \geq r_s)$, luego $num \leftarrow num - 1$, $q_s \leftarrow Q_s$, $d_s \leftarrow t + \alpha_s P_s$ y $r_s \leftarrow t$

E: Cuando el presupuesto del servidor BIPS se agota y tiene instancias de ejecución pendientes, espera una cantidad de tiempo α veces su período para que se recargue. Se realiza una transición al estado ESPERANDO. A continuación se muestran los casos especiales en que se hace esa transición.

E.1: Si una instancia J_j arriba en $t = a_j$ y el servidor BIPS está en el estado INACTIVO y $(t < d_s - q_s\alpha_s\frac{P_s}{Q_s})$ y $d_s < t$ y $q_s = 0$, luego $num \leftarrow num + 1$ y $r_s \leftarrow d_s + \alpha_s P_s$

E.2: Si el servidor BIPS S_s está ejecutando la instancia J_j y $q_s = 0$, luego $num \leftarrow num + 1$ y $r_s \leftarrow d_s + \alpha_s P_s$

D: Cuando un servidor BIPS ejecuta una instancia de una tarea por una unidad de tiempo, decrementa su presupuesto disponible apropiadamente, esto es $q_s \leftarrow q_s - 1$

I: Cuando una instancia de ejecución termina y no hay otras pendientes, el servidor BIPS pasa al estado INACTIVO. En otro caso, permanece en el estado ACTIVO.

2.4. Propiedades

En esta sección, se presentan las propiedades referidas a la factibilidad de un sistema basado en la Política de Planificación Comportamental. Estas propiedades establecen las condiciones bajo las cuales una tarea contenida o no, por un servidor, es factible cuando se utiliza el esquema jerárquico propuesto.

Teorema 2.4.1 (Propiedad de Aislamiento)**.** *Un servidor BIPS, con parámetros* (Q_s, P_s, α_s) *utiliza un ancho de banda* U_s *máximo de* $\frac{Q_s}{P_s}$

Demostración. Las reglas del servidor BIPS (E.1 y E.2) indican claramente, que los instantes de recarga del presupuesto del servidor están separados $\alpha_s P_s$, y por las reglas A.1, A.2, A.3 y D, en cada intervalo entre recargas, el servidor puede ejecutar sólo Q_s unidades de tiempo. El ancho de banda, de un servidor, se define como la relación entre el presupuesto de tiempo del BIPS y su período. De lo anterior, surge claramente que el ancho de banda de un servidor está acotado superiormente por $\frac{Q_s}{P_s}$, ya que en el peor caso, α_s tomará el valor mínimo expresado en la Definición 2.2.9, esto es 1, y el servidor puede ejecutar a lo sumo Q_s unidades de tiempo de P_s. En el caso que $\alpha_s > 1$ para toda instancia de ejecución, por las reglas E.1 y E.2, habrá un intervalo de recarga más largo (*i.e.*, $r_s = d_s + \alpha_s P_s$) y el ancho de banda estará acotado por $\frac{Q_s}{\alpha_s P_s}$. Sin embargo, en el caso general, habrá una variedad de instancias de ejecución con diferentes valores de α_s, así aplicando la propiedad de superposición, todo el ancho de banda estará acotado por $\frac{Q_s}{P_s}$. □

Teorema 2.4.2 (Propiedad de Factibilidad)**.** *Dado un conjunto de* v *tareas independientes con factor de utilización total* $U_T = \sum_{k=1}^{v} \frac{C_k}{T_k}$ *y un conjunto de* w *servidores BIPS con factor de utilización total* $U_{BIPS} = \sum_{s=1}^{w} \frac{Q_s}{P_s}$ *planificados por EDF, luego ni los servidores ni las tareas perderán un vencimiento si y sólo sí* $U_T + U_{BIPS} \leq 1$

Demostración. En el esquema jerárquico de la Política de Planificación Comportamental, la política de nivel superior establecida es EDF. En [65] se demostró que la condición de

factibilidad necesaria y suficiente para un sistema operando bajo EDF es que tenga un factor de utilización menor o igual a 1. Como cada BIPS puede verse como una tarea especial con un factor de utilización igual a su ancho de banda, todo el sistema se reduce a la conocida condición de Liu y Layland para EDF. En consecuencia, ni las tareas independientes ni los servidores perderán vencimientos.

La recíproca del teorema se demuestra en forma análoga. □

Teorema 2.4.3 (Propiedad de Factibilidad de Tareas Duras)**.** *Dada una tarea dura τ_i con parámetros C_i, D_i y T_i, contenida en un servidor BIPS con parámetros Q_s, D_s y P_s, tal que $C_i \leq Q_s$, $D_i = D_s$, $T_i = P_s$ y $\alpha_s = 1$ para cada instancia de ejecución J_{ij} de τ_i, luego la tarea será factible si y sólo si es factible por EDF.*

Demostración. Como la tarea τ_i es dura, el intervalo entre las activaciones de sus sucesivas instancias de ejecución está dado por su período (o mínimo tiempo entre arribos), el cual es, por diseño, igual al período del servidor BIPS. La condición $C_i \leq Q_s$ le asigna al servidor el presupuesto suficiente para completar la ejecución de cada instancia sin posponer su vencimiento. Al ser una tarea dura, el valor de la función α_s es constante e igual a 1. Por esto, el vencimiento generado por las reglas de la Planificación Comportamental es $d_s = r_s + P_s$; que es, de hecho, el mismo vencimiento de la tarea. Por el Teorema 2.4.2, el servidor no perderá ningún vencimiento, y en consecuencia tampoco lo hará la tarea encapsulada en él.

La recíproca del teorema se demuestra análogamente. □

2.5. Comparación Conceptual con Otros Métodos de Planificación

La Política de Planificación Comportamental, presentada a lo largo de este capítulo, propone un nuevo enfoque al momento de planificar un conjunto de tareas con restricciones temporales. Este nuevo punto de vista se basa en la consideración no sólo de las características

temporales de las tareas, sino también de la importancia, que su comportamiento tiene en el sistema. Este comportamiento y su importancia están definidos por el desarrollador y pueden representar la calidad de la salida producida por la tarea, una estimación del comportamiento futuro de la misma, la precisión o error en un cálculo realizado, entre otras. Aunque los conceptos introducidos por la Política de Planificación Comportamental son originales en cuanto a considerar la importancia del comportamiento de las tareas en el sistema y, en base a ello, manejar su frecuencia de ejecución; en el área de investigación de los sistemas de tiempo real se han presentado algunos trabajos que guardan cierta similitud con lo que aquí propuesto. En el punto específico de considerar, además de la prioridad, la importancia de la tarea al tiempo de planificar, se tienen los trabajos presentados en [60], [61] y [57]. En cuanto a la variación dinámica del período de las tareas y realizar una planificación bajo EDF, se destacan los aportes realizados en [23, 25, 34, 26]. Por otro lado, el modelo de computación imprecisa fue propuesto en [36] y extendido en [11]. Finalmente, existe una línea de investigación que comienza con la tesis de Locke [66] en la cual se proponen soluciones para el problema de planificar tareas en tiempo de ejecución con el mejor esfuerzo. Sobre esta línea se encuentran los trabajos [38, 40, 64, 13]. Una derivación del trabajo de Locke la constituyen los trabajos, que tratan el tema de planificación basada en valor. En general, este último enfoque caracteriza a las tareas en base a una ganancia que le aportan al sistema si son completadas. Así, los trabajos mencionados buscan formas alternativas de maximizar esa ganancia.

A continuación se contrastan conceptualmente los trabajos referidos a planificación basada en valor, con la Política de Planificación Comportamental, a fin de mostrar las diferencias, similitudes, ventajas y desventajas de cada enfoque.

Como se ha mencionado, la decisión de que una tarea ejecute o en un determinado tiempo la toma el planificador en base a la prioridad de la tarea en ese momento. Dicha acción no toma en cuenta la importancia de la tarea en el sistema, la cual puede ser distinta de la prioridad. Esto es, se podría dar el caso de que una tarea tenga alta prioridad pero no sea

importante para el funcionamiento del sistema, y viceversa. En este sentido, Kosugi *et ál* [61] proponen un esquema de planificación basado en prioridades fijas, donde cada tarea tiene una *importancia semántica*:

> La importancia de la tarea es la importancia semántica de la tarea representada por un número natural. Se puede establecer por el usuario. Cuanto menor es el número, mayor es la importancia [...] Entonces, si el número de tareas es n, el nivel de importancia es también n, y no hay tareas que tengan el mismo nivel de importancia.

De lo anterior se ve claramente que la importancia de la tarea en el sistema es un valor fijo a lo largo del tiempo. El esquema de planificación lo que hace es escalar la utilización del procesador de acuerdo a la importancia de las tareas, y luego ajusta el factor de utilización de cada tarea en base a su importancia, sin modificar su período (*i.e.*, su prioridad se mantiene inalterada). Esto se realiza en cada arribo de tareas. Por otro lado, Kalogeraki *et ál*, en [60], aplican una visión similar de importancia fija a un sistema distribuido de tiempo real blando orientado a objetos. Los autores consideran que cada tarea tiene una *importancia* que representa la criticalidad de la tarea para el diseñador (el valor es fijo). A la vez, cada tarea es una secuencia de invocaciones a objetos, los cuales también tienen una importancia propia, independiente de la de la tarea que los invocó. Con estas consideraciones, el aporte fundamental del trabajo es el diseño de un planificador local (en cada nodo del sistema distribuido) que despacha objetos de acuerdo a su importancia. Este planificador asegura que un objeto de alta importancia se ejecutará antes que uno de baja, si se determina que las tareas que los invocaron cumplirán sus vencimientos. Finalmente, el último esquema de planificación que se analizará en lo referido a contemplar la importancia de las tareas, es el trabajo de Jin *et ál* [57]. En este trabajo, los autores plantean un sistema con dos tipos de aplicaciones: multimediales (MM) y de no tiempo real (NR). Cada tipo de aplicación está, a la vez, compuesta por tareas. Las tareas tienen un *nivel de importancia* de acuerdo a su tipo (*e.g.*, críticos interactivos). Asimismo, las clases de aplicaciones tienen un *parámetro de*

importancia, el cual se determina en base a la cantidad de tareas con alto nivel de importancia que las componen. El planificador (llamado IMAC) propuesto por los autores se divide en dos partes:

1. Actualiza el valor de importancia de la clase correspondiente (MM o NR) cuando arriba o finaliza una tarea perteneciente a dicha clase.

2. Se realimenta la información de las clases para ajustar el tiempo de procesador que se le asignará a cada una.

Aunque la idea es interesante, como afirman los autores, el mecanismo no es lo suficientemente sofisticado como para contemplar situaciones que pueden perjudicar el desempeño de aplicaciones de la clase MM (por ejemplo, cuando una aplicación de la clase NR consume mucho tiempo del procesador).

En referencia a la Política de Planificación Comportamental, a pesar de que los tres aportes anteriores analizados (esto es, [61, 60, 57]) consideran un valor de importancia, además del de prioridad, al momento de planificar, ninguno de ellos lo hace en forma dinámica y teniendo en cuenta el comportamiento de las tareas. Asimismo, el hecho de establecer la importancia de manera fija, no permite aprovechar todas las ventajas de planificación que esa propiedad podría brindar.

Un modelo de sistema que dista bastante del clásico planteado por Liu y Layland [65], es el de *planificación basada en recompensa*, propuesto por Chung *et ál*, en [36]. Este modelo, también llamado de *computación imprecisa*, fue extendido por Aydin *et ál*, en [11]. Básicamente, el enfoque plantea que una tarea se divide en dos partes: una mandatoria y una opcional. La parte mandatoria se asume que produce un resultado aceptable; mientras que la opcional, mejora dicho resultado. Cada ejecución completa de una parte opcional, a la vez, provee una recompensa al sistema. En [11] se propone una solución óptima que maximiza la recompensa total del sistema.

El modelo de computación imprecisa, aunque completamente diferente al planteado por

la Política de Planificación Comportamental, guarda una cierta relación con este último en el hecho de considerar la calidad de la salida producida por una tarea al momento de planificarla. Sin embargo, una comparación más profunda sería injusta ya que que ambos modelos son, por definición, distintos. Mientras que el de computación imprecisa divide el tiempo de ejecución de cada tarea en dos partes y toma decisiones de planificación únicamente sobre la segunda; en el de Planificación Comportamental, cada tarea se compone de un único tiempo de ejecución y las decisiones se toman en base al comportamiento de la misma.

Además de ponderar el comportamiento de una tarea, la Política de Planificación Comportamental introduce el concepto de función de postergación. Esta función ajusta dinámicamente la frecuencia de ejecución de cada tarea en base al comportamiento de la misma. La idea de ajustar en forma dinámica la frecuencia de ejecución de las tareas es también utilizada por Buttazzo *et ál* [23]. En ese trabajo, que luego fue ampliado por ese mismo grupo de investigación en [26] y [25], se plantea el *modelo de tarea elástica*. Este modelo tiene su base en una analogía, que realizan los autores con el modelo clásico del resorte, presente en cualquier curso introductorio de física. Bajo este modelo "cada tarea está caracterizada por cinco parámetros: un tiempo de computación C_i, un período nominal T_{i_o}, un período mínimo $T_{i_{\text{mín}}}$, un período máximo $T_{i_{\text{máx}}}$, y un coeficiente de elasticidad $e_i \geq 0$, el cual especifica la flexibilidad de la tarea para variar su utilización, a fin de adaptar el sistema a una nueva configuración factible". Los tres trabajos mencionados, donde se presenta y estudia el modelo de tarea elástica, comparten la misma idea básica al momento de planificar las tareas. Tanto las tareas por sí mismas, como el planificador pueden ajustar la frecuencia de ejecución. Para esto se proponen algoritmos de compresión y descompresión. En ambos casos, los algoritmos se pueden ejecutar efectivamente si su resultado es una planificación factible. Los autores expresan que la flexibilidad de este modelo de tareas se basa en el coeficiente de elasticidad de las tareas. Sin embargo, este coeficiente es fijo para cada tarea y no se aclara en base qué condiciones debe determinarse. Por otro lado, Chen y Doi en [34] ampliaron aún más este enfoque al incluir en las funciones de compresión y descompresión una realimentación

dinámica, sin tratar el tema del coeficiente de elasticidad, que sigue siendo la base de la política.

Tanto el modelo de tarea elástica [23, 26, 25, 34], como el propuesto por la Política de Planificación Comportamental, tienen la misma idea de base, que es ajustar dinámicamente la frecuencia de ejecución de las tareas, variando, a la vez, su prioridad entre dos instancias de ejecución sucesivas. Sin embargo, se puede ver que el modelo planteado en esta tesis es más general que el de tarea elástica y lo incluye. Además, el hecho de realizar la variación de la frecuencia de acuerdo a una función que pondera el comportamiento y proveer otra función que realiza el ajuste del período en base a dicho comportamiento, resulta en un esquema no sólo más flexible, sino también más adaptable a diferentes alternativas de ejecución. En este sentido, la Figura 2.4 muestra cómo los parámetros de un servidor BIPS de la Política de Planificación Comportamental se ajustan a los propuestos por el modelo de tarea elástica:

Parámetro	**Descripción**
P_s	Período del servidor BIPS
α_o	Valor nominal de postergación
$\alpha_{\text{mín}}$	Valor mínimo de postergación
$\alpha_{\text{máx}}$	Valor máximo de postergación

(a) Descripción de parámetros.

$$
\begin{aligned}
T_{i_{\text{mín}}} &= \alpha_{\text{mín}} P_s \\
T_{i_o} &= \alpha_o P_s \\
T_{i_{\text{máx}}} &= \alpha_{\text{máx}} P_s
\end{aligned}
$$

(b) Transformaciones de parámetros.

Figura 2.4: Adaptación entre los modelos BIPS y de Tarea Elástica.

Un enfoque fundamental, y de enorme influencia posterior en la teoría de sistemas de tiempo real, es el introducido por Locke, en 1986, en su tesis doctoral [66]. Locke plantea que para un sistema de n tareas, existen $n!$ posibles maneras de organizarlas (planificarlas). En este sentido, el autor propone que cada tarea tenga asociada una función de *valor* $V_i(t)$, que representa la ganancia que aporta la tarea τ_i al sistema, en el tiempo t. La política propuesta, llamada de *mejor esfuerzo*, intenta maximizar la ganancia del sistema cuando se toma una decisión de planificación (de aquí, que la función de valor sea dependiente del tiempo). En consecuencia, la política de mejor esfuerzo es altamente dinámica y su definición se reduce a un problema de optimización matemática. Asimismo, desde un punto de vista práctico,

las políticas de mejor esfuerzo generalmente se dividen en dos: una *política de admisión*, que determina si la tarea recién arribada puede ser aceptada o no; y una *política de despacho* que realiza la planificación efectiva del conjunto de tareas.

La línea de investigación iniciada por Locke, en políticas de planificación de mejor esfuerzo (también llamadas basadas en *valor* o utilidad), fue continuada por un gran número de investigadores, que se han concentrado en distintas aristas del enfoque[3]. Ronen *et ál*, en [80], hacen una analogía entre el problema de planificación de tareas y el de *deliberación-planificación* de agentes autónomos en el área de la inteligencia artificial. En [38], Cortes *et ál* se enfocan, en la utilización de un algoritmo de mejor esfuerzo estático, en un sistema monoprocesador compuesto por tareas duras y blandas. En particular, los autores proponen una serie de heurísticas para maximizar la ganancia generada por las tareas blandas, que no lleve a la pérdida de vencimiento de las tareas duras. Previo al trabajo de Cortes *et ál*, Davis *et ál*, en [40], trabajan también sobre un conjunto mixto de tareas. En este caso, tareas duras y tareas opcionales. Las primeras, se asume que se planifican por prioridades fijas. Con respeto a las segundas, se supone que arriban sólo una vez. Sobre estas últimas se realiza la planificación de mejor esfuerzo. Los autores dirigen su estudio al control de un vehículo autónomo. En el trabajo, se propone una nueva política de admisión, que se basa en la idea de que ejecutar una tarea opcional de bajo valor es contraproducente. En cambio, es mejor guardar ese espacio de tiempo para otras tareas opcionales que puedan llegar en el futuro. Un punto débil de este enfoque es que la ganancia que aporta una tarea opcional se determina cuando esta arriba. Sin embargo, esta determinación (que los autores explican que aporta flexibilidad al enfoque), no se analiza ni se dan criterios de asignación o bases para definir esa ganancia. Un área crítica en la utilización de políticas de mejor esfuerzo es la sobrecarga impuesta al sistema operativo en la ejecución del algoritmo de planificación. En este sentido, Chen y Xia, en [32], proponen una idea interesante para contemplar dicha sobrecarga al

[3]La lista de trabajos es muy extensa. Aquí se presentan los más representativos para contrastarlos con la Política de Planificación Comportamental. Este análisis se puede completar con las comparaciones hechas en [21] y [64].

momento de planificar y evitarla. Por otro lado, Carlson *et ál*, en [30], atacan el problema de balancear la carga de un sistema distribuido de tiempo real, mediante la migración de tareas desde un nodo sobrecargado a otro que no lo está. Para esto, proponen una política de admisión en los nodos que rechaza aquellas tareas que causan sobrecarga y las migra a otros nodos. La complejidad del algoritmo de admisión es claramente NP. Sin embargo, los autores proponen una heurística que vuelve su complejidad a un orden polinomial. Otro aporte diferente a la línea de investigación iniciada por Locke, es el que realizan Aldarmi y Burns, en [8]. Allí, los autores retoman el tema de planificar tareas blandas en un sistema monoprocesador. El aporte significativo del trabajo es que los autores establecen explícitamente cómo determinar la prioridad de cada tarea dinámicamente, en base a la utilidad de la misma en ese tiempo. Sin embargo, esa utilidad sigue siendo una función del tiempo y tiene un punto crítico en el vencimiento, con lo cual todo el enfoque sigue estando basado en características temporales, sin tener en cuenta otras. En [13], Banachowski *et ál* presentan el Servidor de Ancho de Banda de Mejor Esfuerzo (BEBS, por la sigla en inglés). Éste es un servidor aperiódico que ajusta su período basado en el comportamiento, en tiempo de ejecución, de las tareas; y que se conforma por tres componentes: un servidor aperiódico, un procedimiento de reclamo de tiempo libre (en inglés, *slack*) y un algoritmo para determinar los parámetros apropiados del servidor. Asimismo, los autores distinguen tres tipos de tareas en el sistema: tareas de computación intensa, que están la mayor parte del tiempo listas para ejecutar; tareas interactivas, que están mucho tiempo bloqueadas esperando entradas de usuarios o de dispositivos; y tareas periódicas, que tienen un comportamiento de tiempo real blando. El algoritmo que ajusta los parámetros del servidor lo hace mediante la observación de tasas de ejecución previas e infiriendo si las tareas son interactivas, de computación intensa o periódicas. De esta forma, la variación del período se basa en la clase de tarea y en lo que consumió de presupuesto en sus instancias previas (*i.e.*, un parámetro temporal). La diferencia fundamental con la Política de Planificación Comportamental es que ésta propone un enfoque más general, que puede tener en cuenta instancias previas, así como

predicciones del futuro y no solamente aspectos temporales, sino específicos del comportamiento computacional de cada tarea. En [64], Li y Ravindran retoman la línea original de Locke y proponen varias mejoras a algoritmos bien conocidos. En particular, se centran en la complejidad computacional de los algoritmos y presentan una serie de heurísticas que los llevan en el peor de los casos, a un orden $\mathcal{O}(n \log(n))$. Por otro lado, un estudio comparativo de gran utilidad es el que realizan Buttazzo *et ál*, en [22]. Allí los autores comparan 12 algoritmos bien conocidos, en situaciones de sobrecarga, mediante extensas simulaciones. Las comparaciones se realizan entre algoritmos que se basan en utilidad y otros que sólo tienen en cuenta el vencimiento de las tareas *i.e.*, variantes de EDF. El desempeño de todos se mide en términos de la suma total de tareas que terminan efectivamente antes del vencimiento. Para finalizar este análisis de políticas de planificación de mejor esfuerzo, es importante mencionar los trabajos de Burns *et ál* [21] y de Welch y Brandt [97]. En ambos se realizan amplias y criteriosas discusiones sobre varios aspectos conceptuales de estas políticas. En particular, de [21], se pueden extraer la siguientes conclusiones:

- Los enfoques basados en utilidad son para sistemas dinámicos, ya que utilizan información disponible en tiempos de ejecución.
- En general, las políticas se dividen en: 1) admisión y 2) planificación.
- En la admisión es donde se aplican técnicas de optimización matemática, las cuales son computacionalmente costosas.
- La *utilidad* de las tareas generalmente se asigna en forma estática. Las asignaciones dinámicas son costosas.
- La determinación de la utilidad se suele hacer en base a un criterio temporal.

Por otro lado, Welch y Brandt analizan tres áreas donde una política de planificación basada en utilidad se puede aplicar: multimedia, defensa aérea y sistemas de tiempo real cooperativos en nivel empresarial. Asimismo, la contribución fundamental del trabajo es

el estudio y clasificación del *beneficio*, en las políticas de mejor esfuerzo. Así, los autores distinguen tres niveles de abstracción para determinar el beneficio: en función del tiempo, del tiempo y los recursos necesarios; y de la calidad de la salida. En este último punto, donde menos investigación se ha hecho, los autores citan el trabajo de Brandt y Nutt [20], en el cual el beneficio es establecido por el usuario de la aplicación en base a un criterio personal. Este criterio, aunque diferente al del resto de los trabajos en este tema, sigue siendo menos abarcativo y poderoso que el propuesto por la Política de Planificación Comportamental, presentada en esta tesis.

A modo de conclusión de esta sección, se puede afirmar, que el análisis de más de 20 trabajos de investigación de muy diversos enfoques, reafirma la originalidad de la Política de Planificación Comportamental, en el sentido que ninguno de ellos contempla, al momento de planificar, alguna información relacionada al comportamiento computacional de las tareas.

2.6. Síntesis del Capítulo

Como se expuso en este capítulo, gran parte de las políticas de planificación se basan en las características temporales de las tareas para la toma de decisiones. A pesar de esto, aquí se fundamentó la necesidad de tomar en cuenta el comportamiento, para determinar la importancia de las mismas y brindar una planificación más precisa. Por otro lado, la naturaleza crítica y altamente dinámica de los sistemas de tiempo real motivó la utilización de mecanismos basados en servidores sobre prioridades dinámicas. Con todo, en este capítulo, se presentó la Política de Planificación Comportamental, que posee las siguientes características principales: pondera el comportamiento de las tareas y lo utiliza para determinar su prioridad, es dinámica y utiliza servidores como medio de protección entre tareas. En particular, se realizaron los siguientes aportes:

- Se desarrolló un nuevo modelo de tareas de tiempo real y de servidor que las encapsula, los cuales proveen las bases para ponderar el comportamiento, determinar su

importancia y ajustar su prioridad.

- Se desarrolló una nueva política de planificación dinámica para sistemas de tiempo real, denominada Política de Planificación Comportamental, que contempla las características temporales y el comportamiento de las tareas, al momento de asignar prioridades.

- Se demostraron propiedades de la política formulada y se presentó un test de factibilidad simple y de bajo costo computacional.

- Se realizó un extenso análisis comparativo de los conceptos propuestos por la Política de Planificación Comportamental y otros enfoques.

Capítulo 3

Recursos Compartidos en Tiempo Real

El tratamiento de recursos compartidos entre diferentes tareas dentro de un sistema de tiempo real es un tema sensible para el área de planificación. Las políticas, en estos casos, se vuelven más complejas y hasta pesimistas, ya que se deben tener recaudos extra para evitar situaciones indeseadas. En este sentido, el presente capítulo propone una política de manejo de recursos compartidos que se adapta a las políticas clásicas de planificación de tareas independientes, de forma simple y a bajo costo computacional. Además, el enfoque propuesto es directamente aplicable junto a la Política de Planificación Comportamental, presentada en el capítulo anterior. En particular, este capítulo desarrolla los siguientes puntos:

- *Se mejora un esquema de manejo de recursos compartidos clásico, al introducir conjuntos con relaciones de bloqueo.*
- *Se propone una política de manejo de recursos compartidos eficiente, que sólo contempla en el análisis a aquellas tareas que potencialmente pueden tener conflictos de recursos.*
- *Se exponen aspectos prácticos a tener en cuenta al momento de implementar un sistema basado en esta nueva formulación de la política.*
- *Se muestra que esta política puede coexistir y complementarse efectivamente con otras bien conocidas e incluso con la Política de Planificación Comportamental.*
- *Se realiza un análisis comparativo detallado con otros enfoques que toman la misma base que el aquí expuesto.*

3.1. Motivación

En un sistema computacional generalmente existen recursos que son necesarios para la concreción de los objetivos del sistema. Estos recursos pueden ser de hardware (periféricos) o de software (variables, señales, etc.). En un ambiente multiprogramado, donde el tiempo de uso del procesador es dividido entre las diferentes entidades de ejecución, los recursos suelen ser compartidos por varias de esas entidades. Las tareas de un sistema de tiempo real, por tener una fuerte interacción con el ambiente, usualmente manejan una importante cantidad tanto de periféricos diferentes, como de componentes de software. En este sentido, es común encontrarse con la situación de que varias tareas accedan al mismo componente de software o utilicen el mismo dispositivo de hardware. Lo cual vuelve este hecho extremadamente sensible a lo largo de todo el proceso de desarrollo del sistema.

Las problemáticas que surgen al abordar el tema de recursos compartidos en sistemas de tiempo real se pueden dividir en dos grandes grupos: las que son comunes a sistemas de propósito general y aquellas de particular atención en estos sistemas. Dentro del primer grupo, se destacan: el *deadlock* y la inanición [95]. Mientras que en el segundo, sobresalen la inversión de prioridades y el tiempo de bloqueo no acotado. A continuación, se describe cada una de las problemáticas.

Deadlock: Un conjunto de tareas está en *deadlock* si cada tarea en el conjunto está esperando un evento que sólo otra tarea en el conjunto puede provocar.

Inanición: Una tarea sufre inanición si nunca puede acceder a un recurso que necesita para completar su ejecución.

Inversión de prioridad: Esta situación se produce cuando, habiendo dos o más tareas listas para ejecutar, se ejecuta la de menor prioridad, debido a que posee un recurso que necesita la tarea de mayor prioridad[1].

[1]Un ejemplo clásico en sistemas de tiempo real es el problema experimentado por la *Mars Pathfinder* en 1997, el cual involucró una inversión de prioridad (ver [79]).

Tiempo de bloqueo no acotado: Esta situación está muy relacionada con la anterior, en el sentido que, cuando se produce una inversión de prioridad, generalmente no se sabe cuánto tiempo durará dicha inversión.

Las situaciones planteadas anteriormente pueden llevar a un mal funcionamiento del sistema, en general, y a la pérdida de vencimientos de algunas tareas, en particular. Para evitar este tipo de problemas, es necesario contar con técnicas y mecanismos confiables que provean predecibilidad en todas las etapas del proceso de desarrollo. Desde las primeras fases del modelado hasta la ejecución misma de la aplicación se requiere tener en cuenta la problemática que surge al compartir recursos en un sistema de tiempo real. Por esto, es de fundamental importancia que una política de manejo de recursos compartidos sea lo suficientemente versátil para ser aplicada en los diferentes niveles de abstracción que involucra el desarrollo de una aplicación.

3.1.1. Contribución

En este capítulo, se presenta una política de manejo de recursos compartidos para sistemas de tiempo real. La política propuesta se denomina Política de Recursos Pila Extendida (ESRP, por la sigla en inglés) y evita todas las situaciones problemáticas enumeradas anteriormente. A la vez, ESRP provee una nueva visión sobre una política clásica y bien conocida. Este nuevo punto de vista beneficia al desarrollador en el sentido que facilita el tratamiento de recursos compartidos, al permitirle contemplar dicho tratamiento a lo largo de todo el proceso de desarrollo. Este hecho hace a la política ESRP una herramienta útil en el desarrollo de sistemas de tiempo real.

La idea general de la política ESRP es dividir el conjunto de tareas del sistema en subconjuntos, de acuerdo a los posibles conflictos de recursos que puedan tener. Asimismo, dentro de cada subconjunto y entre subconjuntos, se establecen condiciones para la ejecución de las tareas. Por otro lado, ESRP está libre de *deadlocks*, no permite inanición, sólo admite una inversión de prioridad y establece una cota de tiempo máximo de bloqueo para cada

tarea. Otro aspecto que lo vuelve atractivo es su flexibilidad para poder adaptarse a sistemas que utilicen prioridades fijas (*e.g.*, *Rate Monotonic* [65]), prioridades dinámicas (*e.g.*, *Earliest Deadline First* [65]) o basados en servidores (*e.g.*, Política de Planificación Comportamental).

3.2. Conceptos Previos

El tratamiento de recursos compartidos es un área de investigación muy prolífica en sistemas de tiempo real. A lo largo de muchos años, se han planteado soluciones y políticas para atacar esta problemática desde diferentes ángulos y bajo distintas suposiciones. Así, se tienen políticas para manejo de recursos compartidos en prioridades fijas [87]; en prioridades dinámicas [12]; en sistemas abiertos [87, 62, 83], donde la creación de tareas se puede hacer en forma dinámica; en sistemas basados en servidores [28, 27]; entre otros. Sin embargo, en todos estos casos y otros que completan una larga lista, las bases sobre las que se construyen las soluciones son los dos primeros trabajos nombrados anteriormente. Esto es, el de Sha *et ál* de 1990 [87] y el de Baker de 1991 [12]. Este último trabajo, de hecho, será tomado como referencia principal en el desarrollo de la política de manejo de recursos compartidos presentada en este capítulo.

Como se explicó en la Sección 2.2, una tarea de tiempo real (dura o blanda) τ_i es una entidad de ejecución representada por una tupla (C_i, D_i, T_i); mientras que un sistema de tiempo real $\mathbf{\Gamma}$, está conformado por n tareas, siendo $\mathbf{\Gamma} = \{\tau_1, \tau_2, \ldots, \tau_n\}$. En aquella oportunidad, se asumió que las tareas no compartían otro recurso que el procesador, es decir eran independientes. En este capítulo, se asumirá que tanto las tareas duras como la blandas pueden compartir un conjunto $\mathbf{R} = \{R_i \mid i = 1, 2, ..., m\}$ de m recursos no-apropiables, accesibles serialmente, los cuales pueden ser físicos o lógicos, declarados en tiempo de compilación. El hecho de que los recursos sean no-apropiativos y accesibles serialmente, significa que una vez que una tarea comienza a utilizar un recurso sólo libera el control del mismo cuando no lo utiliza más (es decir, ninguna otra tarea se lo puede quitar). Esto no implica que lo utilice

un intervalo de tiempo continuo, sino que la tarea puede ser apropiada, pero la apropiación no determina que libere el recurso. De lo anterior, surge que cuando una tarea realiza una operación sobre un recurso compartido, el acceso debe hacerse de modo mutuamente exclusivo a fin de mantener consistencia. La porción de código ejecutable, durante la cual la tarea accede al recurso, se denomina *sección crítica* y se simboliza $\xi_k(\tau_i)$ para cada sección crítica k de la tarea τ_i. El acceso a una sección crítica, en general, está protegido por algún tipo de mecanismo que garantiza la exclusión mutua. En este punto, se asumirá que las secciones críticas están protegidas por semáforos de tipo *mutex*, mediante las operaciones clásicas `lock()` y `unlock()`. Asimismo, los accesos sucesivos a secciones críticas pueden anidarse, siempre que se hagan correctamente. Esto es, dado un par cualquiera de secciones críticas ξ_p y ξ_q, o bien $\xi_p \subset \xi_q$, o bien $\xi_q \subset \xi_p$, o bien $\xi_p \cap \xi_q = \emptyset$, donde las operaciones entre secciones críticas se entienden como las clásicas de conjuntos.

```
task  Tarea1{
        ...
        lock(R1);
        ...
        lock(Rx);
        ...
        unlock(Rx);
        ...
        unlock(R1);
        ...}
```

(a) Sección crítica anidada únicamente.

```
task  Tarea2{
        ...
        lock(R1);
        ...
        unlock(R1);
        ...
        lock(R2);
        ...
        lock(Rx);
        ...
        unlock(Rx);
        ...
        unlock(R2);
        ...
        lock(Ry);
        ...
        unlock(Ry);
        ...}
```

(b) Mezcla de secciones críticas anidadas y no anidadas.

Figura 3.1: Diferentes definiciones de $\xi_{\text{máx}}(\tau_i)$.

En lo concerniente al tiempo de duración de una sección crítica, éste puede variar entre

tareas, incluso accediendo al mismo recurso. Un caso especial, que será tomado como referencia en las secciones subsiguientes, es el de la sección crítica más externa, no trivial, la cual es aquella no anidada en ninguna otra. Esto es válido para una tarea que presenta un esquema de acceso a recursos como el de la Figura 3.1(a) (*i.e.*, la sección crítica más externa, no trivial, comienza con `lock(R1)` y finaliza con `unlock(R1)`). Como se ve claramente, esta sección crítica es, además de la única, la de máxima duración de la tarea. Para el caso de una tarea que incluye secciones críticas anidadas y no anidadas, como el de la Figura 3.1(b), la sección crítica de máxima duración se debe buscar entre todas las que contiene la tarea. Esto lleva a la siguiente definición de sección crítica, que es válida para ambos casos presentados en la Figura 3.1.

Definición 3.2.1 (Sección crítica máxima)**.** Se define a la sección crítica máxima $\xi_{\text{máx}}(\tau_i)$ de una tarea τ_i, como aquella de máxima duración entre todas las que incluye la tarea. Esto es, $\xi_{\text{máx}}(\tau_i) = \text{máx}\{\xi_k(\tau_i)\}$ con k variando entre 1 y el número máximo de secciones críticas de τ_i.

3.3. Política de Recursos Pila Extendida

En esta sección, se presenta la Política de Recursos Pila Extendida (ESRP). Esta política se basa en el trabajo de Baker [12], donde se presenta la *Stack Resource Policy* (SRP). La política ESRP evita los principales problemas que surgen al compartir recursos en sistemas de tiempo real, al tiempo que beneficia al desarrollador proveyéndole predecibilidad desde las etapas más tempranas del modelado.

Definición 3.3.1 (Conjunto de recursos de una tarea)**.** El conjunto de recursos usado por una tarea se define como $\mathbf{R}_{\tau_i} = \{\bigcup R_j \mid \tau_i \texttt{ usa } R_j\}$

Definición 3.3.2 (Bloqueo)**.** Una tarea τ_i está bloqueada en un recurso R si existe una tarea de menor prioridad τ_j que tiene posesión sobre R evitando que τ_i lo pueda usar y en consecuencia, pueda continuar su ejecución.

Definición 3.3.3 (Relación de bloqueo)**.** Dos tareas tienen una relación de bloqueo $\between$ si comparten directa o indirectamente un recurso. Ésta es una relación de equivalencia que tiene las siguientes propiedades:

1. $\tau_i \between \tau_i$
2. Si $\tau_i \between \tau_j$ luego, $\tau_j \between \tau_i$.
3. Si $\tau_i \between \tau_j$ y $\tau_j \between \tau_k$ luego, $\tau_i \between \tau_k$, incluso si τ_i y τ_k no tienen recursos en común.

En este punto, es importante distinguir diferentes tipos de bloqueos de acuerdo a cómo y cuándo se dan. Conceptualmente, el caso 2. de la definición previa es un *bloqueo directo*, ya que ambas tareas acceden al mismo recurso. En el caso 3., las tareas τ_i y τ_k forman parte de una cadena, pero no comparten ningún recurso directamente. Por lo tanto, éste es un *bloqueo indirecto*. Por otro lado, los bloqueos también se pueden clasificar de acuerdo al momento en que ocurren. Así, se tiene el *bloqueo tardío*, que se da en el momento en que la tarea intenta tomar el recurso (*i.e.*, cuando realiza una operación `lock()` sobre el semáforo que protege al mismo); y el *bloqueo temprano*, el cual se da en el instante en que una tarea intenta apropiar a otra que durante la instancia de ejecución actual la bloqueará. Esto es, el SOTR controla, antes de enviar a ejecución a la tarea, si dicha tarea va a sufrir un bloqueo por parte de alguna de las tareas que actualmente compiten por el procesador. Si ese bloqueo es posible, entonces el SOTR bloquea a la tarea antes de que comience a ejecutar. Este mecanismo es útil si se pretenden evitar cambios de contexto innecesarios. Una discusión más cercana a la implementación práctica de este tipo de bloqueo se presenta en la Sección 3.4.

La Definición 3.3.3 presenta un resultado muy fuerte, ya que introduce la posibilidad de particionar el conjunto de tareas $\mathbf{\Gamma}$ en subconjuntos.

Definición 3.3.4 (Conjunto de tareas bloqueantes)**.** Es una clase de equivalencia obtenida por la relación de bloqueo $\between$. Se simboliza como $\mathbf{\Upsilon} = [\tau_i] = \{\tau_j \in \mathbf{\Gamma} \mid \tau_i \between \tau_j\}$.

En consecuencia, dado $\mathbf{\Gamma}/\not\wr = \{\mathbf{\Upsilon_1}, \mathbf{\Upsilon_2}, \ldots, \mathbf{\Upsilon_n}\} \mid \mathbf{\Upsilon_v} \cap \mathbf{\Upsilon_w} = \emptyset,\ \forall \mathbf{\Upsilon} \in \mathbf{\Gamma}/\not\wr$, luego $\tau_i \sim\not\wr\ \tau_j, \forall \tau_i\tau_j \mid \tau_i \in \mathbf{\Upsilon_v} \ \wedge\ \tau_j \in \mathbf{\Upsilon_w}$. Es decir, dado cualquier par de tareas pertenecientes a dos conjuntos de tareas bloqueantes diferentes, entre ellas no existe una relación de bloqueo. Esto no implica que no puedan bloquearse entre sí, sino que indica que comparten un recurso directa o indirectamente.

Definición 3.3.5 (Conjunto de recursos bloqueantes)**.** Se define como el conjunto de recursos usados por las tareas en un conjunto de tareas bloqueantes. Se denota como $\mathbf{\Phi_w} = \{\bigcup \mathbf{R}_{\tau_i} \mid \tau_i \in \mathbf{\Upsilon_w}\}$.

Por las definiciones 3.3.1 y 3.3.3, si $\tau_i \not\wr\ \tau_j$: $\mathbf{R}_{\tau_i} \cap \mathbf{R}_{\tau_j} \neq \emptyset$ o $\exists\{\tau_{k_1}, \tau_{k_2}, \ldots \tau_{k_m}\} | (\mathbf{R}_{\tau_i} \cap \mathbf{R}_{\tau_1}) \wedge (\mathbf{R}_{\tau_1} \cap \mathbf{R}_{\tau_2}) \wedge \ldots \wedge (\mathbf{R}_{\tau_{m-1}} \cap \mathbf{R}_{\tau_m}) \wedge (\mathbf{R}_{\tau_m} \cap \mathbf{R}_{\tau_j}) =$ `VERDADERO`. Luego si $[\tau_i] = [\tau_j] = \mathbf{\Upsilon_w}; \mathbf{R}_{\tau_i} \bigcup \mathbf{R}_{\tau_j} \subset \mathbf{\Phi_w}$, donde $\mathbf{\Phi_w}$ es el invariante de clase bajo $\not\wr$.

Definición 3.3.6 (Conjunto de tareas activo)**.** Un conjunto de tareas bloqueantes $\mathbf{\Upsilon}$ está *activo* cuando al menos una tarea en $\mathbf{\Upsilon}$ está lista para ser ejecutada.

Cada tarea τ_i, en el sistema tiene asociado un nivel de apropiación y una prioridad, denotados $\pi(\tau_i)$ y $p(\tau_i)$, respectivamente. La prioridad se establece de acuerdo a la política de planificación seleccionada (*e.g.*, *Earliest Deadline First*, *Rate Monotonic*, Planificación Comportamental) y puede ser fija o dinámica; mientras que el nivel de apropiación es fijo para cada instancia de ejecución de la tarea. Así, el nivel de apropiación es fijo mientras no se modifiquen ni el vencimiento relativo ni el período de la tarea. En general, los niveles de apropiación están relacionados a una propiedad específica de las tareas. En la Sección 3.4, se expondrán las consideraciones a tener en cuenta para aplicar ESRP con un esquema de prioridades fijas, uno de prioridades dinámicas y sobre la Política de Planificación Comportamental. A continuación se adelanta una definición genérica del nivel de apropiación de una tarea τ_i:

$$\forall i \quad \pi(\tau_i) \leq \tfrac{1}{D_i} \leq \tfrac{1}{T_i} = \pi^{\text{máx}}(\tau_i) \tag{3.3.1}$$

A partir de las definiciones del modelo de sistema encontradas en [12], una tarea toma un recurso R mediante la ejecución de un pedido. Si el recurso no está disponible inmediatamente, debe esperar a que la asignación se haga factible. Mientras la tarea espera que se libere R, se dice que está *bloqueada* y, una vez asignado el recurso, mientras lo tiene en posesión se nombra a esa posesión como *destacada*. Luego de que la asignación destacada terminó, la tarea debe ejecutar una instrucción que libere el recurso R.

Definición 3.3.7 (Techo del Recurso[2])**.** Cada recurso R tiene asociado un techo $\lceil R \rceil$ que es una función entera de las asignaciones destacadas de R.

Definición 3.3.8 (Techo del Conjunto)**.** Cada conjunto de tareas bloqueantes $\boldsymbol{\Upsilon}_{\mathbf{v}}$ tiene asociado un techo del conjunto, dado por $\overline{\pi}_{\boldsymbol{\Phi}_{\mathbf{v}}}$, el cual es el máximo $\lceil R_i \rceil$ de todos lo recursos en $\boldsymbol{\Phi}_{\mathbf{v}}$. Formalmente, $\overline{\pi}_{\boldsymbol{\Phi}_{\mathbf{v}}} = \text{máx}\{\lceil R_i \rceil | R_i \in \boldsymbol{\Phi}_{\mathbf{v}}\}$

Como se mencionó al comienzo de este capítulo, los *deadlocks* y las múltiples inversiones de prioridad son dos de los problemas principales a considerar al compartir recursos. Para prevenirlos es necesario imponer algunas condiciones al comienzo de la ejecución de las tareas.

Condición 3.3.1 (Deadlocks)**.** A una tarea no debería permitírsele comenzar a ejecutar hasta que todos los recursos necesarios para su ejecución estén disponibles.

Condición 3.3.2 (Múltiples inversiones de prioridad)**.** A una tarea no debería permitírsele comenzar a ejecutar hasta que todos los recursos necesarios para su ejecución estén disponibles y puedan satisfacer los requerimientos de cada instancia de ejecución que pueda apropiarla.

La siguiente condición establece la relación entre techos de recursos y niveles de apropiación de las tareas.

Condición 3.3.3. Si la tarea τ_i está actualmente ejecutando o puede apropiar a la tarea que está actualmente ejecutando, y puede ser bloqueada al pedir R_j por otra tarea con mayor nivel de apropiación, luego $\pi(\tau_i) \leq \lceil R_j \rceil$.

[2]La denominación *techo* no debe entenderse como la homónima y bien-conocida función matemática. Esta designación proviene de la palabra inglesa *ceiling* que intenta expresar una idea de tope o límite.

Lema 3.3.1. *La Condición 3.3.1 garantiza que una tarea no puede ser bloqueada luego que inicia su ejecución.*

Demostración. Supóngase que el Lema 3.3.1 es falso. En ese caso, dada la Condición 3.3.1 una tarea τ_i puede ser bloqueada luego que inicia su ejecución. El modelo de recursos propuesto (*i.e.*, recursos simples protegidos por semáforos mutex) hace que, por la Condición 3.3.1, una vez que a τ_i se le permite comenzar su ejecución todos sus recursos necesarios están disponibles. Suponiendo que τ_i es apropiada por τ', (es decir, $p(\tau_i) < p(\tau')$), luego se tienen los siguientes casos:

1. Si τ_i no tomó ningún recurso antes de la apropiación.

   a) Se supone $[\tau'] = [\tau_i]$. Luego, $p(\tau_i) < p(\tau')$ y si toma cualquier recurso no se considera como un bloqueo. En consecuencia, τ_i no puede apropiar a τ' durante la instancia de ejecución actual. Por lo tanto, no hay un bloqueo.

   b) Se supone $[\tau'] \neq [\tau_i]$. Como τ' y τ_i no comparten recursos, no existe ningún bloqueo.

2. Si τ_i tomó un recurso antes de la apropiación.

   a) Se supone $[\tau'] = [\tau_i]$. Por la Condición 3.3.1, τ' no podría haber comenzado su ejecución si sus recursos necesarios no estaban disponibles. Para probar que τ_i no puede ser bloqueada indirectamente, se consideran tres tareas $\tau' \lozenge \tau'' \lozenge \tau_i$ y $p(\tau'') > p(\tau') > p(\tau_i)$. En este caso, τ_i fue apropiada por τ' e incluso perteneciendo al mismo $\boldsymbol{\Upsilon}$, no comparten recursos. Luego, arriba τ'' que tiene mayor prioridad que las otras dos tareas y comparte recursos con ambas. Sin embargo, τ'' no puede apropiar a τ', ya que por la Condición 3.3.1 no puede haber comenzado su ejecución sin tener todos los recursos necesarios disponibles.

   b) Se supone $[\tau'] \neq [\tau_i]$. Este caso es análogo a 1.b).

□

Teorema 3.3.2. *La Condición 3.3.1 es suficiente para prevenir deadlocks.*

Demostración. Una manera de prevenir la ocurrencia de deadlocks es evitando una espera circular [75]. Por el Lema 3.3.1, una tarea no puede ser bloqueada luego que inicia su ejecución. Con esto, no puede haber dos o más tareas en ejecución esperando por recursos que posee otra tarea. En consecuencia, no se puede dar una espera circular. □

Teorema 3.3.3. *Si se impone la Condición 3.3.1, entonces la Condición 3.3.2 es suficiente para prevenir múltiples inversiones de prioridad.*

Demostración. Supóngase que hay una múltiple inversión de prioridad. Una tarea τ' puede estar sujeta a tal inversión de prioridad si dos o más tareas de menor prioridad τ y τ'' la están bloqueando. Por la Condición 3.3.1, τ y τ'' tienen que haber arribado antes que τ' comience su ejecución y se tienen que haber apropiado mutuamente. Sin pérdida de generalidad, se puede asumir que τ apropió a τ''. Luego, la Condición 3.3.2 fue violada al permitir que τ comience su ejecución.

Es importante destacar, que todas las tareas (τ, τ' y τ''), deben pertenecer al mismo $\boldsymbol{\Upsilon}$ a fin de tener un relación de bloqueo. □

En lo que sigue, se presenta la Política de Recursos Pila Extendida (ESRP), basada en los conceptos de Baker y los planteados anteriormente acerca de conjuntos de tareas bloqueantes y conjuntos de recursos.

Lema 3.3.4. *Supóngase que τ no está ejecutando, R es un recurso y $p(\tau) = p_{\text{máx}}$, donde $p_{\text{máx}}$ es la máxima prioridad del sistema:*

1. *Si $\lceil R \rceil < \pi(\tau)$, luego τ puede ejecutar sin tener que esperar que se libere R;*
2. *Si $\lceil R \rceil \leq \pi(\tau)$, luego toda tarea que pueda apropiar a τ puede tomar el recurso.*

Demostración. 1. Se supone $\lceil R \rceil < \pi(\tau)$, pero τ tiene que esperar que se libere R para poder ejecutar. Luego, por la Condición 3.3.3, $\pi(\tau) \leq \lceil R_i \rceil$, una contradicción.

2. Se supone $\lceil R \rceil \leq \pi(\tau)$, pero para algún τ_H que puede apropiar a τ, τ_H no puede tomar R. Por la Condición 3.3.3, $\pi(\tau) \leq \lceil R_i \rceil$, pero para que τ_H pueda apropiar a τ (y por el Lema 3.3.1 y la parte 1. del Lema 3.3.4) se tiene que dar $\pi(\tau) \leq \lceil R \rceil < \pi(\tau_H)$. Esto es equivalente a $\pi(\tau) < \pi(\tau_H)$ y contradice que si $\lceil R \rceil \leq \pi(\tau)$, τ_H no puede tomar R.

□

Teorema 3.3.5. *Sea un único $\boldsymbol{\Upsilon}$ y $R_i \in \boldsymbol{\Phi}$; si a ninguna tarea $\tau \in \boldsymbol{\Upsilon}$ se le permite comenzar su ejecución hasta que $\lceil R_i \rceil < \pi(\tau)$ para todo recurso $R_i \in \boldsymbol{\Phi}$ luego:*

1. *Ninguna tarea puede ser bloqueada luego de que comienza su ejecución por otra tarea perteneciente a $\boldsymbol{\Upsilon}$;*

2. *No puede haber deadlocks;*

3. *Ninguna tarea puede ser bloqueada por más tiempo que la duración de una sección crítica máxima $\xi_{\text{máx}}(\tau_j)$ con τ_j de menor prioridad y perteneciente a $\boldsymbol{\Upsilon}$ (i.e., la única manera que puede ocurrir esto es mediante un bloqueo temprano antes de que la tarea comience su ejecución).*

Demostración. La parte 1. sigue de la parte 1. del Lema 3.3.4 y del Lema 3.3.1. La parte 2. sigue de la parte 2. del Lema 3.3.4 y del Teorema 3.3.2. Finalmente, la parte 3. sigue de la parte 2. del Lema 3.3.4 y del Teorema 3.3.3. □

En el siguiente teorema, la política original SRP se extiende a múltiples conjuntos de tareas bloqueantes $\boldsymbol{\Upsilon}$ en el sistema.

Teorema 3.3.6. *Si a ninguna tarea $\tau_i \in \boldsymbol{\Upsilon}_{\mathbf{v}}$ se le permite comenzar su ejecución hasta que $\overline{\pi}_{\boldsymbol{\Phi}_{\mathbf{u}}} < \pi(\tau_i)$ para cada conjunto de tareas bloqueantes activo $\boldsymbol{\Upsilon}_{\mathbf{u}}$:*

1. *Ninguna tarea puede ser bloqueada luego de que comienza su ejecución por ninguna otra tarea;*

2. *No puede haber deadlocks;*

3. *Ninguna tarea puede ser bloqueada por más tiempo que la duración de una sección crítica máxima $\xi_{\text{máx}}(\tau_j)$ con τ_j de menor prioridad (i.e., la única manera que puede ocurrir esto es mediante un bloqueo temprano antes de que la tarea comience su ejecución).*

Demostración. Para probar este teorema se considera la definición de Techo del Conjunto (Definición 3.3.8). En particular, la parte 1. sigue de la parte 1. del Teorema 3.3.5, la parte 2. sigue de la parte 2. del Teorema 3.3.5 y la parte 3. sigue de la parte 3. del Teorema 3.3.5. □

Corolario 3.3.6.1. *Una tarea $\tau_i \in \boldsymbol{\Upsilon}_{\mathbf{v}}$ puede ser apropiada por una tarea de mayor prioridad $\tau_j \in \boldsymbol{\Upsilon}_{\mathbf{u}}$ si $\overline{\pi}_{\boldsymbol{\Phi}_{\mathbf{v}}} < \pi(\tau_j)$.*

Demostración. Por el Teorema 3.3.6, τ_j tiene que haber arribado después de que τ_i haya comenzado su ejecución. Así, τ_j no fue tomada en cuenta cuando se consideraba el inicio de τ_i. Como $\tau_j \notin \boldsymbol{\Upsilon}_{\mathbf{v}}$ y $\overline{\pi}_{\boldsymbol{\Phi}_{\mathbf{v}}}$ es el máximo techo actual de todos los conjuntos bloqueantes activos, τ_j, teniendo mayor prioridad y nivel de apropiación, puede ejecutar. En consecuencia, puede apropiar a τ_i. □

En base al Teorema 3.3.6 la política SRP se extendió para manejar este nuevo concepto de conjuntos de tareas bloqueantes. En este sentido, una tarea se bloquea antes de comenzar su ejecución (*i.e.*, bloqueo temprano) hasta ser la de mayor prioridad y nivel de apropiación en el sistema, que solicita ejecución. Asimismo, para que τ_i pueda apropiar a la tarea que se está ejecutando actualmente, debe valer:

$$\lceil R_j \rceil < \pi(\tau_i), \quad \forall R_j \in \boldsymbol{\Phi}_{\mathbf{v}} | \boldsymbol{\Upsilon}_{\mathbf{v}} \textit{ está activo} \tag{3.3.2}$$

De esta manera, una tarea que comienza su ejecución no podrá ser bloqueada por ninguna otra tarea que se ejecute concurrentemente con ella. Esto es así debido a que para poder ser bloqueada, la tarea que haría el bloqueo debería tener mayor prioridad y mayor nivel de apropiación que el techo de su propio conjunto de tareas bloqueantes y que todos los techos de todos los conjuntos activos. Con lo cual, no sería, en realidad, un bloqueo sino

una apropiación normal en un sistema basado en prioridades. A partir de esto, se tiene la siguiente definición:

Definición 3.3.9 (Techo del sistema)**.** Se define como el máximo techo de todos los conjuntos activos. Formalmente, $\overline{\pi} = \text{máx}\{\overline{\pi}_{\mathbf{\Phi_v}} | \mathbf{\Upsilon_v}$ *está activo*$\}$

Con todo, se tiene que para que una tarea τ_i pueda apropiar a una τ_j el siguiente *Test de Apropiación* debe ser verdadero.

$$p(\tau_j) < p(\tau_i) \wedge \pi(\tau_j) = \overline{\pi} < \pi(\tau_i) \tag{3.3.3}$$

Del test anterior, se desprende que, cuando una tarea comienza su ejecución, *hereda* el máximo nivel de apropiación posible en su conjunto de tareas bloqueantes, esto es el techo de su conjunto. Sin embargo, a fin de favorecer y alentar el carácter apropiativo de la política, este hecho se efectiviza realmente cuando la tarea intenta tomar un recurso. En la sección siguiente esta situación se profundizará al considerar un SOTR genérico que implementa la política ESRP. Asimismo, es de destacar, que una tarea puede ser eventualmente apropiada por otra con mayor prioridad y nivel de apropiación, ya que el techo del sistema $\overline{\pi}$ se computa a partir de los conjuntos de tareas activos. Con la nueva formulación, las inversiones de prioridad siguen acotadas (como en la política original) a una.

Finalmente, en la introducción de este capítulo se mencionó un problema al compartir recursos que aún no se ha tratado: la inanición. Este problema se puede definir como una espera infinita por la liberación de un recurso. La diferencia principal entre esta situación y aquella que se da cuando el tiempo de espera no está acotado es que en el segundo caso la tarea eventualmente obtendrá los recursos que necesita, mientras que en el primero eso no se dará nunca. Una técnica para evitar este problema, comúnmente utilizada en sistemas operativos, es el *envejecimiento* [95]. Este método establece que las tareas listas para ejecutar ganan paulatinamente prioridad a medida que transcurre el tiempo y continúan en ese estado de listas, pero no ejecutan. Se puede ver que este concepto es análogo a lo que ocurre al utilizar

las políticas EDF y ESRP combinadas, ya que en EDF una tarea lista para ejecutar, gana prioridad a medida que transcurre el tiempo, es decir, que se acerca a su vencimiento. Lo cual hará que eventualmente esa tarea sea la de mayor prioridad en el sistema. Para el caso de que se utilice una política de prioridades fijas junto con ESRP, la situación es similar. Esto es, si el conjunto de tareas supera el test de factibilidad (ver la sección siguiente) todas las tareas ejecutarán antes de su vencimiento, lo cual implica que en algún momento serán las de máxima prioridad del sistema. Con esto, se puede ver que la política de manejo de recursos compartidos ESRP soluciona todos aquellos problemas presentes al compartir recursos.

3.4. Consideraciones de Diseño

3.4.1. Implicancias sobre el Modelado

La política ESRP provee una interesante visión sobre las relaciones entre tareas que comparten recursos, mediante la definición de conjuntos de tareas y de recursos. La determinación temprana de estos conjuntos beneficia al desarrollador, ya que le permite tener en cuenta y hacer consideraciones sobre el tratamiento de los recursos compartidos desde el comienzo del proceso de desarrollo.

En la etapa de modelado y diseño del sistema, donde se comienzan a definir cuestiones que luego serán implementadas, resulta necesario no sólo tener en consideración las características temporales, sino también las relaciones existentes entre tareas. Esto facilita la detección y prevención de posibles conflictos que lleven al sistema a un mal funcionamiento. En este sentido, el descubrimiento temprano de dichas situaciones proporciona mayores alternativas para solucionarlas y menores perjuicios que en etapas avanzadas del desarrollo.

3.4.2. Condiciones de Factibilidad

En esta sección, se presentan los test de factibilidad correspondientes a la aplicación de ESRP en conjunto con *Rate Monotonic* (RM) [65], *Earliest Deadline First* (EDF) [65] y la

Política de Planificación Comportamental (BIPS), respectivamente. Los resultados aquí expuestos enfatizan la amplia variedad de sistemas que se pueden tratar con ESRP, su adaptabilidad a diversos esquemas de asignación de prioridades y su simplicidad de aplicación.

Teorema 3.4.1 (Factibilidad con RM)**.** *Un conjunto de n tareas es factible por la política de planificación RM con el protocolo de manejo de recursos compartidos ESRP si:*

$$\forall i \text{ con } 1 \leq i \leq n, T_i \geq \min t | t = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{t}{T_k} \right\rceil C_k$$

donde $B_i = \max\{\xi_{\max}(\tau_j) | p(\tau_j) < p(\tau_i)\}$*, es decir,* B_i *representa la duración de la sección crítica máxima de aquellas tareas que pueden bloquear a* τ_i*.*

Demostración. La prueba es análoga a la presentada en [87], ya que en el peor caso todas las tareas pertenecen al mismo Υ y la política SRP es idéntica a la presentada en el mencionado trabajo para el caso de niveles de apropiación fijos e iguales a las prioridades de las tareas (ver [12]). □

Teorema 3.4.2 (Factibilidad con EDF)**.** *Un conjunto de n tareas es factible por la política de planificación EDF con el protocolo de manejo de recursos compartidos ESRP si:*

$$\forall k \mid k = 1, \ldots, n \quad \left(\sum_{i=1}^{n} \frac{C_i}{T_i} \right) + \frac{B_k}{T_k} \leq 1$$

donde $B_k = \max\{\xi_{\max}(\tau_j) | D_k < D_j\}$*, es decir,* B_k *representa la duración de la sección crítica máxima de aquellas tareas que pueden bloquear a* τ_k*.*

Demostración. La prueba es análoga a la presentada por Baker en [12]. □

Finalmente, a continuación se establecen las condiciones de factibilidad para un esquema jerárquico EDF-BIPS con el protocolo de manejo de recursos compartidos ESRP.

Teorema 3.4.3 (Factibilidad con BIPS)**.** *Un conjunto de n servidores BIPS, que encapsulan n tareas, es factible por la política de planificación EDF con el protocolo de manejo de recursos*

compartidos ESRP si:

$$\forall k \mid k = 1, \ldots, n \left(\sum_{i=1}^{k} \frac{Q_i}{P_i} \right) + \frac{B_k}{P_k} \leq 1$$

donde $B_k = \text{máx}\{\xi_{\text{máx}}(\tau_j) | D_k < D_j\}$*, es decir,* B_k *representa la duración de la sección crítica máxima de aquellas tareas encapsuladas que pueden bloquear a* τ_k*.*

Demostración. La demostración es análoga a la del Teorema 3.4.2 considerando el Teorema 2.4.2 del Capítulo 2. □

3.4.3. Aspectos de Implementación

Como se anticipo anteriormente, en esta parte final del capítulo se expondrán algunas cuestiones relacionadas a la implementación de la Política de Recursos Pila Extendida (ESRP) en un Sistema Operativo de Tiempo Real (SOTR) genérico. Se abarcarán aspectos relacionados a los estados que una entidad de ejecución (tarea o servidor) atraviesa bajo ESRP; las adaptaciones necesarias, en especial referidas al cálculo del nivel de apropiación π, para diferentes políticas de planificación; y por último una serie de consideraciones prácticas al momento de implementar un sistema basado en ESRP.

3.4.3.1. Estados de una Entidad de Ejecución bajo ESRP

La principal función de un SOTR, en un ambiente multiprogramado de tiempo compartido, es controlar la ejecución de tareas. Esto incluye determinar cuál tarea se va ejecutar en un determinado instante de tiempo, realizar las asignaciones de prioridad impuestas por la política de planificación elegida, administrar los recursos compartidos entre las diferentes tareas y manejar las estructuras de datos internas que garanticen la correcta ejecución de esas tareas. Estas características mencionadas se pueden ver más claramente en un Modelo de Estados de la Tarea, el cual muestra los diferentes estados que atraviesa una tarea a lo largo de su ciclo de vida. Las transiciones entre los diferentes estados son hechas por el SOTR de acuerdo a la política de planificación de tareas y a la de manejo de recursos compartidos

que se utilicen.

El caso particular de un Modelo de Estados para la política ESRP se muestra en la Figura 3.2. A continuación se describen cada uno de ellos y luego la dinámica del modelo.

OCIOSA: La tarea no es considerada al momento de planificar. En general, para una tarea periódica, esto se debe a que ya concluyó una instancia de ejecución y ahora espera su reactivación.

LISTA: La tarea pertenece al conjunto de aquellas que actualmente compiten por el procesador, pero su prioridad no es la máxima del sistema en ese momento.

BLOQUEADA: Los recursos necesarios para ejecutar la tarea no están disponibles. Esto es, su nivel de apropiación no es el mayor del sistema en ese momento.

EJECUTANDO: La tarea tiene el mayor nivel de apropiación y la mayor prioridad del sistema actualmente.

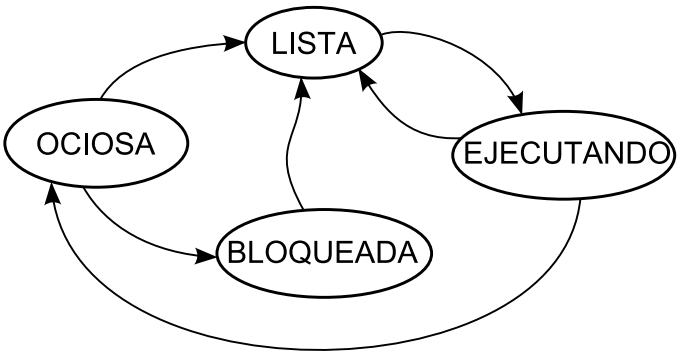


Figura 3.2: Modelo de Estados de una Tarea bajo ESRP

En la Figura 3.2 se puede ver que una tarea que está OCIOSA tiene dos estados posibles a seguir: puede pasar a LISTA, si su nivel de apropiación es mayor que el máximo del sistema actualmente (esto es, $\overline{\pi} < \pi(\tau)$); o bien será BLOQUEADA, en caso contrario. Esta última alternativa muestra cómo se implementa el *bloqueo temprano* que impone la política ESRP. Una vez liberados los recursos que necesita la tarea, ésta pasa al estado LISTA. Allí compite, con el resto de las tareas que están en condiciones de ejecutar, por el procesador.

La elección de cuál tarea debe ejecutar se hace en base a la política de planificación elegida, pero, en todos los casos, la tarea que estará EJECUTANDO será la de mayor prioridad y mayor nivel de apropiación (excepto en el caso de inversión de prioridad, como se explicó en secciones anteriores). Esa tarea que se está ejecutando volverá al estado LISTA, si entra en la competencia por el procesador una nueva tarea de mayor nivel de apropiación y mayor prioridad que la apropie. Por otro lado, una vez terminada la ejecución de una tarea, ésta vuelve al estado OCIOSA a esperar por su próxima reactivación.

Finalmente, cabe señalar que lo expuesto anteriormente sobre tareas es válido también para servidores (en particular, BIPS), independientemente de la planificación interna del servidor. El esquema planteado es genérico y puede ser fácilmente adaptado a las necesidades particulares de cada sistema

3.4.3.2. Cálculo del Nivel de Apropiación

El aspecto fundamental introducido por Baker [12] en la política SRP es el nivel de apropiación. Éste indica la precedencia de una tarea sobre las otras al momento de acceder a los recursos que necesite. En otras palabras, mientras la prioridad indica cuán importante es esa tarea dentro del sistema; el nivel de apropiación indica si puede o no tomar los recursos necesarios para su ejecución. Es decir, el nivel de apropiación establece, en base a los recursos requeridos por una tarea, si ésta puede apropiar a la tarea que se está ejecutando actualmente. Este concepto sigue siendo fundamental en la política ESRP presentada aquí. Por esto, resulta necesario hacer una distinción acerca del cálculo del nivel de apropiación para diferentes tipos de políticas de planificación que se pueden ejecutar junto con ESRP. Se debe notar que las políticas de planificación sobre las que se hace el análisis representan los casos más destacables de políticas para prioridades fijas, prioridades dinámicas y servidores abordados en esta tesis.

Independientemente de la política utilizada para planificar a las entidades de ejecución, el nivel de apropiación, de cada una de ellas, es, en general, fijo a lo largo de toda la vida del

sistema y sólo varía cuando la tarea hereda el máximo nivel de apropiación de su conjunto de tareas bloqueantes. Este caso se prolonga únicamente por una instancia de ejecución, siendo su uso impuesto por la política ESRP, a fin de evitar múltiples inversiones de prioridad. Sin embargo, en particular, la Política de Planificación Comportamental presenta una excepción a la regla de que el nivel de apropiación es fijo para cada tarea a lo largo de toda la vida del sistema. De hecho, los servidores BIPS varían dinámicamente su vencimiento y su frecuencia de ejecución. Por lo tanto, en este caso, se verá que el nivel de apropiación se ajusta en base a dichas variaciones.

Para prioridades dinámicas, se tomará como política representativa a EDF. En este caso, los niveles de apropiación de dos tareas τ_i y τ_j deben respetar la siguiente condición:

Condición 3.4.1 (Nivel de Apropiación para EDF)**.** El nivel de apropiación de una tarea τ_i es mayor que el de una tarea τ_j si y sólo si el vencimiento relativo de τ_i es anterior al de τ_j. Esto es,

$$\pi(\tau_i) > \pi(\tau_j) \iff D_i < D_j$$

La Condición 3.4.1 no establece explícitamente cómo se debe determinar el nivel de apropiación de cada tarea. Esto se debe a que, para la política ESRP, este hecho no es relevante mientras se respete la relación expresada en dicha condición. De este modo, el desarrollador tiene absoluta libertad para utilizar la escala que desee y que más se ajuste a sus requerimientos.

Para el caso de servidores BIPS, rige una condición similar. Sin embargo, como se mencionó anteriormente, debe contemplarse que un BIPS puede variar su vencimiento relativo durante la ejecución, con lo cual también varía su prioridad. Por ello, el cálculo del nivel de apropiación, en este caso, resulta más conveniente hacerlo dinámicamente y en base al vencimiento absoluto.

Condición 3.4.2 (Nivel de Apropiación para BIPS)**.** El nivel de apropiación de una tarea

τ_i encapsulada en un servidor BIPS S_r es mayor que el de una tarea τ_j encapsulada en un servidor S_s si y sólo si el vencimiento absoluto de S_r es anterior al de S_s. Esto es,

$$\pi(\tau_i) > \pi(\tau_j) \Longleftrightarrow d_r < d_s$$

Por otro lado, de la condición anterior surge la siguiente relación entre nivel de apropiación y vencimiento de una tarea y de su servidor asociado. Nótese el uso del superíndice d para remarcar el aspecto dinámico del nivel de apropiación.

$$\pi(\tau_i) = \frac{1}{d_i - a_i} = \frac{1}{d_s - r_s} = \pi^d(\tau_i) \leq \frac{1}{D_i} = \frac{1}{P_s} = \pi^{\text{máx}}(\tau_i)$$

Finalmente, para políticas de planificación basadas en prioridades fijas, se toma a RM como destacada. En este caso particular, tanto las prioridades como los niveles de apropiación son fijos para cada tarea a lo largo de toda la vida del sistema. De esto surge, que no sea necesario distinguir entre prioridades y niveles de apropiación. De todos modos, con el espíritu de permitir la mayor libertad de adaptación posible al desarrollador, la siguiente condición establece un criterio general para la elección del nivel de apropiación.

Condición 3.4.3 (Nivel de Apropiación para RM)**.** El nivel de apropiación de una tarea τ_i es mayor al de una tarea τ_j si y sólo si la prioridad de τ_i es mayor que la de τ_j. Esto es,

$$\pi(\tau_i) > \pi(\tau_j) \Longleftrightarrow p(\tau_i) > p(\tau_j)$$

De las tres condiciones planteadas, se puede apreciar que la política ESRP es fácilmente adaptable para trabajar con diversos esquemas de manejo de prioridad. Asimismo, se puede notar que dicha adaptación es simple de realizar, a la vez que deja la suficiente libertad al desarrollador para ajustarla a las características propias de su aplicación.

3.4.3.3. Consideraciones sobre Mecanismos del SOTR

En este apartado se expondrán algunas consideraciones a tener en cuenta al momento de implementar la política ESRP en un SOTR. Las mismas están relacionadas a los mecanismos utilizados por un SOTR genérico y muestran algunas de las ventajas prácticas de utilizar la política ESRP. En particular, se presentarán conceptos referidos al control del presupuesto de un servidor BIPS, a la definición de una sección crítica y sus mecanismos de exclusión mutua y, finalmente, a la situación cuando efectivamente se produce un bloqueo temprano.

En primer lugar, hay un aspecto crítico a tener en cuenta al momento de utilizar servidores BIPS junto con ESRP y es el control del presupuesto Q_s del servidor. En realidad, esto ocurre con muchos tipos de servidores. El problema surge porque la ejecución de un servidor se pospone al agotar su presupuesto. Si dicha situación se produjera mientras la tarea encapsulada por el servidor se encuentra dentro de una sección crítica, se podría producir un bloqueo cuya duración sería potencialmente indeterminada. Se pueden plantear dos maneras de atacar este problema: 1) posponer la ejecución, sin liberar el recurso, con lo cual el tiempo de bloqueo se incrementa en función del tiempo de recarga del servidor; y 2) realizar un control del presupuesto antes de entrar a la sección crítica. Esta segunda alternativa es la que se sigue en esta tesis. Para ello, el desarrollador debería, en primer lugar, establecer un presupuesto suficiente para garantizar la ejecución completa de la sección crítica (esto es, $Q_s \geq \xi_{\text{máx}}$). Y en segundo lugar, realizar un control del presupuesto disponible en ese momento antes de entrar a la sección crítica (esto es, $q_s \geq \xi_{\text{máx}}$). En la práctica, se supone que la primera condición se cumpliría siempre y sólo se debería implementar la segunda. Es decir, antes que la tarea entre a la sección crítica, el servidor BIPS correspondiente debe controlar que su presupuesto actual sea mayor que la duración de la mayor sección crítica declarada por la tarea. En caso contrario, el BIPS debe posponer su ejecución y recargar el presupuesto.

Con respecto las secciones criticas, en forma general, se pueden definir como una pequeña porción de código, donde se acceden a recursos compartidos. Como la clave de una sección

crítica es que dentro de ella se accede a recursos compartidos por varias tareas, se entiende que el uso de esos recursos debe hacerse de manera mutuamente excluyente. En sistemas de tiempo real, las características mencionadas de una sección crítica deben tomarse seriamente en cuenta. Más aún, una mala codificación de la sección crítica o un error en el mecanismo de exclusión mutua puede llevar a inversiones de prioridad no controladas o a la pérdida de un vencimiento. Además, en sistemas de tiempo real, se hace hincapié que dichas secciones estén claramente delimitadas y sean lo más cortas posible.

Una ventaja fundamental del uso de la política ESRP referida a las situaciones planteadas en el párrafo anterior, es que en el caso de una única sección crítica externa, no trivial, sólo se requiere que el desarrollador establezca los límites de la misma, sin necesidad de utilizar ningún otro mecanismo de sincronización como semáforos o monitores. Esto es, únicamente se necesita que el SOTR provea dos primitivas que indiquen el comienzo y finalización exactos de la sección crítica (*e.g.*, `inicioSC()` y `finSC()`). En la Figura 3.3 se ilustra este mecanismo para varias secciones críticas anidadas. La fundamentación del uso de las dos primitivas introducidas previamente viene dada por la utilización del bloqueo temprano. En el caso de secciones críticas consecutivas es necesario contar con mecanismos como semáforos o *mutexes*, con las operaciones clásicas `lock()` y `unlock()`.

```
task  Tarea1{
        ...
        lock(R1);     ⟹  inicioSC();
        ...
        lock(Rx);
        ...
        unlock(Rx);
        ...
        unlock(R1);   ⟹  finSC();
        ...}
```

Figura 3.3: Diferentes definiciones de secciones críticas.

Es preciso resaltar las acciones que debe llevar a cabo el SOTR al invocarse las operaciones mencionadas anteriormente. En primer lugar, al entrar a una sección crítica (ya sea mediante una operación `inicioSC()` o `lock()`), se debe hacer un ajuste de los techos de la tarea y

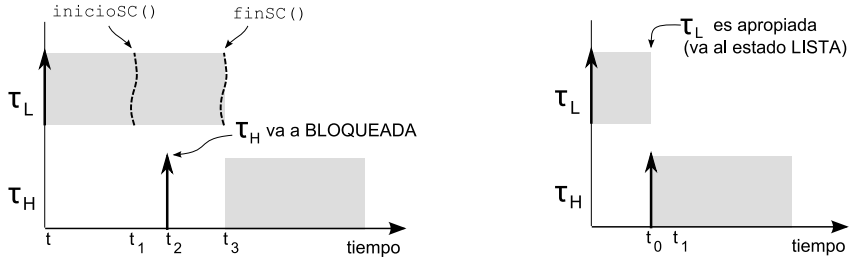
del sistema. La tarea debe *heredar* el techo máximo de su conjunto de tareas bloqueantes. Esto es, $\pi(\tau_i) \leftarrow \overline{\pi}_{\Phi_v}$, con $\tau_i \in \Phi_v$ siendo la tarea que intenta ingresar a la sección crítica. Del mismo modo, se debe ajustar el techo del sistema al valor del nivel de apropiación de la tarea. Es decir, se hace $\overline{\pi} \leftarrow \pi(\tau_i)$. Por otro lado, cuando la tarea τ_i abandona una sección crítica, sea por una operación `finSC()` o `unlock()`, se deben restaurar los valores de nivel de apropiación de la tarea y el del techo del sistema. Esto es, se hace $\pi(\tau_i) \leftarrow \pi_{old}(\tau_i)$ y $\overline{\pi} \leftarrow \overline{\pi}_{old}$, donde π_{old} y $\overline{\pi}_{old}$ representan los valores anteriores, a la entrada de la sección crítica, del nivel de apropiación de la tarea y del techo del sistema, respectivamente. Asimismo, como puede haber ocurrido una inversión de prioridad y estar una tarea bloqueada de mayor prioridad que la actual, se debe hacer una replanificación.

En el párrafo anterior, se enunciaron las acciones que debe llevar a cabo el SOTR en el momento en que se accede a una sección crítica y al salir de ella. En este punto, es necesario responder a una pregunta planteada al momento de presentar la política ESRP: ¿cuándo se hace efectivo el *bloqueo temprano*? En primer lugar, la política ESRP establece que cuando a una tarea se le permite pasar al estado LISTA, los recursos necesarios para su ejecución están disponibles. Sin embargo, eso no significa que se le asignen en ese momento. De hecho, esos recursos le son asignados efectivamente cuando accede a la sección crítica correspondiente a cada recurso (esto es, una vez en el estado EJECUTANDO). De esto surge la utilidad de marcar el inicio de la primera sección critica (para el caso de una única sección crítica más externa, no trivial) y de utilizar la operación `lock()` (en otros casos), ya que una vez que la tarea accede a su primer recurso necesario, ninguna otra tarea puede tomar otro recurso que requiera la tarea que ya obtuvo acceso ese recurso. Es decir, otra tarea de su mismo conjunto de tareas bloqueantes. De lo anterior se desprende que una tarea que no entró aún a su sección crítica puede ser apropiada por otra tarea de su mismo conjunto de tareas bloqueantes, que esté en condiciones de hacerlo. En este sentido, la herencia del máximo nivel de apropiación del conjunto se hace efectiva al momento que la tarea entra a cada sección crítica. Así, todos los bloqueos tempranos que ocurran mientras la tarea está dentro

de su sección crítica se harán sobre tareas que hayan arribado después de que la tarea haya entrado a su sección crítica. A continuación se muestran varios ejemplos de las alternativas planteadas.

Ejemplo: *Supóngase dos tareas τ_L y τ_H ambas pertenecientes al mismo conjunto Υ, tales que $p(\tau_L) < p(\tau_H)$ y $\pi(\tau_L) < \pi(\tau_H)$. En la Figura 3.4(a), se muestra una situación en la cual la tarea τ_L arriba en un tiempo t y comienza su ejecución. En el instante t_1 entra a su sección crítica, con lo cual hereda el máximo nivel de apropiación de su conjunto. Luego, en t_2, arriba τ_H, pero el SOTR la bloquea porque su nivel de apropiación no es mayor que el del conjunto actualmente. Ésta es una situación de bloqueo temprano. Recién en el tiempo t_3, que es cuando τ_L libera sus recursos (sale de su sección crítica), se le permitirá ejecutar a τ_H.*

Por otro lado, la Figura 3.4(b) muestra la situación anterior para el caso de que la tarea τ_H haya arribado en t_0 (anterior a t_1). En dicho caso, como la tarea τ_L no accedió aún a su sección crítica, τ_H la apropia directamente. Nótese que esta última alternativa no involucra ningún bloqueo, ya que τ_H tiene mayor prioridad y nivel de apropiación que τ_L, por lo que es una simple apropiación.



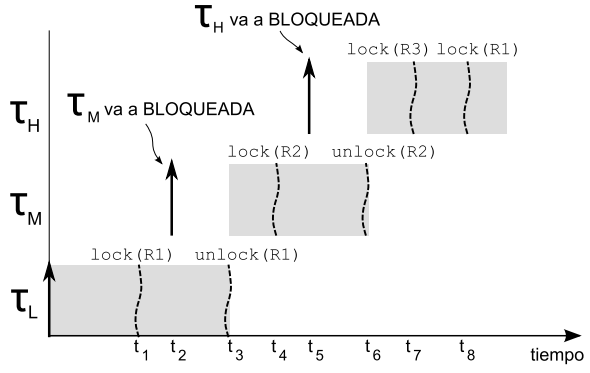
(a) Bloqueo temprano. (b) Apropiación.

Figura 3.4: Situaciones de bloqueo temprano.

Ejemplo: *En la Figura 3.5(a) se muestra una situación más compleja con tres tareas τ_L, τ_M y τ_H pertenecientes al mismo conjunto de tareas bloqueantes Υ, donde $p(\tau_L) < p(\tau_M) < p(\tau_H)$ y $\pi(\tau_L) < \pi(\tau_M) < \pi(\tau_H)$. Además, estas tareas comparten tres recursos: R1, R2 y R3. En*

la Figura 3.5(b) se muestran la secuencias de pedidos de recursos para cada tarea.

Se puede ver que la tarea τ_L arriba y comienza inmediatamente su ejecución. En el tiempo t_1 solicita el recurso R1 y lo obtiene. En t_2 arriba la tarea τ_M, de mayor prioridad y mayor nivel de apropiación, pero como τ_L tomó un recurso, su nivel de apropiación se elevó al máximo del conjunto y τ_M es bloqueada. Cuando, en t_3, τ_L libera R1, los techos del sistema y del conjunto y el nivel de apropiación de la tarea se restauran al valor que tenían antes de que se tome el recurso. Allí, se realiza una replanificación y, en ese instante, el procesador es asignado a τ_M. En un tiempo posterior, t_4, esta tarea requiere R2 y nuevamente se realiza un ajuste de techos y nivel de apropiación, como en el caso anterior. Del mismo modo, en t_5, arriba τ_H, pero es bloqueada hasta que τ_M libere el recurso que tomó. Nótese que tanto la situación de bloqueo en t_2 como ésta en t_5 son, de hecho, bloqueos tempranos, ya que no se permite que la tarea comience su ejecución. La liberación de R2 ocurre en t_6 e inmediatamente se realiza una nueva replanificación, la cual asigna el procesador a τ_H. Luego, sucesivamente τ_H solicita R3 y R1. La ejecución de τ_H continúa hasta su finalización, ya que aunque libere sus recursos sigue siendo la tarea de mayor prioridad y nivel de apropiación. Una vez finalizada esa ejecución, el procesador es asignado a τ_M y τ_L, respectivamente para que completen sus ejecuciones.



(a) Diagrama temporal.

τ_L	τ_M	τ_H
`lock(R1)`	`lock(R2)`	`lock(R3)`
`unlock(R1)`	`unlock(R2)`	`lock(R1)`
`lock(R3)`	`lock(R1)`	`unlock(R1)`
`unlock(R3)`	`unlock(R1)`	`unlock(R3)`

(b) Secuencias de pedido de recursos.

Figura 3.5: Bloqueo temprano entre tres tareas.

Los ejemplos anteriores muestran la importancia que tiene, en sistemas de tiempo real, el hecho de que las secciones críticas sean lo más cortas posibles y que una vez que la tarea no

utilice más sus recursos, los libere. De esta manera, se permite que otras tareas, en particular aquellas con prioridad mayor que la actual, accedan al procesador. Aquí cobra importancia la especificación del final de la sección crítica (tanto por medio de `unlock()` como de `finSC()`), ya que es allí cuando se liberan los recursos, se recupera el nivel de apropiación original y se permite la ejecución de otras tareas del conjunto de tareas bloqueantes. Por otro lado, se debe remarcar que las situaciones planteadas son directamente extrapolables a diversos conjuntos de tareas bloqueantes. En particular, si en el primer ejemplo se considera que $\tau_L \in \Upsilon_v$, $\tau_H \in \Upsilon_u$ y $\pi(\tau_L) < \pi(\tau_H) = \overline{\pi}_{\Phi_u} < \overline{\pi}_{\Phi_v}$; se ve que las situaciones planteadas en la Figura 3.4 continúan siendo válidas para tareas de diferentes conjuntos Υ. De la misma manera, para el caso del segundo ejemplo, si se considera que τ_L y τ_H pertenecen a Υ_v, $\tau_M \in \Upsilon_u$, con $\pi(\tau_L) < \pi(\tau_M) < \pi(\tau_H) = \overline{\pi}_{\Phi_v} < \overline{\pi}_{\Phi_u}$; también se puede verificar que la situación de la Figura 3.5 sigue siendo válida. Es necesario aclarar que, para las extrapolaciones propuestas, se deben reformular los pedidos de recursos, a fin de adaptarlos a las situaciones planteadas. Finalmente, a partir de estos ejemplos (en particular, el segundo) se comprueba concretamente lo afirmado en secciones anteriores acerca del tiempo de bloqueo acotado y que dicho tiempo se limitaba a una única sección crítica.

3.5. Comparación Conceptual con Otros Enfoques basados en SRP

La Política de Recursos Pila Extendida (ESRP) amplía la definición de la política SRP, definida por Baker en 1991 [12]. Esto se logra mediante la determinación de las *relaciones de bloqueo* y la construcción de *conjuntos de tareas bloqueantes*.

Como se ha discutido en la introducción de esta tesis, la problemática de planificar tareas que comparten recursos es crítica en sistemas de tiempo real. Asimismo, también se ha mencionado, que generalmente la investigación sobre dicha problemática se asienta en tres aportes fundamentales: los protocolos techo y de prioridades heredadas [87], que se

enfoca en sistemas basados en prioridades fijas; y la política SRP, para prioridades dinámicas [12]. Estos tres métodos, con algunas modificaciones, continúan siendo los más utilizados a la hora de contemplar recursos compartidos en una planificación de tareas de tiempo real. En particular, diversos autores han trabajado para introducir mejoras y modificaciones a la política SRP. A continuación, se describen y contrastan estos enfoques con la política ESRP, presentada en esta tesis.

En [15], Baruah hace una revisión de la política SRP y plantea nuevos resultados sobre ésta. En primer lugar, propone otra versión del test de factibilidad. Este nuevo test se basa en una *función de demanda*, la cual determina para cada tarea el tiempo de ejecución requerido por ella, en un intervalo cualquiera de tiempo. Este aporte es significativo, ya que permite tomar decisiones en tiempo de ejecución. Sobre todo, si el sistema admite tareas esporádicas, cuyo tiempo de arribo no es conocido. En segundo lugar, prueba que esta nueva formulación es superior a la constituida por una política de prioridades fijas y el Protocolo Techo (PCP) [87]. Finalmente, el autor explora la posibilidad de replicar recursos, en el caso de que el sistema no pase el test de factibilidad. En relación a este último punto, se expone una heurística muy básica que se basa únicamente en aspectos temporales para realizar la replicación. Como argumenta el autor, sólo mirar las características temporales no es suficiente, ya que en un sistema existen recursos no replicables, replicables hasta un cierto número y otros demasiado costosos, en cuyo caso deben considerarse como última instancia para una replicación. Sin embargo, la heurística propuesta no contempla esas situaciones.

En lo referido a recursos compartidos en sistemas basados en mecanismos de reserva, de Niz *et ál* [73] hacen un minucioso estudio teórico-práctico de las características, particularidades y cuestiones a tener en cuenta en esta clase de sistemas. Los autores, en este sentido, analizan las ventajas y desventajas (en particular, en el aspecto práctico) de los protocolos de prioridades heredadas y techo. En la misma línea de trabajo, pero sobre sistemas de prioridades dinámicas basadas en el Servidor de Ancho de Banda Constante (en inglés, CBS) [6], Caccamo y Sha [28] proponen una modificación a la política SRP. Los autores introducen el

concepto de *techo dinámico*, con el cual pueden aplicar SRP en un sistema basado en CBS para tareas aperiódicas. Este concepto es también utilizado en la política ESRP, presentada en esta tesis, ya que facilita la aplicación de ésta con la Política de Planificación Comportamental. El trabajo de Caccamo y Sha, asimismo, es extendido en [27], para considerar un mecanismo de reclamo de ancho de banda entre servidores.

Una línea de investigación que ha tomado fuerza en el último tiempo es la de los *sistemas jerárquicos*. En esta clase de sistemas hay un planificador global, que planifica aplicaciones; y dentro de cada aplicación hay un planificador local que lo hace sobre tareas. En [41] y [18] se trata el problema de planificar tareas con recursos compartidos globales. Esto es, dos o más tareas de diferentes aplicaciones compartiendo recursos. En primer lugar, Davis y Burns, en [41], plantean un esquema de planificación jerárquico de dos niveles (ambos basados en prioridades fijas), donde las aplicaciones son encapsuladas en un *servidor periódico*, el cual ejecuta siempre todo su presupuesto, tenga o no tareas disponibles. Para tratar el acceso a recursos compartidos, introducen la política HSRP (en inglés, política SRP jerárquica). Allí, cada recurso tiene un techo igual a la máxima prioridad de todos los servidores que tienen una tarea que accede a ese recurso. Así, cuando una tarea accede a un recurso compartido la prioridad de ella se eleva al máximo de su aplicación y la de su servidor al máximo techo asociado con el recurso. Asimismo, el mecanismo que los autores utilizan para manejar la situación de que un servidor agote su presupuesto mientras una tarea está en una sección crítica, es peligroso y puede traer aparejado la pérdida de vencimientos. En este caso, los autores proponen que dicha tarea continúe su ejecución hasta que libere el recurso, lo cual puede generar un tiempo de bloqueo no acotado y la consecuente pérdida de vencimientos. Por otro lado, el mecanismo de control del presupuesto antes de comenzar a ejecutar en una sección crítica, utilizado en esta tesis al implementar la política ESRP junto con servidores BIPS, resulta más seguro y confiable [28].

El trabajo que resta analizar, dentro de los sistemas jerárquicos basados en la política SRP, es el realizado por Behnam *et ál* [18], denominado SIRAP. Al igual que el propuesto

por Davis y Burns, el esquema de planificación es de dos niveles y los autores utilizan servidores para encapsular aplicaciones. Asimismo, el trabajo se enfoca en recursos globales que comparten tareas de diferentes aplicaciones. En este sentido, ambos trabajos presentan un modelo de sistema muy similar. En cuanto a la factibilidad, SIRAP también se orienta en servidores (que ellos llaman *procesador virtual*), para encapsular las aplicaciones. Al igual que en esta tesis, Behnam *et ál* posponen la ejecución de un servidor que probablemente agote su presupuesto dentro de una sección crítica. Asimismo, Behnam *et ál*, también proponen un esquema alternativo para la asignación de los techos a los recursos globales, que consiste en asignarles a éstos el máximo nivel de apropiación de todos los servidores. Los autores explican que este método de asignación simplifica la implementación, pero "no es tan eficiente como el original" propuesto por Baker en [12].

Finalmente, los trabajos de Gai *et ál* ([49] y [50]), aportan nuevos resultados en las bases de la política SRP aplicada a sistemas multiprocesadores. Aunque el modelo de sistema diste bastante de presentado para la política ESRP, estos aportes resultan interesantes y útiles de destacar. El resultado fundamental de los trabajos de Gai *et ál* es la construcción de conjuntos de tareas no apropiativas, mediante la definición de *pseudo-recursos* y *umbrales de apropiación*. A pesar de cumplir, gracias a estas modificaciones, con el objetivo de la investigación realizada por los autores (*i.e.*, minimizar el uso de memoria RAM).

3.6. Síntesis del Capítulo

La motivación de este capítulo fue proveer una política de manejo de recursos compartidos para sistemas de tiempo real, que resuelva las problemáticas que surgen en dichas situaciones y pueda ser aplicada a lo largo de todo el proceso de desarrollo de software. En términos generales, se propuso una extensión a la política clásica SRP, denominada ESRP, mediante la inclusión de los conceptos de conjunto de tareas y de recursos. En particular, se realizaron los siguientes aportes:

- Se extendió un esquema de manejo de recursos compartidos por medio de la definición de la relación de bloqueo entre tareas.
- Se propuso un método eficiente para analizar únicamente a aquellas tareas que potencialmente pueden presentar conflictos de recursos.
- Se expusieron aspectos prácticos a tener en cuenta al momento de implementar un sistema basado en esta nueva formulación.
- Se mostró que la política ESRP es fácilmente adaptable y que puede coexistir con esquemas basados en prioridades fijas, dinámicas y con la Política de Planificación Comportamental.
- Se realizó un análisis comparativo con otros enfoques basados también en SRP.

Capítulo 4

Marco de Software

En general, las políticas de planificación de tiempo real suelen presentarse como modelos analíticos o algoritmos incompletos, que muestran su implementación desde un punto de vista teórico y simplificado. Esto dificulta, en muchos casos, que dichas políticas sean llevadas a la práctica, debido a las consideraciones y suposiciones que deben hacerse y son propias de un entorno real. Con el objetivo de acercar los conceptos teóricos presentados en los dos capítulos anteriores, en éste se propone un marco de software genérico que facilite la implementación de dichos aportes. El marco propuesto presenta diferentes visiones de diseño, a fin de que el desarrollador cuente con varios puntos de vista que faciliten la comprensión y adaptación del mismo a sus necesidades. En particular, este capítulo aborda los siguientes temas:

- *Se desarrolla un modelo basado en contratos, para mostrar la interacción de las diferentes entidades involucradas en el sistema.*
- *Se presenta un modelo estructural, donde las entidades son descriptas estáticamente y en forma detallada.*
- *Se describe un modelo dinámico de cada entidad involucrada en el sistema, donde se muestran los diferentes estados que atraviesa cada una en particular.*

4.1. Motivación

El diseño de software para sistemas embebidos todavía sufre los problemas tradicionales de la ingeniería de software, entre los que se destacan la correcta descripción de arquitecturas y el mantenimiento del sistema [99]. En particular, los sistemas embebidos de tiempo real son un campo de la computación que involucra mayormente dos disciplinas disociadas: la teoría de control, que trata los requerimientos no funcionales impuestos por el ambiente físico; y la ingeniería de software, que se relaciona a los requerimientos funcionales dados por la aplicación [54]. La concepción de una arquitectura de software capaz de expresar tanto los aspectos funcionales como los no funcionales de un sistema es entonces de fundamental importancia. Esta arquitectura debería proveer una manera estándar y ajustable de describir sistemas de tiempo real. Así, una ventaja de tener dicha arquitectura genérica es la posibilidad de aplicar la teoría de planificación de tiempo real [88] a un diseño de software, a fin de determinar si es factible.

Una arquitectura de software generalmente se asocia a la estructura "gruesa" de un sistema [51]. Está estructura es estática y, aunque permite y alienta su reutilización, no escapa de la aplicación particular para la que fue desarrollada. Una alternativa para presentar un diseño de software genérico y que favorezca la adaptabilidad y escalabilidad del mismo es mediante lo que se denominará *marco de software*[1]. Dicho marco de software es una forma de reutilización del software que principalmente promueve la readaptación de arquitecturas completas dentro de un dominio de aplicación bien delimitado [77]. De otra manera, se puede ver a los marcos de software como generadores de aplicaciones que están directamente relacionadas con un dominio específico (*i.e.*, una familia de problemas relacionados) [67]. De lo anterior, se desprende que el marco de software es un concepto más abarcador que el de arquitectura y hasta lo incluye. Asimismo, el marco de software posee ciertas características que lo distinguen aún más de una arquitectura: 1) se representa mediante un conjunto de

[1]El término *marco* se toma como traducción de la palabra inglesa *framework*, la cual es bien conocida en la jerga de ingeniería de software.

clases abstractas y a través de la forma en que sus instancias deberían interactuar [58, 70]; 2) su implementación se realiza por medio de *puntos calientes* (*hot spots*) que son aquellas clases o métodos abstractos declarados que deben implementarse [67]. En relación a este último punto, un marco de software establece lo que se conoce como *inversión de control*. Este mecanismo se contrapone al paradigma clásico de la programación procedural, en el que el llamador es quien controla cuándo y cómo responderá quien es llamado. Por el contrario, en un marco de software el que es llamado (generalmente son aquellas clases que implementan puntos calientes) declara frente a la ocurrencia de cuál evento quiere ser llamado [48]. De esta manera, el marco de software permite una mayor adaptación a diversas aplicaciones.

Dado que los marcos de software necesitan ser descriptos de la manera más estándar posible, una elección apropiada para este objetivo es el Lenguaje Unificado de Modelado (UML, por la sigla en inglés) [4]. UML ha demostrado ser la notación más utilizada para describir sistemas complejos. Este hecho se puede ver en la cantidad de artículos, libros y principalmente diseños de la industria que adoptan a UML como lenguaje de modelado, lo cual lo ha convertido en un *estándar de facto*. Sin embargo, para el caso concreto de sistemas embebidos de tiempo real, el estándar UML tiene que adaptarse para hacer frente a las características particulares de este tipo de sistemas. En este sentido, el *Object Management Group* [3] publicó un perfil a fin de adaptar UML al manejo de sistemas embebidos de tiempo real: el perfil para Modelado y Análisis de Sistemas Embebidos de Tiempo Real (MARTE, por la sigla en inglés) [2].

4.1.1. Contribución

Este capítulo presenta un marco de software, denominado BIPS-SF (por la sigla en inglés), que se ajusta a las características mencionadas anteriormente acerca de la necesidad de éstos en sistemas embebidos de tiempo real. El marco BIPS-SF tiene por objeto proporcionar un diseño genérico y reutilizable para facilitar la aplicación de los conceptos de planificación presentados en los capítulos 2 y 3 de una manera concreta. Asimismo, la introducción de

un marco de software refuerza los conceptos sobre planificación mencionados, al acercar los conceptos teóricos a la implementación práctica. Es de destacar, que para la descripción de BIPS-SF se utiliza UML y el perfil MARTE, lo cual permite su implementación final en diferentes paradigmas de programación como expresa dicho perfil. Por otro lado, se debe notar que lo presentado en este capítulo no intenta ser una descripción exhaustiva ni minuciosa del marco BIPS-SF, sino una presentación suficiente sobre los aspectos más relevantes y sobre la interacción de las entidades que lo componen. Esto refuerza el espíritu de los capítulos anteriores, en cuanto a no condicionar al desarrollador a un esquema rígido y permitirle libertad para aplicar los conceptos a su aplicación particular.

4.2. Lenguaje Unificado de Modelado

En esta sección se puntualizará sobre algunos conceptos de UML y MARTE que serán útiles en el resto del capítulo. Lo mencionado aquí intenta ser un simple repaso y no una explicación detallada. Información más completa se puede consultar en [4].

El Lenguaje Unificado de Modelado es un lenguaje de diseño muy utilizado en el campo de la ingeniería de software. Se lo puede definir como un lenguaje de propósito general que usa constructores gráficos para crear un modelo abstracto. Este modelo abstracto representa al sistema que se está tratando. Una de las razones por las que UML se ha convertido en un lenguaje de modelado estándar es que es independiente del lenguaje de programación en el que se implemente finalmente el modelo. Sin embargo, se debe remarcar que UML no se limita simplemente al modelado de software. Puede usarse también para construir modelos de ingeniería de sistemas, procesos de negocios y estructuras de organizaciones.

El modelado es el diseño de aplicaciones antes de traducirlas a código. En este sentido, UML permite modelar aplicaciones desde diferentes puntos de vista (comportamiento, estructura, punto específico en el tiempo, etc.). Para cada uno de esos puntos de vista, UML propone un diagrama particular. En especial, para la descripción de BIPS-SF, se utilizarán

dos diagramas: el de clase y el de estado. El primero es un tipo de diagrama estático que describe la estructura de un sistema mediante sus clases, atributos y relaciones entre esas clases. El diagrama de clase muestra cómo las diferentes entidades se relacionan entre sí. Por otro lado, un *diagrama de estados* modela los diferentes estados en que puede estar una clase y cómo esa clase se mueve entre sus estados.

4.2.1. Perfil para Sistemas de Tiempo Real

El Lenguaje Unificado de Modelado es un lenguaje extensible [3]. Básicamente, provee dos herramientas que se pueden usar para adaptarlo a necesidades específicas: los estereotipos y los perfiles. Los perfiles son subconjuntos de UML desarrollados particularmente para algún propósito específico. De este modo, UML brinda mecanismos de adaptación a sí mismo con el objetivo de afrontar las necesidades propias de un determinado dominio de aplicación.

Un *requerimiento funcional* define una función de un sistema de software o de un componente del mismo. Por otro lado, un sistema tiene ciertas restricciones que no son funcionales (*e.g.*, tiempo máximo de respuesta, cantidad de memoria mínima requerida, cuán a menudo se debe hacer una copia de seguridad, etc.). Esta clase de restricciones se denominan *requerimientos no funcionales*. De este modo, mientras los requerimientos funcionales describen *qué hace* un sistema; el conjunto de requerimientos no funcionales describen *cómo debería ser* dicho sistema. Así, surge que los requerimientos no funcionales son tan importantes como los funcionales en el desarrollo de un sistema. En particular, para el caso de sistemas embebidos de tiempo real, la correcta consideración de estos requerimientos determina, entre otros, la pérdida o no de vencimientos y el adecuado manejo del tiempo dentro del sistema.

En este sentido, el perfil de UML para Modelado y Análisis de Sistemas Embebidos de Tiempo Real (MARTE) [2] se utilizará a fin de expresar los aspectos no funcionales del marco BIPS-SF y, a la vez, identificar claramente las entidades destacadas del mismo. El perfil MARTE se basa en estereotipos que proveen la capacidad necesaria para modelar conceptos específicos del dominio de aplicación. Además, estos estereotipos pueden expresar *restric-*

ciones y, al mismo tiempo, tienen propiedades explícitas llamadas *etiquetas de definición* (en inglés, *tag definitions*), que representan atributos o relaciones. En particular, MARTE fue desarrollado para contemplar y expresar las características propias de los sistemas embebidos de tiempo real (*e.g.*, vencimientos, activaciones periódicas, apropiaciones, recursos compartidos, etc.) y denotar sus entidades más importantes (*e.g.*, tareas, planificador, etc.) [42].

Al analizar un sistema particular, generalmente un método de análisis no se aplica en forma directa y completa a dicho sistema. En este sentido, todos los métodos de análisis usan una visión simplificada y abstracta del sistema a analizar, la cual se centra en aquellos aspectos que son relevantes a la técnica y al propósito del análisis [47]. De esta forma, a los propósitos de este trabajo, el perfil MARTE se adopta parcialmente, sólo con el objetivo de expresar las restricciones temporales típicas en sistemas embebidos de tiempo real y para tipificar las entidades del marco BIPS-SF.

4.3. Modelo de Contrato

En el ámbito legal, un contrato establece los derechos y obligaciones entre varias partes al momento de llevar a cabo una tarea. Estos conceptos fueron tomados originalmente por Bertrand Meyer [71] y adaptados al diseño de software. La idea básica, de este método basado en contratos, es que, en general, dos entidades de software relacionadas establezcan claramente las condiciones necesarias para su correcta colaboración. Del mismo modo, un contrato es algo en que están de acuerdo ambas partes y que pueden mutuamente cambiar a lo largo del tiempo. Este método basado en contratos es de gran uso en la actualidad y en diversos dominios de aplicación. En particular, para sistemas embebidos de tiempo real se han realizado varios esfuerzos [9, 1] con el objetivo de adaptar los conceptos generales a la problemática de los mencionados sistemas. En todos los casos, la idea central que subyace es la misma: establecer claramente los requerimientos de la aplicación y garantizar los recursos

necesarios para cubrir esos requerimientos. En este sentido, el modelo de contrato que se presenta a continuación puede verse que tiene el mismo espíritu que los citados anteriormente, aunque con un enfoque más específico.

Con el objetivo de introducir el funcionamiento general del marco BIPS-SF, un modelo de él, basado en el esquema propuesto en [84], se muestra en la Figura 4.1. En la figura se puede ver que hay un único procesador y un conjunto de recursos, todos manejados por un planificador. Los servidores BIPS, las tareas blandas y las duras requieren el procesador y establecen un contrato de Calidad de Servicio (QoS) para usarlo. Estos contratos establecen los requerimientos no funcionales de las diferentes entidades al momento de competir por dicho procesador (en la próxima sección, esta situación se aclarará aún más mediante el uso del perfil MARTE). Los recursos son pedidos por las tareas blandas y duras. Para esto, el planificador provee mecanismos que controlan el acceso a los mismos mediante la sincronización de las tareas duras y los servidores BIPS siguiendo la política ESRP. A diferencia del esquema propuesto en [84], el manejador de recursos y el agente (*broker*) de recursos, aquí son la misma entidad: el planificador. Por otro lado, el planificador asigna el procesador a las tareas duras y a los servidores BIPS de acuerdo a una Política de Planificación (*i.e.*, EDF); y administra los recursos por medio de una Política de Contención (*i.e.*, ESRP). Estas dos políticas se conjugan en una sola, como se explicó en el Capítulo 3. Asimismo, el Contrato de QoS establece la forma en que se debe asignar el procesador tanto a las tareas duras como a los servidores BIPS. Respecto a las tareas blandas, el esquema es similar al anterior, excepto que la entidad que administra dicho contrato de QoS es el servidor BIPS que encapsula a la tarea. En este punto, es donde se hace visible el esquema jerárquico que impone la Política de Planificación Comportamental a través de los servidores BIPS.

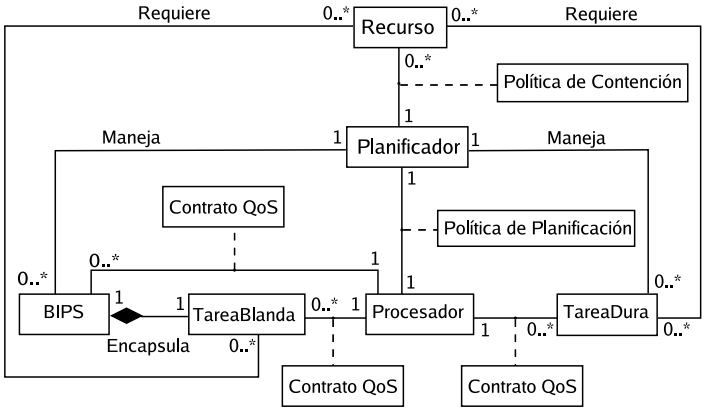


Figura 4.1: Modelo de contrato del marco BIPS-SF.

4.4. Modelo Estructural

Como se mencionó en la introducción de este capítulo, un marco de software incluye en su definición una descripción de su estructura o arquitectura interna. Esta arquitectura ayuda a que el desarrollador tenga una visión global del marco, pueda determinar si se ajusta a sus requerimientos y cuente con un plano general para construir su propia aplicación a partir del mismo [51]. En base a lo anterior, la arquitectura del marco BIPS-SF se muestra en la Figura 4.2. A continuación se analizan los puntos fundamentales de dicha arquitectura.

En primer lugar, la figura muestra que las clases TareaDura y TareaBlanda heredan de la clase Tarea las definiciones abstractas de métodos y atributos genéricos comunes a todas las tareas (*i.e.*, atributos como el período, el identificador, y métodos necesarios para obtener y modificar dichos atributos, además del método genérico `realizarFuncion()`, que implementaría la función de la tarea). Se debe remarcar, que el método abstracto `realizarFuncion()` es un punto caliente, que debe implementarse de acuerdo a la actividad que desempeñe la tarea y a la aplicación particular que se desarrolla. La clase TareaBlanda es manejada por la clase BIPS. Esto se simboliza mediante el uso de una relación de composición, ya que ambas clases tienen tiempos de vida coincidentes. En el caso de las clases TareaDura y BIPS ambas son manejadas por la clase PlanificadorEDF_ESRP, la cual se encarga de realizar las acciones de planificación.

Como se mencionó anteriormente, el perfil MARTE permite una definición precisa de cada una de las clases involucradas en el marco BIPS-SF. Las clases TareaDura, TareaBlanda y BIPS están anotadas con el estereotipo ≪SwSchedulableResource≫, el cual indica que esas entidades compiten por el procesador y deben enlazarse a un planificador. Esto se muestra a través de la relación de composición entre las clases TareaDura y BIPS con la PlanificadorEDF_ESRP; y entre la clase TareaBlanda y BIPS, la cual está anotada, a la vez, como ≪SecondaryScheduler≫. Se debe notar que aquí es donde el esquema jerárquico de la Política de Planificación Comportamental se ve claramente, ya que BIPS es una entidad planificable para PlanificadorEDF_ESRP y un planificador de segundo nivel para TareaBlanda.

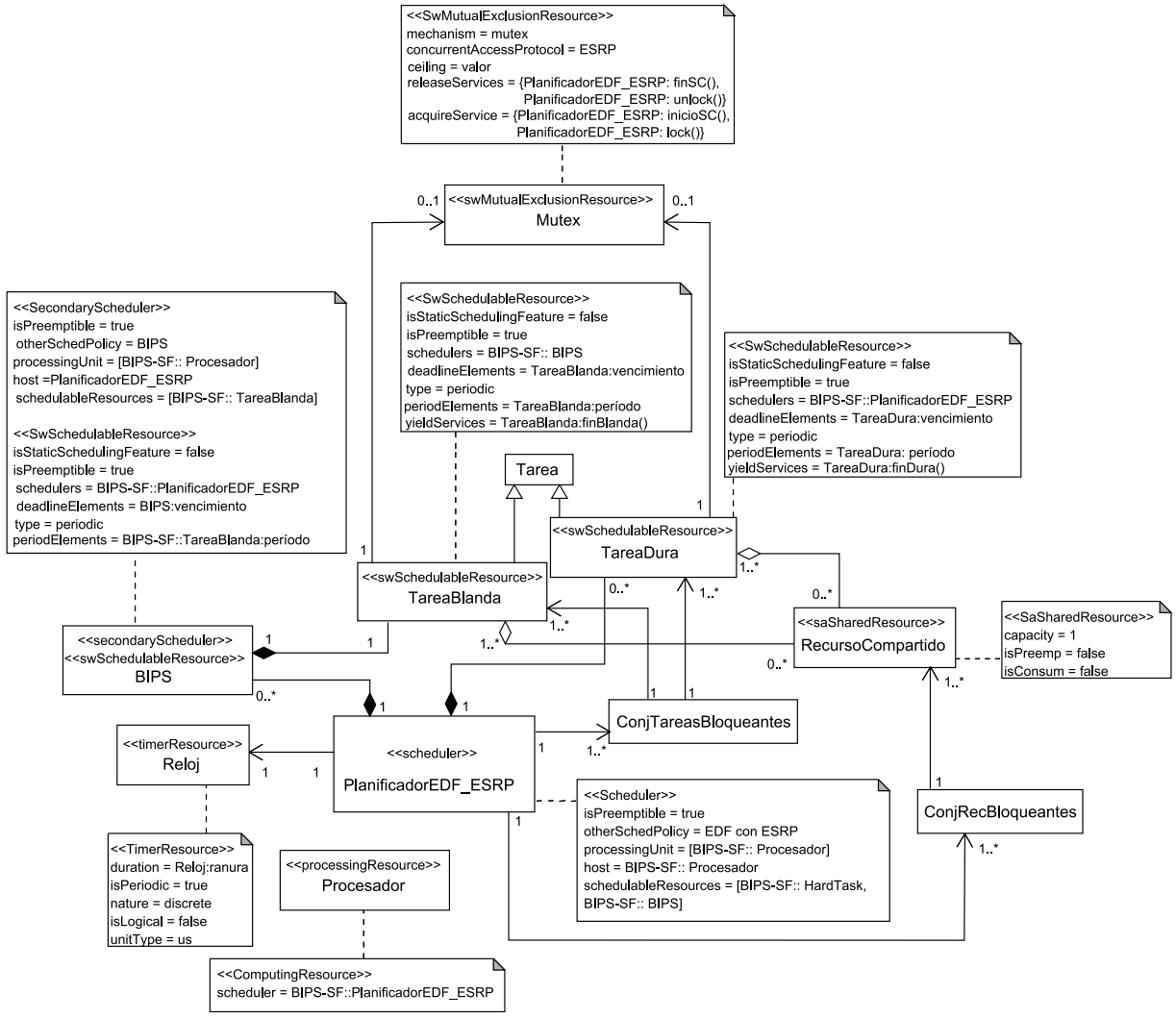


Figura 4.2: Modelo estructural del marco BIPS-SF.

Hasta ahora, nada se ha dicho respecto a los recursos compartidos. De hecho, tanto la clase TareaDura como TareaBlanda pueden tener asociados varios recursos compartidos (esto es, mediante la relación de agregación con la clase RecursoCompartido), los cuales pueden también ser compartidos por diversas tareas. El planificador general (clase PlanificadorEDF_ESRP) es quien implementa la política ESRP que garantiza las propiedades de tiempo real ante la presencia de recursos compartidos. Para ello, además de manejar las tareas, administra dos clases especiales, ConjRecBloqueantes y ConjTareasBloqueantes, las cuales proveen los métodos necesarios para la correcta aplicación de la política ESRP. En particular, la clase ConjTareasBloqueantes proporciona las facilidades para determinar cuál tarea pertenece a qué conjunto, cuál es el techo máximo y actual de un determinado conjunto, la cantidad de tareas del conjunto en estado de bloqueo, entre otras. Por otro lado, tanto la clase TareaDura como TareaBlanda tienen (si utilizan algún recurso compartido) asociada una clase Mutex que implementa los mecanismos de exclusión mutua descriptos en la Sección 3.4.3.3. Es decir, en el caso de que la tarea tenga una única sección crítica más externa, no trivial: `inicioSC()` y `finSC()`; y en el caso de secciones críticas consecutivas: `lock()` y `unlock()`.

Finalmente, se puede ver que el marco BIPS-SF de la Figura 4.2 respeta el modelo de contratos representado en la Figura 4.1. No solamente por las relaciones entre las principales entidades, esto es las clases, TareaDura, TareaBlanda, BIPS, RecursoCompartido, Procesador y PlanificadorEDF_ESRP; sino fundamentalmente por la descripción de las diferentes políticas y los contratos de QoS. Con respecto a este último punto, el uso del perfil MARTE fue de gran utilidad, ya que permitió especificar de forma precisa las características no funcionales de las entidades, las políticas de planificación y manejo de recursos y la manera en que se relacionan dichas entidades.

4.5. Modelo Dinámico

En esta sección se presentan los modelos dinámicos de las principales clases que componen el marco BIPS-SF. En particular, las clases PlanificadorEDF_ESRP, BIPS, TareaBlanda y TareaDura. Con el objetivo de mostrar dicho comportamiento dinámico se utilizarán Diagramas de Estado de UML. Se debe remarcar que los diagramas son esquemáticos e intentan mostrar los principales aspectos de las clases mencionadas durante el tiempo de ejecución. Nótese que algunas transiciones no tienen disparador, esto se hizo para mantener las figuras sencillas y legibles.

El diagrama de estados correspondiente a la clase PlanificadorEDF_ESRP se muestra en la Figura 4.3. Luego de la inicialización del sistema, se entra a un estado en el cual se realizan acciones de administración interna, el estado **MANTENIMIENTO**. Dentro de este estado, primero se controlan los vencimientos de las dos entidades manejadas por la clase (*i.e.*, servidores BIPS y tareas duras). Esta acción se realiza en el subestado **CTRLVC**, el cual envía la señal *VencimPerd* a aquellas tareas que efectivamente perdieron un vencimiento. Luego, en el subestado **CTRLPRE**, se hace una comprobación de los presupuestos de los diferentes servidores BIPS. En este subestado, se envían las señales *PresupAgotado* y *PresupRecargado*, que respectivamente indican que el servidor agotó su presupuesto y debe esperar una recarga, y que dicha recarga se hizo efectiva. La revisión de las reactivaciones de los servidores BIPS y las tareas duras se realiza en el subestado **CTRLREAC**. Allí, se capturan las señales *EstaReact* que hayan enviado las entidades manejadas por PlanificadorEDF_ESRP. Esas entidades que se reactivaron son, junto a otras que ya estuvieran activas, tenidas en cuenta para la planificación. El último subestado es **CTRLTE** y tiene que ver con el control del techo de aquellas entidades que compiten por el procesador. En este subestado es donde se realiza el test presentado en el Capítulo 3 y se envía la señal *IrBloqueada* a aquellas entidades que no lo superan. Las otras pasan a formar parte de las tareas listas para ejecutar.

Una vez completados los controles internos y determinadas las tareas que están en condiciones de competir por el procesador, el planificador efectivamente realiza la planificación.

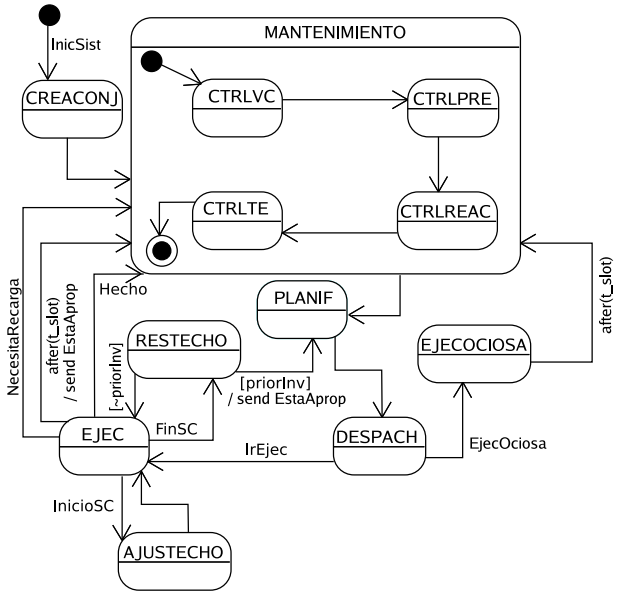


Figura 4.3: Modelo dinámico del planificador EDF-ESRP.

Esto se hace en el estado **PLANIF**. De hecho, esta actividad involucra buscar entre las tareas listas aquella con el vencimiento más próximo al tiempo actual. El despacho de la tarea se modela mediante el estado **DESPACH**, donde se envía la señal *IrEjec* a la entidad correspondiente. Luego de esto, el planificador pasa a un estado **EJEC** que simboliza que hay una tarea ejecutándose. Asimismo, una vez en este estado, el planificador puede pasar al estado **AJUSTECHO**, donde se hereda el techo máximo del conjunto de tareas bloqueantes al que pertenece la tarea. Esta situación se da cuando la tarea entra a una sección crítica (ver Sección 3.4.3.3) y el evento que dispara la transición es *InicioSC*. Del mismo modo, cuando una tarea sale de su sección crítica, se debe restaurar su techo original a fin de permitir que, si hubo una inversión de prioridad, ésta se revierta inmediatamente. La situación descripta se muestra mediante las transiciones al estado **RESTTECHO**. Allí se puede ver el uso de una condición de guarda a través de la variable `priorInv`, la cual simboliza si hubo una prioridad invertida o no. Por otro lado, en caso de que no haya ninguna entidad para ejecutar, se ejecuta la tarea ociosa (en inglés, *dummy*) del sistema. Esta situación se muestra mediante la transición al estado **EJECOCIOSA** disparada por la señal *EjecOciosa*. Finalmente, un aspecto clave de este diagrama de estado es el uso del evento *after* como

disparador de algunas transiciones. Esta cláusula es la que establece una base de tiempo en el sistema y realiza una interrupción periódica, a fin de establecer una base de tiempo en el sistema. Luego de esa transición, el planificador apropia a la tarea actual (mediante el envío de la señal *EstaAprop*) y comienza nuevamente el ciclo explicado. Finalmente, este reinicio del ciclo de vida del planificador, se puede dar también por la finalización de la ejecución de una tarea (esto es, por la señal *Hecho*); o mediante la señal *NecesitaRecarga*, la cual es enviada por un servidor BIPS que no dispone de presupuesto actual suficiente para ejecutar la sección crítica completa de su tarea encapsulada.

En la Figura 4.4, se muestra el modelo dinámico de la clase BIPS. El enfoque propuesto por la Política de Planificación Comportamental establece un esquema de planificación jerárquico. En consecuencia, el modelo dinámico de esta clase es similar al del planificador principal. En la figura, se puede ver este hecho referido a las acciones de mantenimiento interno. En particular, se controla el vencimiento de la tarea encapsulada (subestado **CTRLVC**); las activaciones (subestado **CTRLREAC**). En esos subestados se envía la señal *VencPerdBlanda* y se recibe la señal *EstaReactBlanda*, respectivamente. Luego, si efectivamente la tarea blanda está lista, se la ejecuta. Del mismo modo que el planificador principal, también se realizan acciones referidas al manejo del techo cuando la tarea blanda intenta entrar a una sección crítica. Así, previo control de que el presupuesto disponible sea suficiente para ejecutar la sección crítica completa, se avisa al planificador principal que la tarea intenta entrar a su sección crítica, a fin de ajustar el techo del servidor. Del mismo modo, se da aviso cuando la tarea finaliza la ejecución dentro de la sección crítica. Finalmente, cuando la tarea blanda termina, le avisa de ese hecho a su servidor mediante la señal *HechoBlanda* y el servidor toma la información del comportamiento de la tarea. Con esa información pasa al estado **ADJPARMS**, donde ajusta sus parámetros y determina cuándo se debe reactivar nuevamente. Es decir, computa la función de postergación.

Con respecto al estado **ESPERA**, éste modela la situación en que un servidor agotó su presupuesto y tiene una ejecución pendiente de su tarea encapsulada. En este caso, la espe-

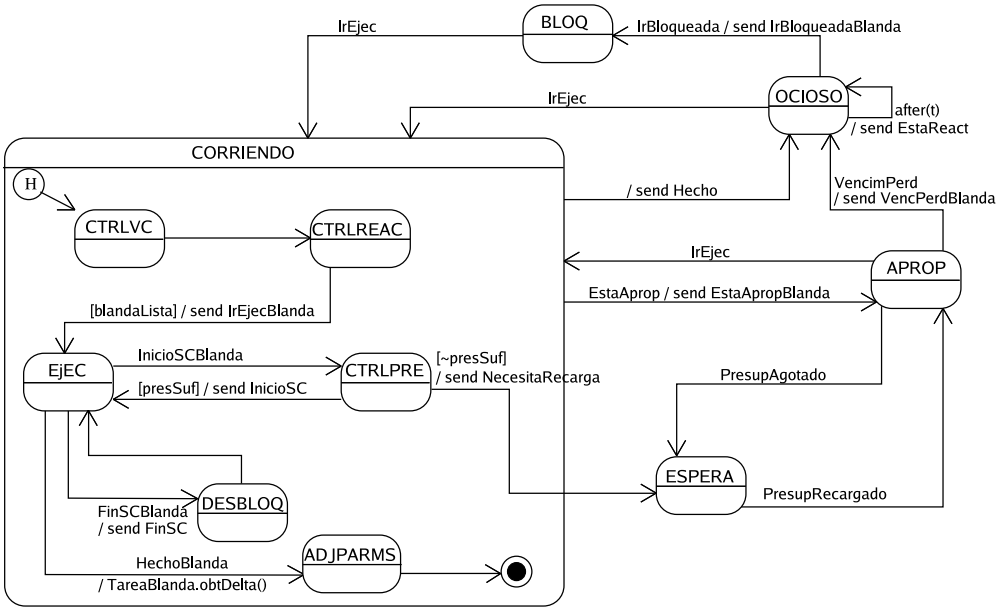


Figura 4.4: Modelo dinámico de un servidor BIPS.

ra se refiere al tiempo que debe transcurrir hasta que su presupuesto pueda ser recargado. En cuanto a los estados **OCIOSO**, **BLOQ** y **APROP**, éstos tienen el mismo significado que para el caso de las tareas, que se explican a continuación. Cabe destacar, que el servidor BIPS fue anotado, según el perfil MARTE, como ≪SecondaryScheduler≫ y como ≪SwSchedulableResource≫. El primer caso, fue descripto recientemente y el segundo es análogo al comportamiento de las tareas (la tarea dura, en particular), que se describe a continuación.

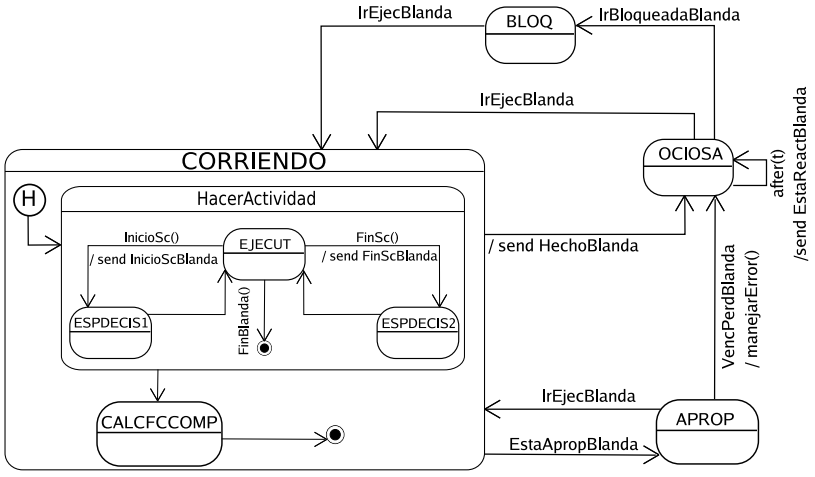


Figura 4.5: Modelo dinámico de una tarea blanda.

El comportamiento en tiempo de ejecución de las tareas, tanto blandas como duras, se muestra en la Figura 4.5 y la Figura 4.6, respectivamente. En las figuras, se puede ver que las tareas se mueven entre cuatro posibles estados. Inicialmente se encuentran en el estado **OCIOSA**, donde esperan el tiempo necesario para poder reactivarse. Como se mencionó anteriormente, esta espera se simboliza mediante el uso de la cláusula *after*. Cuando se reactivan, envían una señal a su correspondiente planificador haciéndole saber de esa situación. Luego, pueden pasar al estado **BLOQ**, si no cumplen los requisitos establecidos por el test de la política ESRP; o bien, van al estado **CORRIENDO** y comienzan su ejecución. Se debe remarcar la adición del sufijo "Blanda" a todas las señales que involucran a la tarea blanda. Ésto tiene el propósito de distinguir aquellas señales que involucran al planificador secundario, de aquel que es principal. Una vez que las tareas comienzan su ejecución, entran a un subestado **HACERACTIVIDAD**. Allí, es donde ejecutan sus acciones correspondientes. Asimismo, se distinguen otros dos subestados que se relacionan con la entrada y salida de sus secciones críticas. Por otro lado, en cualquier momento de la ejecución las tareas pueden ser apropiadas por otras tareas o por el planificador principal. Esta situación se denota mediante la transferencia al estado **APROP**. En este sentido, se debe remarcar el uso del estado de historia superficial tanto en las tareas como en el servidor BIPS. Este estado intenta enfatizar el hecho de que aún cuando una ejecución se vea interrumpida por una apropiación, cuando la ejecución se reanude no se comenzará el ciclo desde cero sino desde el punto en que fue detenida. De otro modo, cuando las tareas terminan envían una señal a sus respectivos planificadores y pasan nuevamente al estado **OCIOSA**, donde esperan su reactivación. En el caso particular de las tareas blandas, éstas antes de terminar su ejecución pasan por un subestado más, el subestado **CALCFCCOMP**, donde calculan su función de comportamiento y ponderan el mismo, para que el servidor BIPS pueda determinar la duración del período de la próxima reactivación. Por otra parte, los dos modelos de tareas presentados aquí, contemplan la situación en la cual una tarea pierde un vencimiento. Esta situación indeseada se puede deber a diferentes factores, pero es importante que este

contemplada en el modelado. Aquí simplemente se indica que ante el aviso por parte del planificador principal o del secundario de esa pérdida se ejecute la acción `ManejarError()`, la cual debe ser adaptada a los requerimientos particulares de cada aplicación.

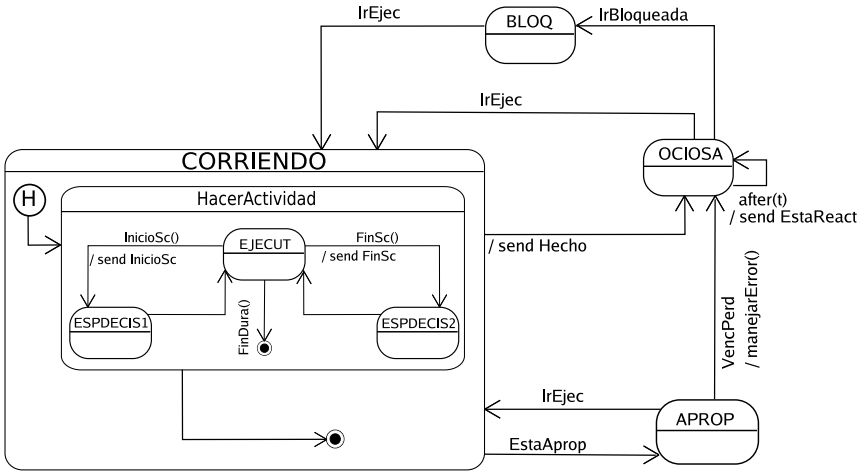


Figura 4.6: Modelo dinámico de una tarea dura.

Para concluir esta sección, se hará una breve nota acerca de la correspondencia entre los modelos de estados presentados en la Sección 2.3 y la Sección 3.4.3.1 y los presentados aquí. En el caso particular de un servidor BIPS, se debe recordar que, en su formulación del Capítulo 2, su ciclo de vida se movía entre tres estados: INACTIVO, ACTIVO y ESPERANDO. La semántica de estos estados se sigue conservando en el modelo presentado en este capítulo. Sin embargo, esos estados se refinaron para representar con mayor fidelidad un comportamiento real. Así, el estado INACTIVO se corresponde directamente con el **OCIOSO**, presentado aquí. El estado ESPERANDO hace lo mismo con el estado **ESPERA**. Y el estado ACTIVO es el que se ve discriminado en otros tres estados. Se debe destacar, que el estado ACTIVO representaba la situación en la cual el servidor BIPS tenía una tarea pendiente de ejecución y su presupuesto era mayor que cero. De este modo, nótese que los estados **BLOQ**, **APROP** y **CORRIENDO** cumplen con esa condición, pero distinguen otras situaciones específicas.

Finalmente, en el caso de las tareas la correspondencia es más directa. Los estados presentados en el Capítulo 3, OCIOSA, BLOQUEADA y EJECUTANDO, se corresponden en

forma clara con **OCIOSA**, **BLOQ** y **CORRIENDO**, respectivamente. En cuanto al estado LISTA del Capítulo 3, la correspondencia con el estado **APROP**, presentado aquí, se da una vez que la tarea es admitida para ejecución. Esto es, cuando la tarea se reactiva, está en el estado **OCIOSA** y permanece allí hasta que efectivamente debe ir a ejecutar o bloquearse, con lo cual no se distingue el hecho de estar en un estado en el cual estarán también varias tareas y simbolice la competencia por el procesador (al menos hasta que comience su ejecución). En las sucesivas apropiaciones que pudiera sufrir la tarea, la semántica de ambos estados es idéntica.

4.6. Síntesis del Capítulo

Como se discutió a lo largo de este capítulo, el desarrollo de software para sistemas embebidos de tiempo real requiere de cuidados y consideraciones especiales. Esta disciplina agrega una dimensión más a los problemas que usualmente se encuentran en otros tipos de software. El hecho de contemplar y tener que respetar restricciones temporales, la vuelve una disciplina sensible, donde es necesario que a lo largo de todo el proceso de desarrollo y desde diferentes puntos de vista se tenga en cuenta la predecibilidad del sistema. Para lograr este objetivo, se tomaron dos políticas de planificación probadas en forma teórica y se aplicaron sus conceptos a un desarrollo práctico genérico. En este contexto, el presente capítulo realizó los siguientes aportes específicos:

- Se desarrolló un modelo basado en contratos, de las diferentes entidades involucradas en el sistema, que brinda una visión general sobre las relaciones entre dichas entidades.
- Se desarrolló una arquitectura de software, que describe las relaciones y características de las entidades en forma estática.
- Se desarrolló un modelo dinámico de cada una de las principales entidades involucradas, que brinda las bases para una descripción del comportamiento, en tiempo de ejecución, de las mismas.

Capítulo 5

Conclusiones y Trabajos Futuros

El estudio y análisis de la literatura especializada y los desarrollos realizados hasta el momento en el área de los sistemas de tiempo real, muestran ciertos tópicos abiertos que deben tratarse a fin de consolidar esta disciplina y poder tomar ventaja de los aportes teóricos en la práctica. La investigación en estos sistemas ha tenido un avance muy grande en los últimos años, que no se ve reflejado en las implementaciones comerciales o industriales. La puesta al día en el estado del arte lleva a distinguir, entre otros, tres factores determinantes como causas de esta problemática. En primer lugar, dicho avance se ha dado principalmente en el área de planificación de estos sistemas, donde se han desarrollado una enorme cantidad de políticas. Sin embargo, en la mayoría de los casos estas políticas se centran en los aspectos no funcionales del sistema para tomar las decisiones, dejando de lado cuestiones funcionales como el resultado de la computación de una tarea, la función que realiza o el aporte que da en beneficio del sistema. En segundo lugar, la naturaleza crítica de los sistemas de tiempo real genera la necesidad de tomar recaudos extra para proteger a las entidades del sistema de malos funcionamientos. En este sentido, la utilización de mecanismos de reserva basados en servidores y de métodos que permitan compartir recursos en forma segura y simple, se vuelve un punto clave de cualquier enfoque. En particular, para el tratamiento de recursos compartidos, debido a que es un tema muy sensible para la planificación, es además necesario

que el método elegido pueda acompañar todo el proceso de desarrollo, a fin de evitar situaciones indeseadas cuando ya se tiene un grado de avance considerable. Finalmente, en tercer lugar, resulta indispensable contar con herramientas que faciliten la implementación de los conceptos teóricos en diseños y desarrollos reales. De este modo, al momento de aplicar una determinada política de planificación o un conjunto de métodos teóricos, se dispone de un marco de software básico antes de comenzar el trabajo. Este marco de software resulta beneficioso para el desarrollador, ya que le sirve de guía a lo largo de todo el proceso de desarrollo y como puente entre los conceptos teóricos y las limitaciones propias de la práctica.

En base a lo anterior, esta tesis alcanza los siguientes objetivos generales:

- ♢ Desarrolla una política de planificación que tiene en cuenta no sólo las características temporales de las entidades del sistema, sino también los aspectos funcionales y el desempeño semántico de sus acciones en tiempo de ejecución (*i.e.*, el comportamiento de las tareas).

- ♢ Desarrolla una política de manejo de recursos compartidos en tiempo real eficiente y que puede utilizarse desde etapas tempranas del proceso de desarrollo.

- ♢ Construye un marco de software que sirve para unificar los conceptos teóricos con los prácticos y de implementación y que, a la vez, brinda las bases suficientes para una correcta y rápida adaptación a los casos particulares de las diferentes aplicaciones.

En particular, en el Capítulo 2, se presenta la Política de Planificación Comportamental, la cual utiliza la ponderación del comportamiento de las tareas para determinar su prioridad. Para esto, se desarrolló un nuevo modelo de tareas de tiempo real, se construyó un modelo de servidor que toma en cuenta el comportamiento de la tarea y varía su frecuencia de ejecución en función del mismo y se desarrolló y probó formalmente una nueva política, que es quien rige el ciclo de vida de las tareas y los servidores en tiempo de ejecución.

Por otro lado, en el Capítulo 3, se propone una extensión a una política clásica de manejo de recursos compartidos. Dicha extensión se denomina ESRP y consiste fundamentalmente

en la inclusión de los conceptos de conjunto de tareas y de recursos, mediante la definición de relaciones de bloqueo. Además de las pruebas y demostraciones formales, se expusieron consideraciones de implementación a tener en cuenta al momento de aplicar ESRP a un sistema concreto basado en esta nueva formulación. Asimismo, se mostró la flexibilidad de la política para poder implementarse junto con esquemas de prioridades fijas, prioridades dinámicas y con la Política de Planificación Comportamental.

Finalmente, en el Capítulo 4, se toman dos políticas de planificación probadas en forma teórica (Política de Planificación Comportamental y ESRP) y se aplican sus conceptos a un desarrollo práctico genérico. El marco de software presentado, denominado BIPS-SF, beneficia al desarrollador proveyéndole de diferentes puntos de vista para diseñar una aplicación. Esto fortalece los conceptos propios de la ingeniería de software y refuerza las herramientas para construir un sistema predecible. El marco de software propuesto se compone de un modelo basado en contratos, donde se da una visión global de las interacciones entre las diferentes entidades involucradas en el sistema; de un modelo estructural, que proporciona la configuración estática de las entidades en términos de diagramas de clase anotados con el perfil MARTE; y de un modelo dinámico, que describe, mediante diagramas de estado, el comportamiento básico de cada una de las entidades del sistema y su interacción en tiempo de ejecución.

Los aportes realizados en esta tesis sirven como base para futuras investigaciones en las líneas de trabajo que siguen naturalmente a las establecidas aquí: ampliar la definición de la Función de Comportamiento δ, para contemplar mayor variabilidad de las computaciones trascendentales; extender los conceptos de los servidores BIPS, para encapsular varias tareas, con diferentes funciones δ, en un mismo servidor; introducir mecanismos de reclamo de ancho de banda disponible, en los casos de servidores BIPS que sufrieron una postergación; y ampliar el marco de software BIPS-SF, a fin de contemplar en el modelado recursos de hardware.

Bibliografía

[1] Framework for real-time embedded systems based on contracts, `http://www.frescor.org` Last visited: 3/2010.

[2] Modeling and analysis of real-time and embedded systems, `http://www.omgmarte.org` Last visited: 3/2010.

[3] Object management group, `http://www.omg.org` Last visited: 3/2010.

[4] Unified modeling language, `http://www.uml.org` Last visited: 3/2010.

[5] Crash warning system interfaces: Human factors insights and lessons learned, Tech. report, United States Department of Transport, 2007.

[6] L. Abeni and G. Buttazzo, Integrating multimedia applications in hard real-time systems, RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 1998, p. 4.

[7] Luca Abeni, Luigi Palopoli, Giuseppe Lipari, and Jonathan Walpole, Analysis of a reservation-based feedback scheduler, RTSS '02: Proceedings of the 23rd IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2002, p. 71.

[8] S.A. Aldarmi and A. Burns, Dynamic value-density for scheduling real-time systems, Real-Time Systems, 1999. Proceedings of the 11th Euromicro Conference on, 1999, pp. 270–277.

[9] M. Aldea, G. Bernat, I. Broster, A. Burns, R. Dobrin, J. M. Drake, G. Fohler, P. Gai, M. Gonzalez Harbour, G. Guidi, J. J. Gutierrez, T. Lennvall, G. Lipari, J. M. Martinez, J. L. Medina, J. C. Palencia, and M. Trimarchi, Fsf: A real-time scheduling architecture framework, RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium (Washington, DC, USA), IEEE Computer Society, 2006, pp. 113–124.

[10] M. Anand, A. Easwaran, S. Fischmeister, and Insup Lee, Compositional feasibility analysis of conditional real-time task models, Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on, May 2008, pp. 391–398.

[11] Hakan Aydin, Rami Melhem, Daniel Mosseé, and Pedro Mejía-Alvarez, Optimal reward-based scheduling for periodic real-time tasks, IEEE Trans. Comput. **50** (2001), no. 2, 111–130.

[12] T. P. Baker, Stack-based scheduling for realtime processes, Real-Time Syst. **3** (1991), no. 1, 67–99.

[13] Scott Banachowski, Timothy Bisson, and Scott A. Brandt, Integrating best-effort scheduling into a real-time system, RTSS '04: Proceedings of the 25th IEEE International Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2004, pp. 139–150.

[14] S. K. Baruah, A general model for recurring real-time tasks, RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 1998, p. 114.

[15] Sanjoy K. Baruah, Resource sharing in edf-scheduled systems: A closer look, RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2006, pp. 379–387.

[16] Sanjoy K. Baruah, Louis E. Rosier, and R. R. Howell, Algorithms and complexity concerning the preemptive scheduling of periodic, real-time tasks on one processor, Real-Time Syst. **2** (1990), no. 4, 301–324.

[17] A. Bechini, T.M. Conte, and C.A. Prete, Guest editors' introduction: Opportunities and challenges in embedded systems, Micro, IEEE **24** (2004), no. 4, 8 – 9.

[18] Moris Behnam, Insik Shin, Thomas Nolte, and Mikael Nolin, Sirap: a synchronization protocol for hierarchical resource sharingin real-time open systems, EMSOFT '07: Proceedings of the 7th ACM & IEEE international conference on Embedded software (New York, NY, USA), ACM, 2007, pp. 279–288.

[19] Richard Bishop, Intelligent vehicle technology and trends, Artech House, Norwood, MA, 2005.

[20] Scott A. Brandt and Gary J. Nutt, Flexible soft real-time processing in middleware, Real-Time Syst. **22** (2002), no. 1/2, 77–118.

[21] A. Burns, D. Prasad, A. Bondavalli, F. Di Giandomenico, K. Ramamritham, J. Stankovic, and L. Strigini, The meaning and role of value in scheduling flexible real-time systems, J. Syst. Archit. **46** (2000), no. 4, 305–325.

[22] G. Buttazzo, M. Spuri, and F. Sensini, Value vs. deadline scheduling in overload conditions, RTSS '95: Proceedings of the 16th IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 1995, p. 90.

[23] G. C. Buttazzo, G. Lipari, and L. Abeni, Elastic task model for adaptive rate control, RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 1998, p. 286.

[24] Giorgio C. Buttazzo, Hard real-time computing systems: Predictable scheduling algorithms and applications, Kluwer Academic Publishers, Norwell, MA, USA, 1997.

[25] Giorgio C. Buttazzo, Giuseppe Lipari, Marco Caccamo, and Luca Abeni, Elastic scheduling for flexible workload management, IEEE Trans. Comput. **51** (2002), no. 3, 289–302.

[26] Marco Caccamo, Giorgio Buttazzo, and Lui Sha, Elastic feedback control, ECRTS '00: Proceedings of the Euromicro Conference on Real-Time Systems (Los Alamitos, CA, USA), IEEE Computer Society, 2000, p. 121.

[27] Marco Caccamo, Giorgio C. Buttazzo, and Deepu C. Thomas, Efficient reclaiming in reservation-based real-time systems with variable execution times, IEEE Trans. Comput. **54** (2005), no. 2, 198–213.

[28] Marco Caccamo and Lui Sha, Aperiodic servers with resource constraints, RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2001, p. 161.

[29] Goverment Canadian, Canadian forest fire danger rating system.

[30] Jan Carlson, Tomas Lennvall, and Gerhard Fohler, Enhancing time triggered scheduling with value based overload handling and task migration, ISORC '03: Proceedings of the Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (Washington, DC, USA), IEEE Computer Society, 2003, p. 121.

[31] Anton Cervin and Johan Eker, The control server: A computational model for real-time control tasks, Proceedings of the 15th Euromicro Conference on Real-Time Systems (ECRTS'03) (Los Alamitos, CA, USA), IEEE Computer Society, 2003, p. 113.

[32] Hui Chen and Jiali Xia, A real-time task scheduling algorithm based on dynamic priority, ICESS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems (Washington, DC, USA), IEEE Computer Society, 2009, pp. 431–436.

[33] Min-Ih Chen and Kwei-Jay Lin, Dynamic priority ceilings: a concurrency control protocol for real-time systems, Real-Time Syst. **2** (1990), no. 4, 325–346.

[34] Yu Chen and Qionghai Dai, Research on dynamic feedback and elastic scheduling model and algorithm for flexible workload, ICCNMC '03: Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing (Washington, DC, USA), IEEE Computer Society, 2003, p. 283.

[35] Houssine Chetto and Maryline Chetto, Some results of the earliest deadline scheduling algorithm, IEEE Trans. Softw. Eng. **15** (1989), no. 10, 1261–1269.

[36] Jen-Yao Chung, Jane W. S. Liu, and Kwei-Jay Lin, Scheduling periodic jobs that allow imprecise results, IEEE Trans. Comput. **39** (1990), no. 9, 1156–1174.

[37] T.M. Conte, Choosing the brain(s) of an embedded system, Computer **35** (2002), no. 7, 106 –107.

[38] Luis Alejandro Cortés, Petru Eles, and Zebo Peng, Static scheduling of monoprocessor real-time systems composed of hard and soft tasks, DELTA '04: Proceedings of the Second IEEE International Workshop on Electronic Design, Test and Applications (Washington, DC, USA), IEEE Computer Society, 2004, p. 115.

[39] T. Cucinotta, L. Palopoli, L. Marzario, G. Lipari, and L. Abeni, Adaptive reservations in a linux environment, RTAS '04: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (Washington, DC, USA), IEEE Computer Society, 2004, p. 238.

[40] R. Davis, S. Punnekkat, N. Audsley, and A. Burns, Flexible scheduling for adaptable real-time systems, RTAS '95: Proceedings of the Real-Time Technology and Applications Symposium (Washington, DC, USA), IEEE Computer Society, 1995, p. 230.

[41] R. I. Davis and A. Burns, Resource sharing in hierarchical fixed priority pre-emptive systems, RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2006, pp. 257–270.

[42] Sébastien Demathieu, Frédéric Thomas, Charles André, Sébastien Gérard, and François Terrier, First experiments using the uml profile for marte, ISORC '08: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing (Washington, DC, USA), IEEE Computer Society, 2008, pp. 50–57.

[43] Z. Deng and J. W.-S. Liu, Scheduling real-time applications in an open environment, RTSS '97: Proceedings of the 18th IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 1997, p. 308.

[44] María C. Dentoni, Miriam M. Muñoz, and Fernando Epele, Implementación de un sistema nacional de evaluación de peligro de incendios: la experiencia argentina, Proceedings of 4th Internation Wildland Fire Conference (Sevilla, España), 2007.

[45] Michael L. Dertouzos, Control robotics: The procedural control of physical processes, IFIP Congress, 1974, pp. 807–813.

[46] Peter Dibble, Deadline scheduling, Embedded Systems Design (2001), `http://www.embedded.com/story/OEG20010304S0004` Last visited: 3/2010.

[47] H. Espinoza, J. Medina, F. Dubois, H. Terrier, and S. Gerard, Towards a uml-based modeling standard for scedulability analisys of real-time systems, nternational Workshop on Modeling and Analysis of Real-Time Embedded Systems at MODELS'06 (Genova (Italy)), October 2006.

[48] Mohamed Fayad and Douglas C. Schmidt, Special issue on object-oriented application frameworks, Communications of the ACM **40** (1997), no. 10.

[49] Paolo Gai, Giuseppe Lipari, and Marco Di Natale, Minimizing memory utilization of real-time task sets in single and multi-processor systems-on-a-chip, RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2001, p. 73.

[50] Paolo Gai, Marco Di Natale, Giuseppe Lipari, Alberto Ferrari, Claudio Gabellini, and Paolo Marceca, A comparison of mpcp and msrp when sharing resources in the janus multiple-processor on a chip platform, RTAS '03: Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium (Washington, DC, USA), IEEE Computer Society, 2003, p. 189.

[51] David Garlan, Software architecture: a roadmap, ICSE '00: Proceedings of the Conference on The Future of Software Engineering (New York, NY, USA), ACM, 2000, pp. 91–101.

[52] T. M. Ghazalie and T. P. Baker, Aperiodic servers in a deadline scheduling environment, Real-Time Syst. **9** (1995), no. 1, 31–67.

[53] M. Hefeeda and M. Bagheri, Wireless sensor networks for early detection of forest fires, Proceedings of the 4th IEEE International Conference on Mobile Adhoc and Sensor Systems (Pisa, Italy), 2007.

[54] Thomas A. Henzinger and Joseph Sifakis, The discipline of embedded systems design, IEEE Computer **40** (2007), no. 10, 32–40.

[55] Michael Jackson, Software requirements and specifications: A lexicon of practice, principles and prejudices, ACM Press, 1995.

[56] Kevin Jeffay, Scheduling sporadic tasks with shared resources in hard-real-time systems, Proc. 13 th IEEE Real-Time Systems Symp, 1992, pp. 89–99.

[57] Hai Jin, Qionghua Hu, Xiaofei Liao, Hao Chen, and Dafu Deng, Imac: an importance-level based adaptive cpu scheduling scheme for multimedia and non-real time applications, AICCSA '05: Proceedings of the ACS/IEEE 2005 International Conference on Computer Systems and Applications (Washington, DC, USA), IEEE Computer Society, 2005, pp. 119–vii.

[58] Ralph E. Johnson, Components, frameworks, patterns, SSR '97: Proceedings of the 1997 symposium on Software reusability (New York, NY, USA), ACM, 1997, pp. 10–17.

[59] W.D. Jones, Keeping cars from crashing, IEEE Spectrum **38** (2001), no. 9, 40–45.

[60] V. Kalogeraki, P. M. Melliar-Smith, and L. E. Moser, Dynamic scheduling for soft real-time distributed object systems, ISORC '00: Proceedings of the Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (Washington, DC, USA), IEEE Computer Society, 2000, p. 114.

[61] Naoko Kosugi, Atsushi Mitsuzawa, and Mario Tokoro, Importance-based scheduling for predictable real-time systems using mart, WPDRTS '96: Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems (Washington, DC, USA), IEEE Computer Society, 1996, p. 95.

[62] Gerardo Lamastra, Giuseppe Lipari, and Luca Abeni, A bandwidth inheritance algorithm for real-time task synchronization in open systems, RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS'01) (Washington, DC, USA), IEEE Computer Society, 2001.

[63] Phillip A. Laplante, Real-time systems design and analysis: An engineer's handbook, Wiley-IEEE Press, 1996.

[64] Peng Li and Binoy Ravindran, Fast, best-effort real-time scheduling algorithms, IEEE Trans. Comput. **53** (2004), no. 9, 1159–1175.

[65] C. L. Liu and James W. Layland, Scheduling algorithms for multiprogramming in a hard-real-time environment, J. ACM **20** (1973), no. 1, 46–61.

[66] Carey Douglass Locke, Best-effort decision-making for real-time scheduling, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1986.

[67] Marcus Eduardo Markiewicz and Carlos J. P. de Lucena, Object oriented framework development, Crossroads **7** (2001), no. 4, 3–9.

[68] Peter Marwedel, Embedded system design, Springer, 2003.

[69] Luca Marzario, Giuseppe Lipari, Patricia Balbastre, and Alfons Crespo, Iris: A new reclaiming algorithm for server-based real-time systems, Proceedings of the 10th IEEE RTAS (Toronto, Canada), IEEE Computer Society, 2004.

[70] Michael Mattson, Object-oriented frameworks, Ph.D. thesis, University College of Karlskrona/Ronneby, 1996.

[71] Bertrand Meyer, Applying "design by contract", Computer **25** (1992), no. 10, 40–51.

[72] A. K. Mok, Fundamental design problems of distributed systems for the hard-real-time environment, Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 1983.

[73] Dionisio de Niz, Luca Abeni, Saowanee Saewong, and Ragunathan (Raj) Rajkumar, Resource sharing in reservation-based systems, RTSS '01: Proceedings of the 22nd IEEE Real-Time Systems Symposium (Washington, DC, USA), IEEE Computer Society, 2001, p. 171.

[74] T. Nolte, M. Nolin, and H.A. Hansson, Real-time server-based communication with can, IEEE Transactions on Industrial Informatics **1** (2005), no. 3, 192–201.

[75] Gary J. Nutt, Centralized and distributed operating systems, Prentice-Hall International Editions, 1992.

[76] Leo Ordinez and David Donari, Pautas de análisis y diseño de sistemas de procesamiento de señales y datos con restricciones temporales, Anales JAIIO 2006, Concurso de Trabajos Estudiantes, EST 2006, Sociedad Argentina de Informática, 2006.

[77] Alessandro Pasetti, Software frameworks and embedded control systems, Lecture Notes in Computer Science, vol. 2231/2002, Springer Berlin / Heidelberg, 2002.

[78] Raj Rajkumar, Kanaka Juvva, Anastasio Molano, and Shui Oikawa, Resource kernels: A resource-centric approach to real-time and multimedia systems, Proceedings of the SPIE/ACM Conference on Multimedia Computing and Networking, january 1998.

[79] Glenn Reeves, What really happened on mars?, `http://research.microsoft.com/en-us/um/people/mbj/mars_pathfinder/Authoritative_Account.html` Last visited: 3/2010.

[80] Yagíl Ronén, Daniel Mossé, and Martha E. Pollack, Value-density algorithms for the deliberation-scheduling problem, SIGART Bull. **7** (1996), no. 2, 41–49.

[81] R. Santos, M. B. D'Amico, J. Orozco, P. Doñate, L. Ordinez, and D. Donari, Power and realtime requirements integrated with an adaptive resource reservation soft real-time scheduler, Proc. XXXI Conferencia Latinoamericana de Informática (CLEI'05) (Santiago de Cali, Colombia), 2005, pp. 833 – 843.

[82] R. Santos, P. Doñate, M. B. D'Amico, L. Ordinez, and D. Donari, Adaptive resource reservation scheduler, Proc. XII Reunión de Trabajo en Procesamiento de la Información y Control (RPIC'07) (Río Gallegos, Argentina), 2007.

[83] R.M. Santos, G. Lipari, and J. Santos, Improving the schedulability of soft real-time open dynamic systems: The inheritor is actually a debtor, Journal of Systems and Software **81** (2008), no. 7, 1093–1104.

[84] Bran Selic, A generic framework for modeling resources with uml, IEEE Computer **33** (2000), no. 6, 64–69.

[85] X. Serra, G. Widmer, and M. Leman, A roadmap for sound and music computing, The S2S Consortium, 2007, `http://mtg.upf.edu/files/publications/142fbf-SMC-Roadmap-2007.pdf`.

[86] L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, Tech. report, Dep. Comput. Sci. - CMU, 1987.

[87] L. Sha, R. Rajkumar, and J. P. Lehoczky, Priority inheritance protocols: An approach to real-time synchronization, IEEE Trans. Comput. **39** (1990), no. 9, 1175–1185.

[88] Lui Sha, Tarek Abdelzaher, Karl-Erik Årzén, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysius K. Mok, Real time scheduling theory: A historical perspective, Real-Time Syst. **28** (2004), no. 2-3, 101–155.

[89] Maryline Silly, The edl server for scheduling periodic and soft aperiodictasks with resource constraints, Real-Time Syst. **17** (1999), no. 1, 87–111.

[90] Brinkley Sprunt, Aperiodic task scheduling for real-time systems, Ph.D. thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1990.

[91] Marco Spuri and Giorgio C. Buttazzo, Efficient aperiodic service under earliest deadline scheduling, Proc. IEEE Real-Time Systems Symp. (San Juan, Puerto Rico), IEEE Computer Society, 1994, pp. 2–11.

[92] William Stallings, Operating systems: Internals and design principles, 4th ed., Ed. Prentice-Hall, 2001.

[93] John A. Stankovic, Misconceptions about real-time computing: A serious problem for next-generation systems, Computer **21** (1988), no. 10, 10–19.

[94] John A. Stankovic and Krithi Ramamritham, What is predictability for real-time systems?, Real-Time Syst. **2** (1990), no. 4, 247–254.

[95] Andrew S. Tanenbaum, Modern operating systems, Prentice-Hall, 1992.

[96] Stephen W. Taylor, Considerations for applying the canadian forest fire danger rating system in argentina, Tech. report, Canadian Forest Service - Pacific Forestry Centre - Natural Resources Canada, September 3 2001.

[97] Lonnie R. Welch and Scott Brandt, Toward a realization of the value of benefit in real-time systems, IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium (Washington, DC, USA), IEEE Computer Society, 2001, p. 93.

[98] Wayne Wolf, What is embedded computing?, Computer **35** (2002), 136–137.

[99] Wayne Wolf, Guest editor's introduction: The embedded systems landscape, IEEE Computer **40** (2007), no. 10, 29–31.

[100] Chen Zhang and David Cordes, Resource access control for dynamic priority distributed real-time systems, Real-Time Syst. **34** (2006), no. 2, 101–127.