UNIVERSIDAD POLITÉNICA DE VALENCIA

Gestión de la carga dinámica de tareas de tiempo real con criterios de ahorro energético y su aplicación en el desarrollo de un middleware de control.

TESIS DOCTORAL PRESENTADA POR:

Javier O. Coronel Parada

DIRIGIDA POR:

José Enrique Simó Ten

Valencia, Febrero de 2016

A mi hermana,
de quien aún me queda mucho por aprender.

Agradecimientos

Agradezco a todos los profesores y compañeros que de una u otra forma han colaborado en el desarrollo de esta tesis. Y en especial, a mi director José Simó cuya orientación, apoyo y sus ánimos han sido fundamentales para la realización de este trabajo.

Igualmente agradezco a los miembros del Grupo de Informática Industrial del DISCA, con especial cariño a Alfons, Paco y Pascual, por su colaboración y acogimiento durante mi formación y desarrollo de esta investigación.

A mis padres, Osvaldo e Idda, a Laura, Lucas y Noa por toda la paciencia, comprensión y motivación inyectada para la culminación de este proyecto.

Resum

El desenvolupament de sistemes de còmput en sectors industrials com són el ferroviari, aeroespacial i automòbil està basat en Sistemes Embeguts Crítics de Temps Real (també coneguts per les sigles en anglès CRTES). Aquests sistemes sénfronten a noves demandes i exigències relacionades amb líncrement de la fiabilitat, major intel·ligència, connectivitat, reducció del volum, millores de rendiment i eficiència en el consum energètic. És en aquest últim aspecte on aquesta tesi doctoral espera fer la seva principal aportació.

El criteri de consum energètic, combinat amb altres criteris com ara planificabilitat, retards de comunicació i estabilitat en aplicacions de control, contribueixen a la determinació del moviment de codi i al balanç de càrregues en sistemes distribuïts. Lóbjectiu principal dáquesta tesi és el desenvolupament de mecanismes de gestió i optimització del consum energètic. Aquests mecanismes es presenten com a possibles funcionalitats en el marc del disseny de middlewares basats en el concepte de nucli de control. El desenvolupament dáquesta tesi considera un entorn dinàmic on sistemes CRTES basats en suport middleware i connectats a una xarxa de comunicacions, permeten dur a terme migracions de tasques i modificacions de la freqüència del processador en temps déxecució. Suposant que el sistema distribuït coneix on i quan assignar les tasques entre les unitats de còmput, cal fer un anàlisi de factibilitat de la planificació en cada arribada i sortida de tasques sobre els sistemes embeguts afectats. Dáquesta manera, es pot garantir que els requisits temporals del sistema seran complerts durant la fase de re-assignació o distribució de tasques. Això també implica que una velocitat de processador nova (escalament de freqüència) ha de ser calculada per permetre una optimització energètica i ládaptació del sistema a les noves condicions de càrrega computacional. I és en aquest punt en el que els algoritmes proposats en aquesta tesi tenen la seva importància. Encara que alguns autors han dut a terme aquestes dues fases (l'anàlisi de planificabilitat i el càlcul de léscalat de freqüència) separadament, aquestes anàlisis estan fortament relacionats i en alguns casos poden ser executats de forma conjunta.

En aquest treball de tesi es proposa un algoritme nou per a lánàlisi de factibilitat de planificació i el càlcul de freqüències estàtiques de processador basat en tècniques déscalament de freqüència i voltatge dinàmic (també conegut com DVFS). La freqüència obtinguda per aquest algoritme és la freqüència mínima que garanteix que si es fa servir de forma invariable en el processador, séstalviarà la major energia possible i a més a més es compliran tots els terminis déxecució de les tasques del sistema. Aquest algoritme utilitza un esquema de planificació per prioritats fixes amb terminis déxecució menors i/o iguals que el període de les tasques. Un dels propòsits dáquest algoritme és el seu ús durant léxecució del sistema, que ha de permetre gestionar adaptacions de càrrega computacional i energètica del processador.

Lálgoritme de càlcul de freqüències estàtiques de processador, és complementat en aquesta tesi amb la proposta de mètodes nous dóptimització dinàmica, que ajusten el consum energètic basat en les condicions de càrregues de còmputs reals en cada instant. Aquests mètodes proposats sútilitzen en temps déxecució de les tasques del sistema i es basen en lássignació de freqüències dinàmiques al processador. Aquestes noves freqüències utilitzen com a referència el càlcul previ de la freqüència estàtica. Els canvis de freqüència dinàmics se succeeixen com a resposta a instants ociosos de processador deguts principalment a terminacions anticipades de tasques.

Per a lávaluació dáquesta tesi es proposen un conjunt de simulacions i experiments que permeten comparar i valorar les contribucions dáquesta tesi respecte a altres algoritmes existents a la literatura.

Resumen

El desarrollo de sistemas de cómputo en sectores industriales tales como el ferroviario, aeroespacial y automóvil está basado en Sistemas Empotrados Críticos de Tiempo Real (también conocidos por las siglas CRTES). Estos sistemas se enfrentan a nuevas demandas y exigencias relacionadas con el incremento de la fiabilidad, mayor inteligencia, conectividad, reducción del volumen, mejoras del rendimiento y eficiencia en el consumo energético. Y es, en ese último aspecto, donde esta tesis doctoral espera hacer su principal aportación.

El criterio de consumo energético, combinado con otros criterios tales como planificabilidad, retardos de comunicación y estabilidad en aplicaciones de control, contribuyen a la determinación del movimiento de código y al balance de cargas en sistemas distribuidos. El objetivo principal de esta tesis es el desarrollo de mecanismos para la gestión y optimización del consumo energético. Estos mecanismos se presentan como posibles funcionalidades en el marco del diseño de middlewares basados en el concepto de núcleo de control. El desarrollo de esta tesis considera un entorno dinámico donde sistemas CRTES, basados en soportes middleware y conectados a una red de comunicaciones, permiten llevar a cabo migraciones de tareas y modificaciones de la frecuencia del procesador en tiempo de ejecución. Suponiendo que el sistema distribuido conoce dónde y cuándo asignar las tareas entre las unidades de cómputo, es necesario realizar un análisis de factibilidad de la planificación en cada llegada y partida de tareas sobre los sistemas empotrados afectados. De esta forma, se garantiza que los requisitos temporales del sistema serán cumplidos durante la fase de re-asignación o distribución de tareas. Esto también implica que una nueva velocidad de procesador (escalamiento de frecuencia) deberá ser calculada para permitir una optimización energética y la adaptación del sistema a las nuevas condiciones de carga computacional. Y es en este punto en el que los algoritmos propuestos en esta tesis tiene su importancia. Aunque algunos autores han llevado a cabo estas dos fases (análisis de planificabilidad y cálculo del escalado de frecuencia) separadamente, estos análisis están fuertemente relacionados y en algunos casos pueden ser ejecutados de forma conjunta.

En este trabajo de tesis se propone un algoritmo nuevo para el análisis de planificabilidad y el cálculo de frecuencias estáticas de procesamiento basado en técnicas de escalamiento de frecuencia y voltaje dinámico (también conocido como DVFS). La frecuencia obtenida por este algoritmo es la frecuencia mínima que garantiza que si se usa de forma invariable en el procesador, se ahorrará la mayor energía posible y además se cumplirán todos los plazos de ejecución de las tareas del sistema. Este algoritmo utiliza un esquema de planificación por prioridades fijas con plazos de ejecución menor y/o igual que el periodo de las tareas. Uno de los propósitos de este algoritmo es su uso durante la ejecución del sistema, que permita gestionar adaptaciones de carga computacional y energética del procesador.

El algoritmo para el cálculo de frecuencias estáticas de procesador, es complementado en esta tesis con la propuesta de métodos nuevos de optimización dinámica, que ajustan el consumo energético basado en las condiciones de carga de computo reales en cada instante. Estos métodos propuestos se utilizan en tiempo de ejecución de las tareas del sistema y se basan en la asignación de frecuencias dinámicas al procesador. Estas nuevas frecuencias utilizan como referencia el cálculo previo de la frecuencia estática. Los cambios de frecuencia dinámicos se suceden como respuesta a instantes ociosos de procesador debidos principalmente a terminaciones anticipadas de tareas.

Para la evaluación de esta tesis se proponen un conjunto de simulaciones y experimentos que permiten comparar y valorar las contribuciones de esta tesis con respecto a otros algoritmos existentes en la literatura.

Abstract

The development of embedded systems in industrial sectors such as railway, aerospace and automotive are based on Critical Real-Time Embedded Systems (CRTES). These systems face new challenges and demand related to increase of dependability, intelligence, connectivity, cost-size-volume reduction and energy efficiency. In this last topic is where this thesis expects to have a higher contribution.

The global energy consumption can be combined with others criteria such as schedulability, communication delays and control application correctness, which contribute to determine the dynamic code movement and on-line load balancing in a system. The main goal of this thesis is the development of mechanisms for the management and optimization of energy consumption. These mechanisms are presented in the context of a distributed real-time control system and from the perspective of control kernel middleware. Let's consider a dynamic environment where an embedded and networked system operates with the support of task migration and processor frequency scaling. Assuming that the system knows where and when it must allocate tasks, we must perform feasibility analyses when each task arrives and departs on the affected embedded units. This guarantees that the temporal requirements of the system will be accomplished during the task allocation phase or delegation of tasks. Additionally, a new processor speed (frequency scaling) should be also computed to enable the system to adapt itself to the new computational workload and reduce energy consumption. And in this last is where the proposed algorithms in this work have their higher relevance. Although some authors have carried out these two phases (feasibility analysis and frequency scaling computation) separately, these analyses are strongly related and in some cases can be performed together.

In this thesis, we present novel algorithm that perform feasibility analyses and compute new processor static frequencies based on dynamic voltage and frequency scaling techniques (DVFS). The frequency obtained as result of applying the algorithm proposed is the minimum processor frequency that minimizes CPU energy consumption while guaranteeing the fulfilment of real-time system constraints. The algorithm uses fixed priority scheduling schemes with deadlines less than, or equal to, the period of the tasks. Other propose of this algorithm is the use on-line during the task allocation and processor speed assignment phases.

In this work, the computation of the minimum static processor frequency is accompanied with the proposed the additional approaches for the dynamic optimization of the energy consumption. Dynamic algorithms are based on the reclamation of additional slack resulting from the early completions of tasks. These are then used to further reduce the processor frequency and save more energy. These algorithms are applied at run-time. The computation of these additional dynamic processor frequencies uses as reference the previous calculation of the minimum static processor frequency.

Through extensive simulations, we evaluate the performance of this algorithm against other existing feasibility tests that have been adapted to compute the minimum processor frequency. This minimum frequency is computed in terms of energy consumption, acceptability ratio, and real computing costs. In addition, predictability in the execution and behaviour of the algorithms in relation to the continuing arrival of tasks is analyzed.

Índice general

Índice de figuras

Índice de tablas

INTRODUCCIÓN Y MOTIVACIÓN

Los Sistemas Ciber-Físicos (CPS-Cyber-Physical Systems) suponen una interacción dinámica y compleja entre el mundo real y los sistemas de cómputo en tiempo real (Khaitan & McCalley, 2015; Wolf, 2009). En este escenario, los sistemas de cómputo están comúnmente formados por varias unidades empotradas interconectadas mediante diversos mecanismos de comunicación. Uno de los desafíos que enfrentan los sistemas futuros es gestionar de forma eficiente la separación del diseño de control modal del desarrollo del código ejecutable. En tales sistemas, esta es una tarea difícil, si no imposible, ya que habría que analizar fuera de línea todas las posibles combinaciones de las cargas del procesador derivadas de todos los posibles modos de funcionamiento. Debido a la gran variedad de dominios de procesos de control, es difícil abordar la modalidad de control únicamente mediante la parametrización de los controladores locales precargados. Por esta razón, se investiga sobre la definición de nuevas arquitecturas flexibles que permitan a los sistemas informáticos adaptarse de forma eficiente (buscando el aprovechamiento óptimo de los recursos) a los cambios potenciales en el entorno de una manera dinámica.

Desde un punto de vista de los sistemas empotrados de control, es bien sabido que la integración del diseño del controlador y la planificación de tiempo real es necesaria (Al-Areqi et al., 2015; Xu et al., 2015; Xia & Sun, 2008). Sin embargo, esta propuesta no es suficiente en entornos dinámicos con recursos computacionales limitados(Xia & xian Sun, 2008). Por tanto, es necesaria la inclusión de mecanismos adicionales en el diseño para la optimización del uso de recursos y proporcionar una adaptación a las condiciones cambiantes del entorno (Khaitan & McCalley, 2015; Zhang et al., 2008). Del mismo modo, se tendrán que garantizar niveles específicos de calidad de servicio (QoS), de tal modo que se asegure la operación correcta del sistema. Estos niveles de QoS asegurarán un rendimiento de control y una predecibilidad temporal apropiada en el sistema de control.

En este contexto, cambios en el modo de operación del sistema, o en el entorno, pueden forzar la inhibición de algunos controladores y la activación de otros. Además, deberán existir mecanismos de conmutación de controladores que permitan agregar o quitar tareas de control tanto de forma local como distribuida, resultando en cambios de cargas de trabajo en las unidades empotradas e interconectadas (Biondi & Buttazzo, 2015; Hang & Hansson, 2011; Emberson & Bate, 2007). Esto puede tener un impacto significativo en el funcionamiento y en las restricciones temporales de todo el sistema. En estos casos, los métodos de migración de componentes y tareas (Santos et al., 2013; Briao et al., 2007) pueden contribuir a una redistribución y reasignación de cargas de trabajo en tiempo de ejecución en función de cada situación.

Un ejemplo de esta clase de arquitectura es la propuesta por el autor en diferentes publicaciones (Simarro et al., 2008; Coronel et al., 2008). Dicha arquitectura se desarrolla en el contexto de sistemas de control de tiempo real distribuidos y desde la perspectiva de middleware de núcleo de control (Sánchez et al., 2014; Crespo et al., 2006; Albertos et al., 2006). La propuesta incluye características tales como fundamentos de control, conmutación de controladores y delegación de código. Desde el punto de vista de CPS estos métodos resuelven parcialmente este tipo de abstracciones (Troger et al., 2015; Lee, 2006).

En resumen, los Sistemas Ciber-Físicos deberían ser capaces de proporcionar nuevos niveles de desempeño y eficiencia, gracias a sofisticados co-diseños control/cómputo. Igualmente, deberíamos cambiar nuestro entendimiento de los computadores y entender que los CPS siempre están en constante contacto con el mundo real, con restricciones de tiempo real y consumiendo energía real.

En este trabajo de tesis se abordan métodos que pueden ser utilizados en la migración de tareas combinando criterios de aceptación de tareas y planificabilidad con técnicas de gestión de energía.

Estos métodos pueden maximizar el ahorro global de energía del sistema, facilitando por ejemplo, la gestión térmica de chips (Zhou & Wei, 2015) mediante el movimiento de tareas desde unidades de procesamiento calientes a otras con menos temperatura; balancear cargas de trabajo en elementos de procesamiento paralelo; decrementar las comunicaciones entre tareas lo cual es particularmente eficiente energéticamente en sistemas de redes de sensores inalámbricos (WSN) (Guo et al., 2014; Stangaciu et al., 2013b; Yi et al., 2009) , y garantizar, al mismo tiempo, el cumplimiento de las restricciones de tiempo real de todo el sistema.

En la literatura e investigación sobre Sistemas de Tiempo Real se pueden encontrar diversas técnicas de ahorro energético (Stangaciu et al., 2015; Mehariya et al., 2014; Stangaciu et al., 2013a; Dharwar et al., 2012). En esta tesis doctoral se aborda el escalamiento dinámico del voltaje (DVS - Dynamic Voltage Scaling) del procesador, también conocido como escalamiento de frecuencia y voltaje dinámico (DVFS - Dynamic Volatage and Frequency Scaling). Esta técnica aprovecha la relación convexa, y normalmente cuadrática, entre el consumo de energía del procesador y el voltaje de alimentación. Adicionalmente, estas técnicas han sido extendidas para disminuir la energía consumida por la memoria del sistema (Lee et al., 2012; Kim et al., 2011; Liang et al., 2008) y en el consumo de energía en redes de comunicaciones (Fateh & Govindarasu, 2015; Yi et al., 2009; Kumar et al., 2008).

El consumo global de energía en sistemas de cómputo puede contribuir en la decisión del balance de carga de trabajo en la fase de desarrollo de código. Este criterio, combinado con otros tales como planificabilidad, retardos en las comunicaciones, precisión/exactitud en aplicaciones de control, contribuyen a la determinación del movimiento dinámico de código de una forma segura y al balance de cargas en tiempo de ejecución en sistemas empotrados de control basados en soportes *middleware*. Sin embargo, esta tesis se enfoca principalmente en el problema de asignación de tareas, y más exactamente, en algoritmos de análisis de viabilidad de planificación llevados a cabo en la migración o emigración de tareas, así como en la gestión energética del sistema. Aunque algunos autores llevan a cabo estas dos fases (análisis de viabilidad de planificación y escalado de frecuencia) de forma separada, estos análisis están fuertemente relacionados y en la mayoría de casos pueden ser llevadas a cabo conjuntamente.

Objetivos

El objetivo principal de la tesis es el desarrollo de mecanismos para la gestión de consumo de energía en sistemas de computo fuertemente interconectados operando en ámbitos con diferentes niveles de criticidad en donde se deben garantizar requisitos de tiempo real, seguridad y confianza. Estos mecanismos se presentan como funcionalidades en el marco del diseño de *middleware* basados en el concepto de núcleo de control.

En este trabajo de tesis se tienen en cuenta tres componentes principales dentro de un sistema *middleware*: *Un primer* componente es el responsable de gestionar las peticiones que van llegando cuando tareas nuevas y activas requieren ser ejecutadas y suspendidas respectivamente. Este componente de gestión interactúa con un segundo componente obteniendo información acerca de los recursos disponibles para aceptar las tareas nuevas. *Un segundo* componente lleva a cabo un análisis de factibilidad para determinar si las nuevas tareas pueden ser aceptadas sin poner en riesgo las restricciones temporales del sistema. Al mismo tiempo se calcula una nueva frecuencia de operación para el procesador. *Un tercer* componente monitoriza y controla la ejecución de las tareas utilizando una política de planificación basada en prioridades y aplica a su vez, una optimización extra en el control de frecuencia del procesador basado en la carga de trabajo en cada instante.

En la investigación nos centraremos principalmente en el segundo componente mencionado, proporcionando un algoritmo que analiza la admisión de tareas de tiempo real estricto en términos del escalado de frecuencia del procesador, con el objeto de utilizarlo de forma dinámica en tiempo de ejecución. La clave de la aplicabilidad del algoritmo es su bajo y predecible coste computacional. El tercer componente, también será abordado, aunque en menor medida, para sistemas de tiempo real cuyas restricciones temporales sean flexibles.

Los objetivos particulares de esta tesis son los siguientes:

- Determinar los requisitos mínimos, las funcionalidades y los componentes necesarios a ser considerados en un marco de desarrollo *middleware* para sistemas de control empotrados.

- Dentro de las funcionalidades definidas en el marco del núcleo de control, y específicamente en la gestión de recursos, este trabajo se enfocará en el control energético del sistema con la propuesta de un nuevo algoritmo para ser utilizado en tiempo de ejecución durante las fases de asignación de tareas y frecuencias de procesador en entornos con restricciones de tiempo real estricto.

- Especificación de un conjunto de métodos adicionales para una optimización del consumo energético en subsistemas con requisitos de tiempo real flexibles en donde este tipo de restricciones permite una optimización completa del ahorro energético mediante el aprovechamiento de los instantes ociosos del procesador que sólo pueden detectarse en tiempo de ejecución.

- Evaluar el rendimiento de los algoritmos de optimización de consumo energético propuestos contra otras propuestas existentes. La frecuencia de procesador mínima se calcula en términos de consumo de energía, grado de aceptación de tareas y coste computacional real de los algoritmos. Adicionalmente se analizará la predecibilidad en la ejecución y comportamiento de los algoritmos en relación a la llegada continúa de tareas.

Contribuciones

La principal aportación de esta tesis consiste en la definición de mecanismos de gestión de consumo energético dentro de una arquitectura *middleware* basada en el concepto de núcleo de control. Concretamente se propone un algoritmo nuevo para el análisis de factibilidad y el cálculo de frecuencias de procesador basado en cambios en tiempo de ejecución del conjunto de tareas de tiempo real. Estos cambios se derivan de la inclusión o expulsión de hilos de ejecución en el sistema de cómputo.

Este análisis se complementa con la propuesta de métodos de optimización dinámicos que ajustan el consumo energético basado en las condiciones de carga de cómputo reales en cada instante. Estos métodos son aplicados durante la ejecución del sistema y se basan en la variación de la frecuencia de operación como respuesta a instantes ociosos (instantes *idle*) de procesador debidos principalmente a terminaciones anticipadas de tareas. Estos cambios en la ejecución de las tareas son detectados por lo general únicamente en tiempo de ejecución.

Las contribuciones que aporta este documento pueden resumirse como sigue:

- (Capítulo 2). Se presenta una *arquitectura middleware* basado en el concepto de núcleo de control.

- (Capítulo 3). Se presenta un nuevo *algoritmo* para el análisis de planificabilidad y para el cálculo de la frecuencia de procesador que *minimiza el consumo energético del procesador*.

- (Capítulo 4). Se presentan procedimientos para la *gestión de instantes ociosos de procesador* y se propone un algoritmo para la gestión *energética en tiempo de ejecución*.

- (Capítulo 5). Se presenta un conjunto de herramientas y procedimientos para *la evaluación* de *sistemas de tiempo real basadas en simulaciones*.

Estructura del documento

El contenido del documento de esta tesis se organiza de la siguiente forma:

- Este capítulo, que presenta una introducción a la temática de la tesis, las principales aportaciones del documento y la organización del documento.

- Capítulo 1: Se presenta un resumen del estado del arte de políticas de planificación de tiempo real que será utilizadas posteriormente en las propuestas de esta tesis.

- Capítulo 2: En este capítulo se muestra un modelo para el desarrollo de aplicaciones de control en entornos empotrados con restricciones de tiempo real basado en el concepto de "kernel de control". Para este próposito, se abordará el diseño y la implementación de una arquitectura de control distribuida desde un marco de desarrollo "middleware", enunciando para ello los requisitos y los servicios mínimos que respectivamente deberían cumplirse y ofrecerse. En este sentido, se han específicado un conjunto de actividades agrupadas en componentes software cuya asociación determinará el modelo *middleware*. Así mísmo, se definirán las principales interfaces de comunicación entre componentes y el usuario.

- Capítulo 3: En este capítulo se hace un resumen de los trabajos relacionados con la gestión de energía para el cálculo de la frecuencia mínima. Se presenta un modelo del sistema: tareas, procesador,costes; en los cuales esta basado el capítulo. Se proponen modificaciones de análisis de planificabilidad existentes para el cálculo de la frecuencia de procesador. Y finalmente se presenta un nuevo algoritmo para el análisis de planificabilidad y para el cálculo de la frecuencia de procesador mínima que optimiza el ahorro energético.

- Capítulo 4:En este capítulo se introduce la gestión dinámica de la frecuencia de procesador, se presentan las motivaciones para el uso de esta técnica y se hace una valoración de los trabajos relacionados. Se presenta un modelo del sistema: tareas, procesador y consumo, en los cuales está basado el capítulo. Como parte final, se proponen un conjunto de procedimientos para la gestión de tiempos ociosos del procesador desde el punto de vista energético. Igualmente, se propone un algoritmo dinámico para el cálculo de las frecuencias de procesador en tiempo de ejecución.

- Capítulo 5: En este capítulo se evalúan experimentalmente los algoritmos propuestos en los dos capítulos anteriores a través de simulaciones.

- Capítulo 6: En este capítulo se presenta un caso de estudio que evalúa en la práctica la arquitectura planteada para el *middleware* de núcleo de control.

- Capítulo 7: En este capítulo se presentan las conclusiones generales de todo el documento y se proponen posible ampliaciones de los resultados y futuras líneas de investigación.

Capítulo 1

PLANIFICACIÓN DEL USO DE CPU

1.1. Introducción

Se presenta un resumen del estado del arte de políticas de planificación de tiempo real que será utilizadas posteriormente en las propuestas de esta tesis.

1.1.1. Modelo de tareas

En este capítulo utilizaremos un modelo de tareas periódico. Supondremos un conjunto $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ de *n* tareas expulsivas de tiempo real ejecutándose sobre un sistema uni-procesador, en donde cada tarea $\tau_i(T_i, D_i, C_i)$ está caracterizada por un período T_i, un plazo máximo de ejecución relativa D_i y un tiempo de ejecución para el peor caso C_i. Además supondremos instantes críticos de activación para las tareas, es decir, desfaces iniciales de activación, iguales a 0. La planificación de las tareas se llevará a cabo utilizando un planificador por prioridades fijas y estará ordenado de mayor a menor prioridad, siendo τ_1 la tarea de mayor prioridad y τ_n la tarea de menor prioridad. El conjunto de períodos de todas las tareas estará representado por la letra mayúscula y en negrita $\mathbf{T} = \{T_1, T_2, ..., T_n\}$, al igual que el tiempo de ejecución del peor caso para todas las tareas $\mathbf{C} = \{C_1, C_2, ..., C_n\}$.

Consideraremos que en la CPU se ejecutarán únicamente tareas de tiempo real, siendo esto no una restricción sino una simplificación para llevar a cabo el análisis.

1.2. Planificabilidad en sistemas periódicos con asignación de prioridades fijas

1.2.1. Prueba de planificabilidad Liu Layland (LL)

Uno de los test más conocidos es el de Liu y Layland (Sha et al., 2004), en el que un conjunto de n tareas será planificable por RM si:

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1) \tag{1.1}$$

en donde U_i hace referencia a la utilidad de la tarea i. A este test nos referiremos como prueba *LL*. Haciendo $n \to \infty$ tenemos que el factor $n(2^{1/n} - 1)$ es aproximadamente igual a el $ln\ 2$, por tanto para conjuntos de tareas grandes la utilidad máxima va tendiendo aproximadamente a $0{,}69$, aunque en la práctica por el número de tareas este valor de utilidad variará entre $0{,}72$ y $0{,}77$. Este test es *suficiente pero no necesario*

1.2.2. Prueba del límite hiperbólico (HB)

Otro test que mejora un poco el pesimismo del test LL sin aumentar, es el del límite hiperbólico propuesto por Bini en (Bini et al., 2003), en el que un conjunto de n tareas será planificable por RM si:

$$\prod_{i=1}^{n}(U_i + 1) \leq 2 \tag{1.2}$$

a este método lo denominaremos método *HB*. Este test sigue siendo *suficiente pero no necesario*

Tal y como ocurre con el test LL, a medida que aumenta el número de tareas en el sistema, aumenta también la diferencia entre el factor α exacto y el aproximado por HB.

1.2.3. Análisis de tiempo de respuesta (RTA)

Un test exacto es el análisis de tiempo de respuesta (Sha et al., 2004), que lo denominaremos RTA, y está enunciado como:

$$\begin{cases} W_i^{(0)} = C_i \\ W_i^{(k)} = C_i + \sum\limits_{\forall j \in hp(i)} \left\lceil \dfrac{W_i^{k-1}}{T_j} \right\rceil C_j \end{cases} \tag{1.3}$$

donde hp(i) representa el conjunto de tareas con prioridad mayor que la tarea i. Con este test un conjunto de tareas periódico será planificalbe *sí y sólo sí* el tiempo de respuesta del peor caso para cada tarea es menor que D.

1.2.4. Análisis de Liu y Layland mejorado (LLM)

Este método se presenta como una extensión de la prueba *LL* (sección 1.1), para tareas con plazos de ejecución menor que el período. Este método esta basado en el análisis de la utilización Racu et al. (2005). Este algoritmo es un análisis de planificabilidad suficiente pero no necesario. La planificabilidad del sistema está determinada por el cumplimiento de la siguiente igualdad:

$$f_i \leq U(p, \Delta_i) \tag{1.4}$$

donde f_i está definido como:

$$f_i = \sum_{j \in H_p} \frac{C_j}{T_j} + \sum_{k \in H_1} \frac{C_k}{T_i} + \frac{C_i}{T_i}$$

de donde H_1 y H_p está definido como:

- H_1 está formado por el conjunto de tareas de mayor prioridad que pueden expulsar la tarea τ_i una única vez antes de su plazo de ejecución D_i, es decir, el conjunto de tareas de mayor prioridad con períodos mayores o igual a D_i.
- H_p está formado por el conjunto de tareas de mayor prioridad que puede expulsar la tarea τ_i más de una vez antes de su plazo de ejecución D_i, o en otras palabras, el conjunto de tareas de mayor prioridad con períodos de ejecución menores que D_i.

Y $U(p, \Delta_i)$ esta definido como:

$$U(p, \Delta_i) = \begin{cases} p((2\Delta_i)^{1/p} - 1) + 1 - \Delta_i & 0{,}5 \leq \Delta_i \leq 1 \\ \Delta_i & 0 \leq \Delta_i < 0{,}5 \end{cases}$$

donde Δ_i y p son:

$$\Delta_i = \frac{D_i}{T_i} \qquad p = num(H_p) + 1$$

$num(H_p)$ corresponde al número de tareas en el conjunto H_p.

1.2.5. Análisis de planificación mediante regiones de planificabilidad - Método $\mathcal{S}$

Abordaremos inicialmente el análisis de planificabilidad y el cálculo mínimo y estático de α desde la perspectiva del análisis en el espacio C (*C-space*). Con este análisis los parámetros de las tareas serán vistas como un punto en un espacio específico en comparación con otras tareas, los períodos T_i y los plazos D_i serán considerados como parámetros fijos, mientras que los tiempos de cómputo C_i serán considerados parámetros variables libres. Por consiguiente, obtendremos una restricción para las variables C_i, que dependerá de todos los T_i y D_i de las tareas.

El análisis de planificabilidad se reduce a verificar si un punto se encuentra dentro de una región delimitada por coordenadas C_i. Esta región la denominaremos como $\boldsymbol{R}_n$, la cual fue inicialmente propuesta por Lehoczky en (Lehoczky et al., 1989):

$$\boldsymbol{R}_n(T_1,...,T_n,D_1,...,D_n)|_{D_i=T_i} = \{(C_1,...,C_n) \in \mathbb{R}^n_+ : \mathcal{T}_n \text{ es planificable por PF}\} \tag{1.5}$$

La formulación formal de la región de factibilidad $\boldsymbol{R}_n$ fue derivada de (Lehoczky et al., 1989) y convenientemente demostrada en (Bini & Buttazzo, 2004) mediante el siguiente teorema:

Teorema 1.1. *(Teorema 2 en (Bini & Buttazzo, 2004)). Cuando los plazos de ejecución D_i son iguales a los períodos T_i, la región $\boldsymbol{R}_n$ de un conjunto de tareas planificable está dado por:*

$$\boldsymbol{R}_n(T_1,...,T_n,D_1,...,D_n)|_{D_i=T_i} = \{(C_1,...,C_n) \in \mathbb{R}^n_+ : \max_{i=1...n} \min_{t\in\mathcal{S}_i} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\} \tag{1.6}$$

donde $\mathcal{S}_i = \{kT_j : j = 1 \ldots i, k = 1 \ldots \left\lfloor \frac{T_i}{T_j} \right\rfloor\}$.

El conjunto de elementos de $\mathcal{S}_i$ representan conceptualmente los tiempos de activación de las tareas con prioridades mayores que τ_i antes de D_i y el plazo de ejecución D de la tarea τ_i y para ello se supone $\phi_i = 0$. Este subconjunto de valores es a lo que denominaremos *puntos de planificación rate monotonic*. En (Bini & Buttazzo, 2004) este teorema utiliza operadores lógicos para la representación de máx ($\wedge$) y mín ($\vee$).

Estos puntos $\mathcal{S}_i$ pueden extraerse del intervalo continúo t entre $(0, T_i]$, ya que este intervalo continúo puede acotarse a un subconjunto de valores para los cuales la expresión $\lceil t/T_j \rceil / t$ (ver ecuación 1.6) no es extrictamente decreciente. Y lo anterior ocurre en los mínimos locales de la función, lo cual tienen lugar cuando t es un múltiplo de uno de los períodos T_j, la función es continua por la izquierda y salta a un valor mayor por la derecha. Por tanto, para determinar si τ_i puede alcanzar su deadline únicamente es necesario buscar sobre esos mínimos locales, los múltiplos de $T_j \leq T_i$ para $1 \leq j \leq i$.

Para una mejor comprensión y caracterización en términos de complejidad y costes de ejecución de este método y de las posteriores propuestas, vamos utilizar una representación matricial de la ecuación 1.6. Los elementos del conjunto $\mathcal{S}_i$ pueden ser representados por la siguiente matriz:

$$\mathcal{S}_i = \{s_{kl}\}_i \quad \text{donde} \quad k \in [1,\ldots,i] \wedge l \in [1,\ldots,\left\lfloor \frac{T_i}{T_1} \right\rfloor] \tag{1.7}$$

siendo el conjunto de elementos $\mathcal{S}_i$ igual a una matriz con kl elementos, siendo s_{kl} igual al producto:

$$\{s_{kl}\}_i = J \otimes K$$

en donde:

$$J = T_k = \begin{bmatrix} T_1 & T_2 & \ldots & T_k \end{bmatrix} \ \forall k \in [1 \ldots i]$$

y K igual a:

$$K = \begin{bmatrix} 1 & 2 & 3 & \ldots & \left\lfloor \frac{T_i}{T_k} \right\rfloor \end{bmatrix} \ \forall k \in [1 \ldots i]$$

siendo el resultado del producto s_{kl} igual a:

$$\{s_{kl}\}_i = \begin{bmatrix} T_1 \\ T_2 \\ T3 \\ \dots \\ T_k \end{bmatrix} \otimes \begin{bmatrix} 1 & 2 & 3 & \dots & \left\lfloor \frac{T_i}{T_k} \right\rfloor \end{bmatrix} = \begin{bmatrix} T_1 & 2T_1 & 3T_1 & \dots & \left\lfloor \frac{T_i}{T_1} \right\rfloor T_1 \\ T_2 & 2T_2 & 3T_2 & \dots & \left\lfloor \frac{T_i}{T_2} \right\rfloor T_2 \\ \dots & \dots & \dots & \dots & \dots \\ T_k & 2T_k & 3T_k & \dots & \left\lfloor \frac{T_i}{T_k} \right\rfloor T_k \end{bmatrix} = \begin{bmatrix} s_{11} & s_{12} & s_{13} & \dots & s_{1l} \\ s_{21} & s_{22} & s_{23} & \dots & s_{2l} \\ s_{31} & s_{32} & s_{33} & \dots & s_{3l} \\ \dots & \dots & \dots & \dots & \dots \\ s_{k1} & 0 & 0 & \dots & 0 \end{bmatrix} \tag{1.8}$$

dando como resultado una matriz de dimensiones ix$\left\lfloor \frac{T_i}{T_1} \right\rfloor$.

El número de columnas en cada fila de la matriz s_{kl} esta determinado por la diferencia entre los períodos de las tareas ($\left\lfloor \frac{T_i}{T_k} \right\rfloor$), por consiguiente varios elementos de la matrix es posible que sean iguales a 0 y por tanto el conjunto $\mathcal{S}_i$ puede definirse como:

$$\mathcal{S}_i^* = S_i \ / \ s_{kl} \neq 0$$

El total de elementos de $\mathcal{S}_i^*$ es definido por:

$$\text{Tamaño}(\mathcal{S}_i^*) = \sum_{j=1}^{i} \left\lfloor \frac{T_i}{T_j} \right\rfloor \tag{1.9}$$

Observando la ecuación 1.9, podemos darnos cuenta que al estar el tamaño del conjunto $\mathcal{S}_i^*$ determinado por la separación entre los períodos de las tareas, cuanto mayor es la diferencia entre $\lfloor T_i/T_j \rfloor$ mayor será el número de expresiones a evaluar y por tanto mayor su coste computacional.

Por otro lado, siguiendo con la expresión de la ecuación 1.6 esta puede ser expresada como:

$$\max_{i=1\dots n} \min_{t \in \mathcal{S}_i} M(t) \leq t$$

de donde redefiniremos $M(t)$ como $M(\mathcal{S}_i)$:

$$\max_{i=1\dots n} \min M_i(\mathcal{S}_i) \leq \mathcal{S}_i \tag{1.10}$$

en donde $M_i(\mathcal{S}_i)$ será igual a:

$$M_i(\mathcal{S}_i) = \{m_{kl}\}_i \tag{1.11}$$

$$\{m_{kl}\}_i = \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j \ / \ s_{kl} \in \mathcal{S}_i^*$$

y reemplazando la ecuación 1.8 en la expresión anterior, nos dará como resultado una suma matricial punto a punto:

$$\{m_{kl}\}_i = \begin{bmatrix} \left\lceil \frac{1}{T_1} s_{11} \right\rceil C_1 & \dots & \left\lceil \frac{1}{T_1} s_{1l} \right\rceil C_1 \\ \left\lceil \frac{1}{T_1} s_{21} \right\rceil C_1 & \dots & \left\lceil \frac{1}{T_1} s_{2l} \right\rceil C_1 \\ \dots & \dots & \dots \\ \left\lceil \frac{1}{T_1} s_{k1} \right\rceil C_1 & \dots & \left\lceil \frac{1}{T_1} s_{kl} \right\rceil C_1 \end{bmatrix} + \begin{bmatrix} \left\lceil \frac{1}{T_2} s_{11} \right\rceil C_2 & \dots & \left\lceil \frac{1}{T_2} s_{1l} \right\rceil C_2 \\ \left\lceil \frac{1}{T_2} s_{21} \right\rceil C_2 & \dots & \left\lceil \frac{1}{T_2} s_{2l} \right\rceil C_2 \\ \dots & \dots & \dots \\ \left\lceil \frac{1}{T_2} s_{k1} \right\rceil C_2 & \dots & \left\lceil \frac{1}{T_2} s_{kl} \right\rceil C_2 \end{bmatrix} + \dots + \begin{bmatrix} \left\lceil \frac{1}{T_i} s_{11} \right\rceil C_i & \dots & \left\lceil \frac{1}{T_i} s_{1l} \right\rceil C_i \\ \left\lceil \frac{1}{T_i} s_{21} \right\rceil C_i & \dots & \left\lceil \frac{1}{T_i} s_{2l} \right\rceil C_i \\ \dots & \dots & \dots \\ \left\lceil \frac{1}{T_i} s_{k1} \right\rceil C_i & \dots & \left\lceil \frac{1}{T_i} s_{kl} \right\rceil C_i \end{bmatrix}$$

La suma resultante m_{kl} será igual a:

$$\{m_{kl}\}_i = \begin{bmatrix} m_{11} & m_{12} & \dots & m_{1l} \\ m_{21} & m_{22} & \dots & m_{2l} \\ \dots & \dots & \dots & \dots \\ m_{k1} & m_{k2} & \dots & m_{kl} \end{bmatrix}_{i\mathbf{X}\left\lfloor \frac{T_i}{T_1} \right\rfloor}$$

De las expresiones anteriores podemos determinar que valores repetidos s_{kl} darán como resultado valores redundantes en m_{kl}, los cuales no habría necesidad de volverlos a computar. Por tanto, $\mathcal{S}_i^*$ puede redefinirse nuevamente como:

$$\mathcal{S}_i^* = \mathcal{S}_i \ / \ s_{kl} \neq 0 \wedge \exists! s_{kl} \tag{1.12}$$

Finalmente la región $\boldsymbol{R}_n$ (ecuación 1.5) quedará definida como:

$$\boldsymbol{R}_n(T_1,...,T_n,D_1,...,D_n)|_{D_i=T_i} = \{(C_1,...,C_n) \in \mathbb{R}_+^n : \max_{i=1...n} \min M_i(\mathcal{S}_i^*) \leq \mathcal{S}_i^*\} \tag{1.13}$$

donde $\mathcal{S}_i^*$ es igual a la ecuación 1.12.

La ecuación 1.16 es apropiada para establecer las expresiones que definen los límites/planos que acotan la región en la que deben ubicarse los valores absolutos del tiempo de ejecución C_i de las tareas para que el sistema sea planificable. Para obtener estas expresiones se establece C_i como término variable. Sin embargo, para

a partir de esta ecuación, la ecuación 1.10 puede quedar expresada finalmente como (ver figura 1.1):

$$\max_i \left[\min_{\forall k,l} M_i \right] = \varepsilon_i^s \tag{1.14}$$

de cuya ecuación se establece que para que el *conjunto de tareas sea planificable* se debe cumplir que:

$$\varepsilon_i^s \leq \mathcal{S}_i^{**} \tag{1.15}$$

$$\varepsilon_i^s \leq s_{kl}$$

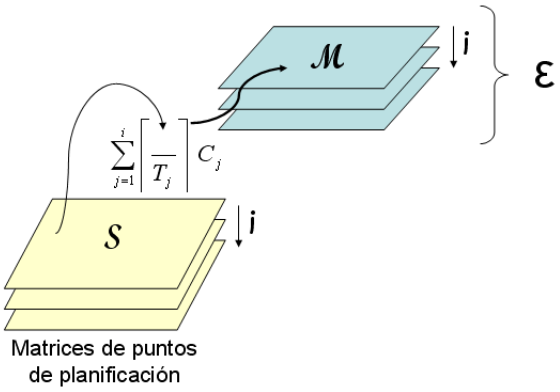

Figura 1.1: Región de planificabilidad ε_i a partir de los puntos $\mathcal{S}_i$

Cuyas ecuaciones serán las que utilizaremos como referencia a lo largo de este capítulo.

Llegado a este punto, la región $\mathbb{R}_n$ (ecuación 1.6) puede ser redefinida de la siguiente forma:

$$\mathbb{R}_n(T_1,...,T_n,D_1,...,D_n)|_{D_i=T_i} = \{(C_1,...,C_n) \in R_+^n : \ \varepsilon_i^s \leq \ \mathcal{S}_i^{**}\} \tag{1.16}$$

1.2.6. Análisis de planificación mediante regiones de planificabilidad - Región $\mathcal{P}_i$

La región $\mathbb{R}_n$ definida en la ecuación 1.16, esta delimitada por hiper-planos que determinan el espacio en el que deben encontrarse los puntos C_i para que el conjunto de tareas sea planificable por FP. Sin embargo, el análisis de conjuntos grandes de tareas mediante este método requiere de un número elevado de ecuaciones a ser procesadas, que dependerá del número de elementos de $\mathcal{S}_i$ y de la

diferencia entre los períodos de las tareas, impidiendo muchas veces por tanto su uso en aplicaciones prácticas.

No obstante y tal como se demuestra en (Bini & Buttazzo, 2004), la resolución de la ecuación 1.16 da como resultado varias ecuaciones innecesarias debidas al tamaño del conjunto $\mathcal{S}_i$. Por tanto la idea es reducir el número de ecuaciones por la eliminación de elementos redundates en $\mathcal{S}_i$, lo que ha llevado a la enunciación del siguiente teorema:

Teorema 1.2. *(Teorema 3 en (Bini & Buttazzo, 2004)). La región de planificabilidad del conjuto de tareas $\mathbb{R}_n$, tal y como se define en la ecuación 1.6, esta dada por:*

$$\mathbb{R}_n(T_1,...,T_n,D_1,...,D_n) = \{(C_1,...,C_n) \in R_+^n : \max_{i=1...n} \min_{t \in \mathcal{P}_{i-1}(D_i)} C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \leq t\}$$

en donde $\mathcal{P}_i(t)$ esta definido por la siguiente ecuación:

$$\begin{cases} \mathcal{P}_0(t) = \{t\} \\ \mathcal{P}_i(t) = \mathcal{P}_{i-1}\left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i\right) \cup \mathcal{P}_{i-1}(t)\} \end{cases} \tag{1.17}$$

Se puede observar que la diferencia entre la enunciación del teorema 1.1 y el teorema 1.2 es la inclusión de $\mathcal{P}_{i-1}(t)$ en lugar de $\mathcal{S}_i$. $\mathcal{P}_{i-1}(t)$ es un subconjunto de $\mathcal{S}_i$ ($\mathcal{P}_{i-1}(t) \subseteq \mathcal{S}_i$), lo cual reduce drásticamente el número de ecuaciones a evaluar y limita el tiempo necesario para establecer si un conjunto de tareas se encuentra dentro de la región $\mathbb{R}_n$.

i	T_i	D_i	S_i	$\mathcal{P}_{i-1}(D_i)$
1	3	3	3	3
2	8	8	3,6,8	6,8
3	20	20	3,6,8,9,12,15,16,18,20	15,16,18,20

Tabla 1.1: Conjunto de tareas para comparar los puntos $\mathcal{S}_i$ y $\mathcal{P}_{i-1}$

Recordemos que los puntos de planificación son los puntos para los cuales $\lceil t/T_i \rceil / t$ no es extrictamente decreciente. Con el fin de esclarecer un poco mejor este concepto, en la figura 1.2 se muestra el resultado de la evaluación de la expresión $\lceil t/T_i \rceil / t$ para la tarea T_3 del conjunto de tareas propuesto en la tabla 1.2.6. t es evaluado para el intervalo continúo $(0, 20]$ y $T_i = [3, 8, 20]$ para $i \in [1, 2, 3]$. En la figura los puntos S_i son representados por cuadrados y los puntos P_{i-1} por elipses. Como se observa en la figura 1.2 no es necesario evaluar t a lo largo de un intervalo continúo, si no únicamente en un conjunto determinado de valores significativos representados por $\mathcal{S}_i$ y acotados por $\mathcal{P}_{i-1}$. En esta misma figura se puede observar el subconjunto $\mathcal{P}_{i-1}$ formado apartir de los puntos $\mathcal{S}_i$ cuando $D = T$.

Siguiendo la misma estrategia utilizada en la representación de los puntos $\mathcal{S}_i$ (ecuación 1.12), los puntos de planificabilidad $\mathcal{P}_{i-1}(t)$ también pueden ser expresados de forma matricial. En esa expresión matricial el número de elementos, o lo que es igual, el número de valores a calcular será igual a:

$$\text{Tamaño}(P_c(D_i)) = \frac{i(i-1)}{2} \tag{1.18}$$

Siguiendo la estructura utilizada en la enunciación de la región de planificabilidad con los puntos $\mathcal{S}_i$, tenemos que el teorema 1.2 puede ser expresado como:

$$\max_{i=1...n} \min_{t \in \mathcal{P}_{i-1}} M(t) \leq t$$

donde redefiniendo $M(t)$ como $M(\mathcal{P}_{i-1})$:

$$\max_{i=1...n} \min M_i(\mathcal{P}_{i-1}) \leq \mathcal{P}_{i-1} \tag{1.19}$$

en donde $M_i(\mathcal{P}_{i-1})$ será igual a:

$$M_i(\mathcal{P}_{i-1}) = \{m\}_i \ / \tag{1.20}$$

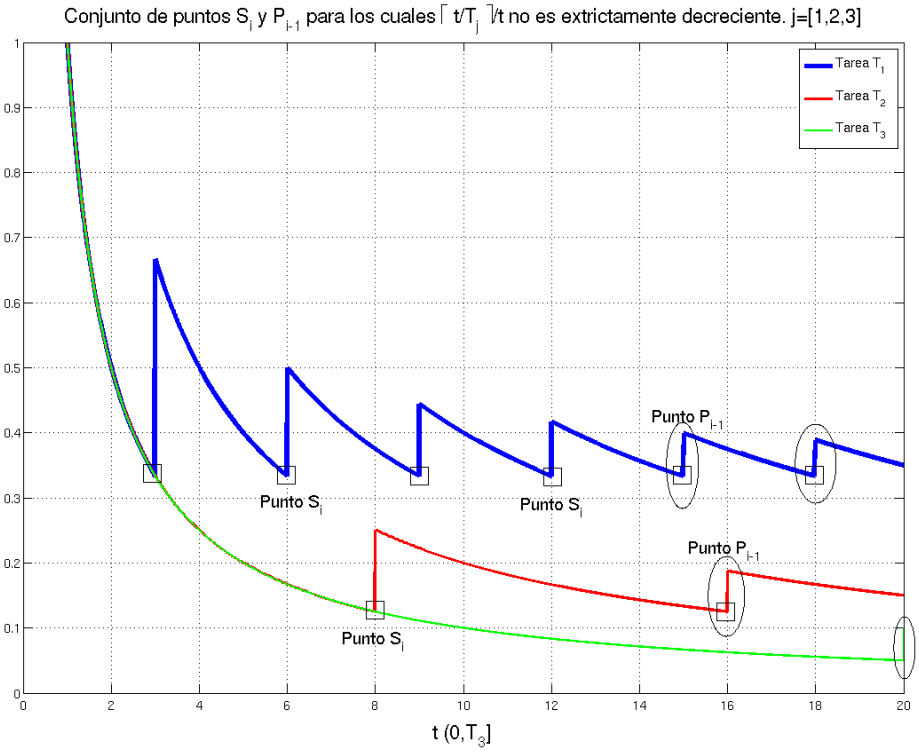

Figura 1.2: Puntos de planificabilidad $\mathcal{S}_i$ y subconjunto $\mathcal{P}_{i-1}$ para la tarea T_3. Estos puntos son aquellos para los cuales $\lceil t/T_i \rceil / t$ no es extrictamente decreciente, en donde $i \in [1, 2, 3]$

$$\{m\}_i = \sum_{j=1}^{i} \left\lceil \frac{p}{T_j} \right\rceil C_j \quad / \; p \in \mathcal{P}^*_{i-1}$$

$$\text{en donde} \quad \mathcal{P}^*_{i-1} = \mathcal{P}_{i-1} \; / p \; \exists! p$$

esto nos dará como resultado una suma matricial punto a punto entre las matrices de los puntos de planificación p relacionadas con cada uno de los períodos T_j y los tiempos de computo C_j.

A partir de la ecuación 1.20, la expresión izquierda de la ecuación 1.19 puede quedar expresada finalmente como:

$$\max_i \left[\min M_i(\mathcal{P}_{i-1})\right] = \varepsilon_i^p \tag{1.21}$$

Finalmente se establece que un *conjunto de tareas periódico será planificable* bajo prioridades fijas **sí y sólo sí** se cumple la relación:

$$\varepsilon_i^p \leq \mathcal{P}^*_{i-1} \tag{1.22}$$

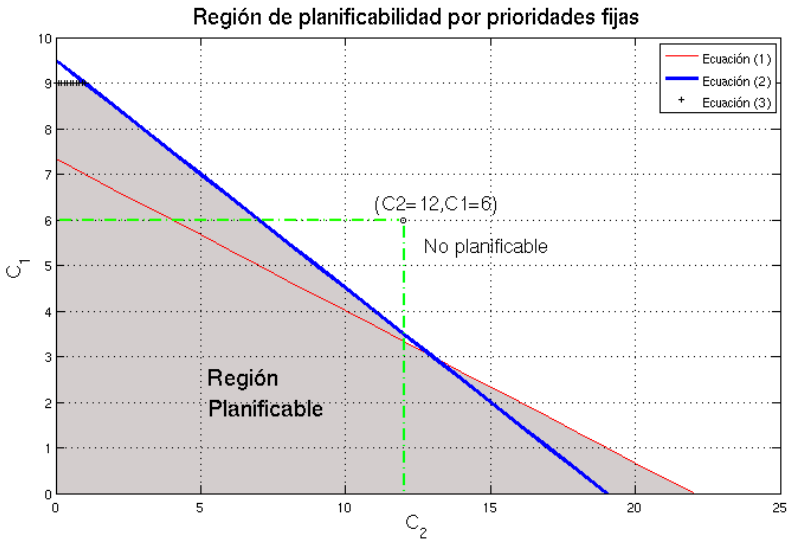

Figura 1.3: Región de planificabilidad para las tareas de la tabla 1.2

i	T_i	D_i	C_i	$\mathcal{P}_{i-1}(D_i)$
1	9.5	9	6	9
2	24	22	12	22,19

Tabla 1.2: Conjunto de tareas para hallar la región de planificabilidad

Ejemplo para la región de planificabilidad

Vamos a considerar un conjunto de tareas con plazos de ejecución más pequeños que el período. Las tareas son mostradas en la tabla 1.2. Aplicando el teorema 1.2 obtenemos las siguientes desigualdades:

$$\begin{cases} i = 1 & \\ & C_1 \leq 9 \quad (3) \\ i = 2 & \\ \| & 3C_1 + C_2 \leq 22 \quad (1) \\ \| & 2C_1 + C_2 \leq 19 \quad (2) \end{cases}$$

en donde el símbolo { representa una operación lógica AND entre las áreas resultantes de cada iteracción (i), y el símbolo $\|$ representa una operación lógica OR entre las ecuaciones resultantes dentro de cada iteracción, es decir, en el ejemplo se realiza una operación lógica OR del área bajo la curva de las ecuaciones (1) y (2), y una operación lógica AND entre la resultante anterior y la ecuación (3). El resultado de esta operación da como resultado la región de planificabilidad del conjunto de tareas, el cual es representado en la figura 1.3 como área sombreada.

En esta misma figura la región sombreada representa el área en la que deben encontrarse los puntos C_i para que el sistema sea planificable, si dibujamos los puntos C_i definidos en la tabla 1.2, podemos darnos cuenta que la intercepción de estos se encuentra fuera de la región de planificabilidad, tal y como se muestra en la figura 1.3, y por tanto ese conjunto de tareas no es planificable mediante prioridades fijas.

1.3. Planificabilidad en sistemas periódicos con asignación de prioridades dinámica

Aunque esta tesis se centra en esquemas de planificación con prioridades fijas, en la evaluación de las propuestas se hace una comparación con el algoritmo EDF(Earliest deadline first).

1.3.1. Análisis de planificación EDF basado en utilidades

El algoritmo EDF (Earliest Deadline First) asigna las prioridades de las tareas en función de los plazos absolutos de ejecución: la tarea con el plazo de ejecución más próximo será ejecutada con la mayor prioridad. Este algoritmo es óptimo cuando $D = T$ desde el punto de vista de factibilidad: si existe un plan factible para un conjunto determinado de tareas, entonces será posible planificarlo con el algoritmo EDF.

Con este esquema de planificación los plazos están garantizados si se cumple que:

- Cuando $D = T$:

$$\sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq 1 \tag{1.23}$$

  Análisis de planificabilidad suficiente y necesario.

- Cuando $D < T$:

$$\sum_{i=1}^{n} U_i = \sum_{i=1}^{n} \frac{C_i}{D_i} \leq 1 \tag{1.24}$$

  Análisis de planificabilidad suficiente pero no necesario.

Capítulo 2

ARQUITECTURA DE CONTROL DE PROCESOS

2.1. Introducción

Los sistemas de control empotrados son usados en aplicaciones civiles e industriales para monitorizar y controlar equipos distribuidos de control con intervención humana remota. Además, esos sistemas usan redes de comunicaciones (Coronel et al., 2006a) para interconectar sensores, controladores, terminales de operadores y actuadores. De estos sistemas empotrados de tiempo real, los más sofisticados incorporan sistemas operativos con núcleos robustos que proveen funcionalidades, servicios e interfaces básicas para el control y el acceso a los recursos *hardware* del sistema. Igualmente se incorporan mecánismos para el manejo de excepciones frente a fallos, de modo que el sistema continúe prestando unos servicios mínimos aún trabajando en condiciones de emergencia (Coronel et al., 2008; Nicolau et al., 2008).

De la implementación de técnicas de control en sistemas por computador se pueden abstraer diversas funcionalidades o servicios que son comunes a un gran número de aplicaciones (Sánchez et al., 2014; Crespo et al., 2006). Ejemplos de estos son las interfaces sensor y actuador en lazos cerrados de control en donde muchas veces, por ejemplo, es necesario poder medir los tiempos derivados de la obtención e emisión de peticiones para calcular los retardos promedios, y en algunos casos, compensarlos directamente en el controlador. Antes que implementar el soporte de estos servicios para cada una de las aplicaciones o de proveer el soporte en el sistema operativo, otra opción más óptima y más utilizada en sistemas distribuidos es la de utilizar la tecnología *middleware*. Y es en esta tecnología en la que se basará este trabajo.

2.2. Kernel de Control

Los sistemas empotrados de tiempo real más sofisticados incorporan sistemas operativos con núcleos[1] robustos que proveen funcionalidades, servicios e interfaces básicas para el control y el acceso a los recursos *hardware* del sistema. Igualmente se incorporan mecanismos para el manejo de excepciones frente a fallos, de modo que el sistema continúe prestando unos servicios mínimos aún trabajando en condiciones de emergencia.

Siguiendo la premisa anterior, es posible proponer una solución pero orientada al control de procesos, ofreciendo de forma análoga al *kernel* de un sistema operativo, una serie de servicios e interfaces esenciales para la gestión de los componentes básicos en un sistema de control: *sensores*, *actuadores* y *controladores*, incorporando así mismo, mecanismos para asegurar en todo momento el comportamiento seguro del sistema bajo control. A este conjunto de servicios e interfaces lo denominaremos *Kernel de Control*.

[1] Más conocido en lengua inglesa como *Kernel*

2.2.1. Descripción

El término *Kernel de control* (CK)[2] ha sido utilizado en diversos artículos tales como (Albertos et al., 2006) y (Crespo et al., 2006), en los que se llevó a cabo una introducción preliminar del concepto. Sin embargo, mientras en el ámbito de sistemas operativos, debido entre otras cosas al largo estudio que se ha venido realizando alrededor de este tema, es bastante claro el concepto de *kernel* y de los servicios que este debe prestar, en el ámbito de control no hay un consenso común sobre lo que se entiende por *kernel* ni cuales deben ser los servicios básicos que este debe ofrecer ni como debe implementarse. Por tal motivo, en este capítulo propondremos un diseño de *kernel de control* estudiando las principales características y las funcionalidades mínimas que deberían ser incorporadas.enunciando las principales características y las funcionalidades mínimas que deberían ser consideradas.

Uno de los propósitos que se desea alcanzar con el CK es la transparencia en el diseño y desarrollo de aplicaciones de control de tiempo real, ofreciendo una interfaz para la configuración de comportamientos, modos de trabajo y ejecución de los componentes internos que conforman el kernel. En este sentido, el *kernel de control* gestionará igualmente la ejecución de un *nivel de aplicación de control*, que en términos de sistema operativo vendría a ser lo que se conoce como aplicación de usuario. Teniendo en cuenta un esquema de particionado de tareas como los expuestos en (Balbastre et al., 2006) y en (Cervin, 2003), el *nivel de aplicación de control* hace referencia principalmente a controladores y tareas opcionales asociadas a sensores y actuadores gestionados por el *kernel de control*.

En términos generales, el *kernel de control* proveerá una interfaz entre el sistema operativo de tiempo real (RTOS) y las actividades de control, o lo que es lo mismo, permitirá enlazar de forma transparente la abstracción del hardware implementado por el RTOS, tal como CPU, timers, memoria, dispositivos de E/S, entre otros, con las aplicaciones de control a través de mecanismos de adquisición, actuación y diversos algoritmos de supervisión y gestión de recursos para asegurar un rendimiento óptimo de todo el sistema de control (ver figura 2.1).

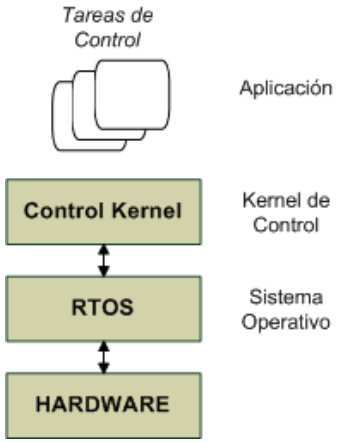

Figura 2.1: Esquema general del kernel de control

Uno de los requisitos más importantes que se debe incorporar en el diseño del *kernel de control* es la garantización de un comportamiento seguro de todo el sistema de control, llevándose a cabo con mayor o menor grado de rigurosidad en función de las restricciones temporales del proceso a controlar. Para ello, uno de los factores a cumplir por el kernel es el asegurar el envío y la entrega de todas las acciones de control a todos los actuadores a nivel local, independientemente del número de variables a ser controladas por un mismo procesador, y a nivel distribuido, sin importar la ubicación de los procesadores en la red.

Al margen del mal funcionamiento de componentes hardware, la seguridad del sistema de control se puede ver afectada por la pérdida de plazos de ejecución de los componentes software, pérdidas de mensajes, variaciones en los retardos de comunicación entre procesadores, cambios en la dinámica

[2] De sus iniciales en inglés *Control Kernel*

del sistema controlado, etc. En este contexto, la detección de estos fallos de ejecución unido a los fallos hardware son servicios básicos que deberían ser considerados en el diseño del *kernel de control*.

Este servicio de tolerancia a fallos del *kernel* está relacionado con la detección y gestión de situaciones anormales. Sin embargo, dependiendo de la naturaleza del fallo, la detección y la gestión se llevará a cabo a nivel del kernel, y en otros casos, la gestión y propagación del fallo se llevará exclusivamente a nivel de aplicación en donde se conocerá más detalladamente como manejar el error.

Con el mismo próposito de mantener un comportamiento seguro, el *kernel de control* puede disponer de diferentes niveles de calidad para la señal de control entregada a los actuadores y de la señal adquirida de sensores, pero estas deben ser suficientes como para evitar la inestabilidad del sistema de control. La calidad de estas señales dependerá del procesamiento realizado: cantidad de datos usados, algoritmo computacional ejecutado, entre otros. Así mismo, la selección del nivel de procesamiento a llevar a cabo dependerá del tiempo y los recursos disponibles que permitan cumplir los plazos de entrega establecidos para todo el sistema.

Trabajando en entornos empotrados en donde los recursos son limitados, la gestión que se haga de estos es crucial en el desempeño del sistema. Por tanto, la seguridad desde el punto de vista de control debe asegurarse indistintamente a la disponibilidad de recursos. En este sentido, se tendrán varios controladores basados en niveles diferentes de consumo de recursos, los cuales se apoyarán en modelos de orden reducido de la planta bajo control y que serán ejecutados alternativamente en función de la disponibilidad de recursos en cada instante. Igualmente, deberán definirse actuaciones seguras a ser ejecutadas en condiciones de emergencia, tales como: en ausencia de señal de control mantener la última acción o desconectar o llevar a un nivel seguro el proceso bajo control.

2.2.2. Modelo de tareas para el Kernel de control

Tal como es presentado por varios autores en la literatura (Xu et al. (2015), Lluesma et al. (2006),Cervin (2003)), el particionado de las actividades de control en subtareas permite una reducción del retardo o *jitter* de control en sistemas monoprocesador en los que se debe planificar la ejecución simultanea de varios procesos de control. Sin embargo, esta no es la única ventaja que se puede extraer de este tipo de técnicas, también es posible aprovechar el particionado para crear cierta independencia entre los componentes básicos de control creando de esta forma zonas aisladas de fallos, que proveen además, flexibilidad en la conmutación del código a ejecutar por cada uno de los componentes de forma separada.

Por tales motivos se ha decidido utilizar este tipo de técnicas en el *kernel de control*, siendo por tanto conveniente hacer un estudio en profundidad de los principales modelos de tareas propuestos en la literatura de tal forma que se pueda hacer una selección o modificación de los ya existentes o realizar una nueva propuesta, buscando el más apropiado para el desempeño y diseño del kernel de control. En este sentido, el autor llevó a cabo este análisis en diversas publicaciones Coronel (2008), Coronel et al. (2008), Simarro et al. (2008).

2.2.3. Kernel de control desde el punto de vista de implementación

El esquema básico del *kernel de control* expuesto en las secciones anteriores, es posible llevarlo a cabo a través de dos líneas de desarrollo principales: como parte anexa al sistema operativo o como un sistema *middleware*:

- **Como parte del sistema operativo:** Consiste en agregarle a los servicios predefinidos del sistema operativo (SO) nuevas funcionalidades que enmarquen los requerimientos del kernel de control. De este modo, el kernel de control se integrará de forma nativa en el SO y se deberán incorporar nuevas llamadas al sistema que permitan configurar los parámetros de los componentes de control y la conexión de estos con el nivel de aplicación (ver figura 2.2).

- **Como middleware:** En este caso, el sistema operativo mantiene sus funcionalidades originales y el kernel de control es implementado como una capa de software entre el nivel de aplicación de control y el SO. Las funcionalidades requeridas por el kernel son implementadas en esta capa

software, en forma de API[3] que permite la configuración interna de los componentes de control. Esta capa ofrece igualmente un acceso transparente a los servicios del SO.

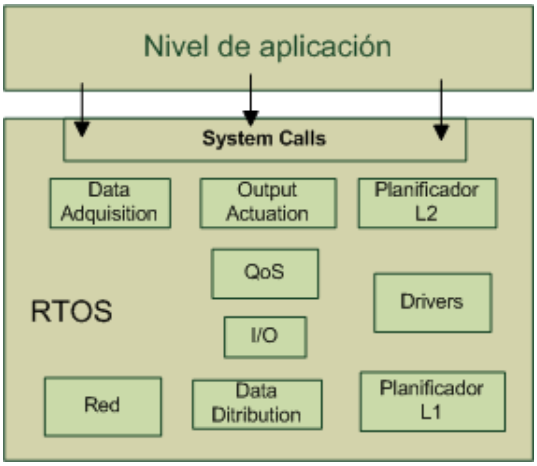

Figura 2.2: Kernel de control como parte del sistema operativo

2.2.4. Modelo de Kernel de Control

Al menos en la etapa inicial de implementación, la opción del *kernel de control* como parte del SO tiene varios inconvenientes debido al gran esfuerzo en la determinación de las modificaciones más apropiadas en el SO, teniendo en cuenta la sensibilidad de los cambios a este nivel. Además, la modificación del código nativo del SO reducirá la portabilidad del *kernel de control* entre sistemas empotrados, debido a que muchas veces se tendrán que hacer re-compilaciones completas del SO en caso de cambios.

Sin embargo, el kernel como un *middleware* específico de control permite un diseño más rápido y un desarrollo más transparente con respecto al SO, debido a que puede implementarse como una librería dinámica que se enlaza con la aplicación evitando parcial o completamente la modificación del código original del sistema operativo, ofreciendo igualmente una separación entre la operación del SO y la aplicación de control.

Considerando estas razones, y algunas otras que serán expuestas más adelante, se propone basar el diseño e implementación del *kernel de control* con la tecnología *middleware* para el desarrollo de aplicaciones de control de tiempo real estrictas o no.

2.3. Middleware de Control

Teniendo como base las especificaciones realizadas para el *kernel de control*, en esta sección determinaremos los requisitos, las funcionalidades y los componentes necesarios en un marco de desarrollo *middleware*. A este tipo de implementación le denominaremos *Middleware de Kernel de Control* (CKM)[4] , y que definiremos como capa de abstracción software que media en las interacciones entre componentes software de control o entre componentes software y su entorno físico, siguiendo el concepto de sistemas *cyber-físicos* en red expuesto en (Lee, 2006), proporcionando servicios relacionados con la ejecución de tareas de control de tiempo real, acceso a sensores y actuadores, y gestión de comunicaciones, que facilitan el desarrollo de aplicaciones tanto a nivel local como distribuido.

Previamente revisaremos algunos trabajos relacionados con el desarrollo *middleware* enfocados a sistemas de tiempo real, para después abordar las características que definen el CKM propuesto.

[3] Application Programming Interface–Interfaz de programación de Aplicaciones
[4] De sus siglas en ingles ***C**ontrol **K**ernel **M**iddleware*.

2.3.1. Trabajos relacionados

La tecnología *middleware* es ampliamente utilizada en sistemas distribuidos como fórmula para proveer servicios de comunicaciones, ejemplos de esto son: *Java-RMI*, *Microsoft's COM* y *CORBA*, y ejemplos dirigidos a sistemas de tiempo real y sistemas empotrados son: *RT-CORBA* y *Embedded CORBA*. A continuación describiremos brevemente algunos de los trabajos orientados a tiempo real:

- ControlWare (Zhang et al., 2002) es una arquitectura de control con QoS (calidad de servicio) diseñada originalmente para ayudar a los programadores a aplicar la teoría de control en el desarrollo de software de control. Esta solución permite a los usuarios especificar niveles de QoS fuera de línea y relacionar esas específicaciones de QoS a lazos apropiados de control realimentados para conseguir la QoS deseada. La abstracción básica en ControlWare es el "componente". Estos componentes son interconectados por medio de puertos de entrada y salida definidos en estos. Define además dos tipos de sensor y actuador software: activo y pasivo. El sensor o actuador pasivo es simplemente una función que retorna un valor de sensor o ejecuta una actuación cuando es invocado por un controlador. Por otro lado, un componente activo es una tarea que normalmente se ejecuta periódicamente para llevar a cabo tareas de sensorización y actuación.

- CAMRIT (Wang et al., 2004) es un middleware adaptativo para la transmisión de imágenes en sistemas de tiempo real empotrados y distribuidos. En CARMIT un lazo de control retroalimentado consigue que la transmisión de las imágenes cumplan sus plazos mediante el ajuste dinámico de la calidad de fragmentos de imágenes. Además, la teoría de control es aplicada al diseño de algoritmos de control para garantizar analíticamente la estabilidad y el rendimiento del sistema aun bajo incertidumbres en la red.

- Etherware (Baliga et al., 2004) es un middleware con paso de mensajes para lazos de control en red. Una de las características a destacar de esta propuesta es la introducción del concepto de servicio de continuidad, que consiste en la capacidad para mantener el canal de comunicación durante la restauración y actualización de un nodo y durante la recuperación de situaciones de fallos.

- TAO (Schantz et al., 2006) es una arquitectura middleware para el desarrollo software, el cual implementa los servicios de las especificaciones de CORBA 2.6 (Group, Dec. 2001) y Real-Time CORBA 1.0 (Schmidt & Kuhns, June 2000), basado en "patrones de programación" de red descritos en (Schmidt et al., 2000) y construidos sobre el *framework* ACE (*Adaptive Communication Enviroment*(Schmidt, Dec. 1993 and June 1994). La incorporación de APIs que implementan el estandar Real-Time CORBA permite cumplir los requerimientos de tiempo real de aplicaciones con prioridad fija mediante la consideración y propagación de prioridades de tareas, evitando inversiones de prioridades ilimitadas, y permitiendo a las aplicaciones configurar y controlar el procesador, las comunicaciones y los recursos de memoria. Uno de los própositos de TAO es proporcionar calidad de servicio (QoS) extremo a extremo.

Muy pocas de estas soluciones han sido desarrolladas específicamente para propositos de control de sistemas físicos usando sistemas de control empotrados interconectados a través de redes de comunicaciones, siendo este uno de los tópicos a considerar en este trabajo. Desde el punto de vista funcional, no hay un consenso sobre si los elementos middleware deben ser activos o pasivos, lo cual tiene sus respectivas ventajas y desventajas. Además, en un marco de desarrollo empotrado en donde se tienen nodos con diversas cantidades de recursos hardware, es necesario establecer diversas clases de servicios middleware a ser implementados en función de la restricción de los recursos de cada sistema de cómputo en el que se instale.

2.3.2. Requerimientos del sistema Middleware

Para el desarrollo del CKM estableceremos algunos requerimientos mínimos a ser considerados en la etapa de diseño y implementación:

- Los retardos en las tareas de control dentro de CKM deberán ser tenidos en cuenta en los bucles completos de control, los cuales serán debidos a que algunos de los recursos deberán ser compartidos dinámicamente entre diferentes actividades de cómputo, y por tanto, la temporización de las tareas no podrá ser determinada con exactitud.
- La utilización de la CPU deberá ser monitorizada y regulada mediante acciones que permitan su optimización.
- Cuando CKM se ejecute en entornos autónomos, el sistema deberá adaptar sus actividades a la energía disponible. La monitorización y el control de la cantidad de energía en las baterías puede provocar cambios en los objetivos y en la estructura de control de todo el sistema.
- El CKM debe asegurar una operación segura y fiable de todo el sistema de control de tiempo real.
- Para permitir un comportamiento autónomo, la detección y el aislamiento de fallos, así como la capacidad de reconfiguración del sistema, deben ser incluidos en el *middleware*.
- La actualización y distribución de la información dentro del CKM debe ser minimizada.
- Deberá considerarse la inclusión de mecanismos que ofrezcan capacidades de distribución y actualización de componentes software dentro del CKM.
- Cuando se trabaja en entornos dinámicos, los objetivos de control podrán cambiar y con ellos los algoritmos de control a utilizar en cada nuevo escenario, siendo por tanto necesaria la consideración de conmutación de tareas y cambios de modo dentro del *middleware*, tomando las precauciones necesarias para garantizar la estabilidad del sistema.
- La prioridad, las señales disponibles, la cantidad de memoria y el tiempo de asignación de recursos para las actividades de control, podrán cambiar dependiendo de las condiciones de trabajo.
- Los cambios en el entorno y la disponibilidad de recursos deberán ser controlados por un nivel de supervisión y resolución que involucre decisiones tales como la selección de las tareas de control más apropiadas en función del estado y el modo de operación actual del sistema, así como la detección y resolución de fallos.
- La disponibilidad o la precisión en la entrega de datos muchas veces no puede garantizarse a lo largo de toda la operación del sistema, lo cual puede estar condicionado a la disponibilidad de recursos o a las condiciones del entorno (ejemplo: fallos en la red debido a cortes en las líneas de comunicaciones), por tanto se debe disponer de mecanismos para manejar estas situaciones.
- Aunque la mayoría de los algoritmos de control son manejados por tiempo mediante acciones separadas por intervalos constantes de tiempo, en entornos con cierta incertidumbre muchas veces es necesario que algunas actividades sean ejecutadas como respuesta a eventos externos que pueden ocurrir aleatoriamente, por lo que es necesario disponer también de mecanismos para controlar eventos asíncronos.

2.3.3. Filosofía de diseño

El diseño que se propone para la arquitectura del *Middleware del Kernel de Control* (CKM) se enmarcará principalmente en entornos distribuidos de control empotrados y de tiempo real, por lo que es necesario partir del hecho que se demandarán mayores requisitos en términos de precisión y previsibilidad temporal, gestión de recursos compartidos y seguridad.

La unidad básica utilizada para la especificación del CKM será el "*componente*". De esta forma, la arquitectura middleware estará compuesta por *componentes software* que en conjunto establecerán los servicios básicos del CKM. Estos componentes serán elementos activos que se ejecutarán periódicamente o en respuesta a eventos.

La estructura general del CKM estará formada por tres niveles principales de dominio de trabajo (ver figura2.3):

- ***Nivel de aplicación de control***: Hace referencia al contexto de trabajo del desarrollador final de software de control. En este nivel, el desarrollador establece los parámetros y configuraciones de las aplicaciones de control que serán ejecutadas en el middleware, también conocido como nivel de usuario.
- ***Nivel Middleware***: Representa el entorno del kernel de control en sí. Reúne los elementos y las interfaces que definen los servicios del CKM. La aplicación de control accede a estos servicios a través de interfaces de usuario.
- ***Nivel Operativo***: Constituye el conjunto de interfaces y elementos que le facilitan el acceso y el control de los recursos hardware al sistema middleware de control. Este nivel puede hacer referencia a un sistema operativo de tiempo real completo o a un conjunto de API's de control hardware.

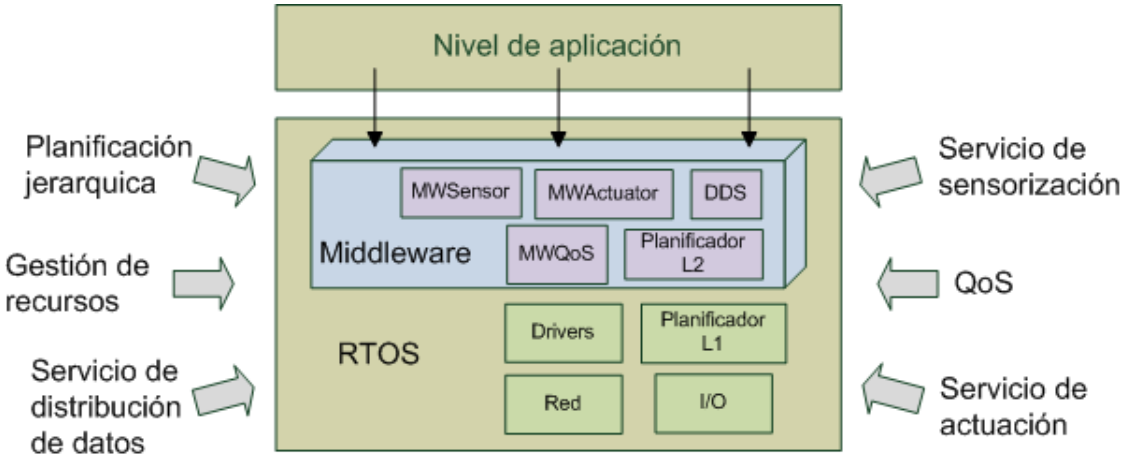

Figura 2.3: Estructura general del Middleware del Kernel de Control

El modelo de programación del *Middleware de Kernel de Control* (CKM), desde el punto de vista de usuario, sigue el concepto de ***delegación de código*** que consiste en transferir desde el nivel de aplicación a el CKM, el código de control que se quiere ejecutar. Una vez delegado el código en el CKM, este se encargará de proporcionarle los recursos computacionales necesarios para su ejecución (ver figura 2.4). Este modelo también le ofrecerá al CKM la posibilidad de decidir acerca de la ubicación física de cómputo que más le convenga al código en función de la disponibilidad de recursos y del espacio de comunicaciones necesario en su ejecución.

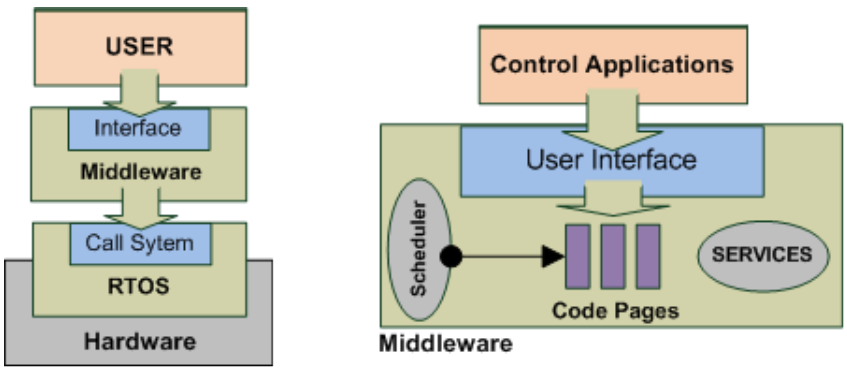

Figura 2.4: Modelo de programación del Middleware de Kernel de Control

Partiendo de las restricciones del entorno al que esta dirigido el middleware, la gestión de recursos tendrá una gran relevancia en el funcionamiento del CKM. En donde una CPU, con su propia memoria y fuente de energía, deberá controlar diversas variables que en muchos casos deberán ser compartidas a través de recursos de red, con otras CPUs que dependen de estas, creándose por tanto dependencias extremo a extremo, en donde el rendimiento de una CPU y el tráfico en la red de comunicaciones podrá afectar el cumplimiento global de los plazos de distribución de datos y con ello, el funcionamiento de los bucles de control del sistema.

Por tales motivos, unos de los objetivos fundamentales dentro del *Middleware de Kernel de Control* será la preservación de un compromiso entre el uso de recursos y el cumplimiento de las restricciones temporales de control (ver figura 2.5), que consistirá en una gestión óptima de memoria, procesador, red y energía basada en la garantización del cumplimiento de los plazos de ejecución de los componentes de control del sistema y en la distribución oportuna de la información de control con el fin de mantener dentro de determinados niveles de *performance* los procesos controlados por el sistema.

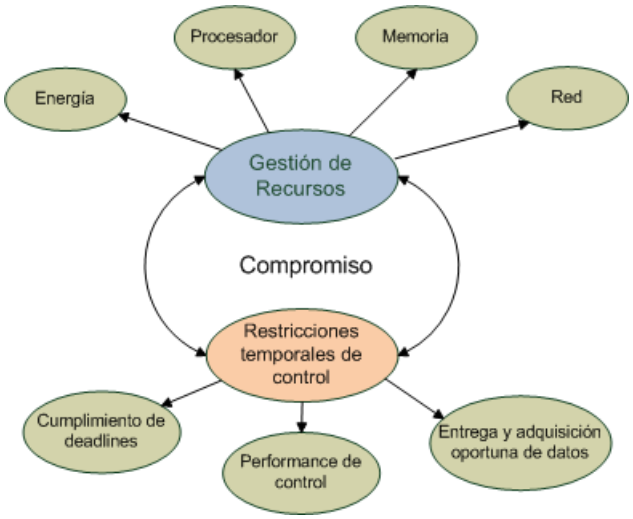

Figura 2.5: Objetivo fundamental en el Middleware de Kernel de Control: mantener un compromiso entre el uso de recursos y el cumplimiento de las restriciones temporales de control.

Teniendo en cuenta lo anterior, el CKM se diseñará basado en *comportamientos adaptativos* para la gestión de recursos compartidos apoyado en el uso de técnicas de *Calidad de Servicio* (QoS) mediante el establecimiento de niveles de calidad que permiten lograr una mayor utilización de los recursos y mejoran a su vez, el rendimiento y la flexibilidad del sistema.

Este comportamiento adaptativo del CKM en la asignación de recursos, puede involucrar cambios en tiempo de ejecución de los parámetros de los componentes software tales como período y plazos de ejecución, que desde el punto de vista de control resultará en un incremento/decremento de los períodos de muestreo o en una simplificación del cómputo de controladores y en una mejora o degradación del funcionamiento de los bucles de control. Todo lo anterior, conllevará adicionalmente a la implementación de cambios de modo de operación, conmutación de controladores y a la delegación/movimiento de controladores entre nodos en condiciones de escasez de recursos disponibles locales. Para tales situaciones, el CKM incorporará mecanismos que permitan llevar a cabo estas acciones con seguridad y preservando la estabilidad del sistema.

Adicionalmente, el uso de tareas opcionales permitirá una reducción del tiempo de cómputo de los controladores por la eliminación de las partes menos relevantes en caso de una reducción en la asignación de recursos, lo que supondrá una disminución en la degradación del rendimiento del sistema controlado. Así mismo, las tareas opcionales también podrán ser utilizadas para obtener soluciones dependiendo del tiempo disponible o lo que es lo mismo de los recursos disponibles, mediante un proceso de mejoramiento de la respuesta utilizando métodos tales como algoritmos anytime, computación imprecisa, métodos múltiples y planificación por refinamiento progresivo (ver capítulo 1), los cuales se caracterizan porque siempre generan un resultado con una calidad de respuesta que se in-

crementa con el tiempo, de tal forma que cuando se agotan los recursos asignados se garantizará el empleo de la respuesta con la mayor calidad posible.

En base a esto, el modelo de tareas de control del CKM, anteriormente propuesto, tendrá un particionado adicional correspondiente a la parte opcional del bucle de control, utilizado para el refinamiento de la señal de control producto de la ejecución de un controlador básico pero suficiente para mantener la estabilidad del sistema. El controlador básico y el tiempo de ejecución de la parte opcional se corresponderán con un factor de calidad de servicio. El nivel de calidad a conseguir dependerá de los recursos negociados por la aplicación, o lo que es lo mismo, de los recursos reservados, y de los recursos disponibles en cada instante. En la figura 2.6 se muestra la relación entre las tareas del modelo de control del CKM con las actividades opcionales. Este arreglo puede ser utilizado en Modelos de control predictivo, control inteligente o en controladores basados en algoritmos de visión.

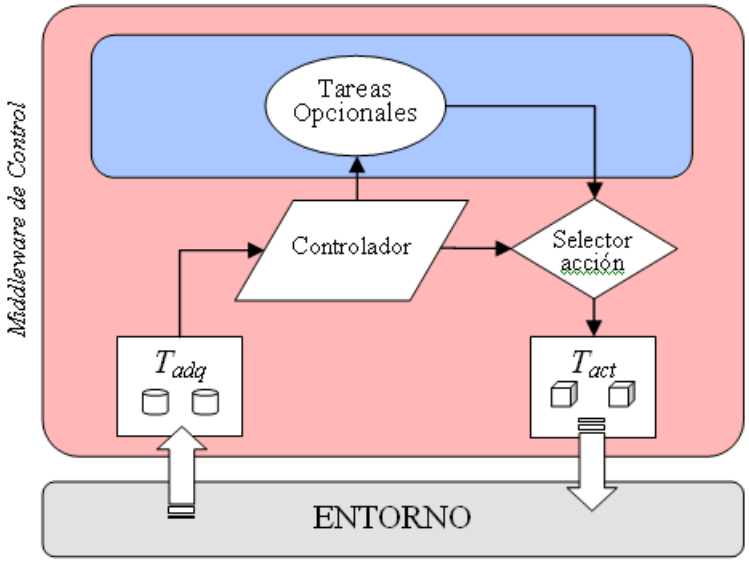

Figura 2.6: Modelo de tareas opcionales para el Middleware de Kernel de Control

Siguiendo el modelo de programación propuesto para el CKM, tanto el código de control como el de las tareas opcionales será delegado al middleware el cual se encargará de coordinar su ejecución mediante una planificación multinivel utilizando esquemas de planificación diferentes tanto para la parte opcional como para la parte básica del modelo de control.

2.3.4. Funcionalidades del sistema middleware

A continuación enumeraremos las principales funcionalidades propuestas a ser ofrecidas por el middleware de control propuesto:

- Adquisición de datos. Las medidas de sensores serán adquiridas según los períodos de tiempo requeridos por la aplicación y los apropiados para cada sensor. Los retardos en la adquisición de datos no deberán reflejarse como retardos en el sistema, es decir, el instante en que se capturan los datos no debe afectar el inicio en que se empieza el cómputo del controlador ni el instante en que se entregan las acciones de control.
- Se empleará una gestión de recursos adaptativa utilizando técnicas de calidad de servicio. Se establecerán niveles de calidad de servicio de ejecución para las aplicaciones de control mediante una negociación previa con el *middleware*.
- Las tareas serán planificadas de forma óptima basados en los objetivos actuales de control, los niveles de calidad contratados y la cantidad de recursos disponibles. Así mismo, se considerará en determinados casos el uso de tareas opcionales que consistirá en la división de algoritmos de

control en partes obligatorias y opcionales, siendo esta última parte llevada a cabo en función del excedente de recursos.

- Al margen y como complemento a las actividades opcionales, se configurarán o delegarán en el CKM varias versiones de algoritmos de control (es decir, la parte obligatoria) para un mismo proceso pero con diferentes tiempos de cómputo y por tanto, asociado a distintos niveles de calidad, de tal modo que puedan ser utilizados en función de la disponibilidad de recursos en cada situación. Para esto, se utilizarán modelos de orden reducido ya que la complejidad del controlador está directamente relacionado con la complejidad del modelo de la planta usado en el diseño.

- Adicionalmente a la delegación de diversos algoritmos de control con diferentes tiempos de cómputo, también se delegarán diferentes versiones de este código para ser ejecutado en múltiples arquitecturas hardware, de tal modo que se puedan formar *paquetes binarios universales* que permitan mover libremente el código de controladores dentro del sistema distribuido.

- Cambios en los parámetros de las tareas. Es bien sabido que en general la calidad de control disminuye si el período de muestro se incrementa, pero también es sabido que a menor período de muestreo mayores serán los recursos requeridos. Por consiguiente, si la disponibilidad de datos sensoriales cambia, el período de la tarea de control también debería cambiar por optimización, lo cual implicaría cambios en los parámetros del controlador y de la información guardada.

- Degradado y estrategias de control seguro. En casos de emergencia se considerará la degradación del comportamiento del sistema o la aplicación de acciones de control grabadas anteriormente (back-up) o la ejecución de determinadas acciones seguras (desconectar, llevar al sistema a un determinado nivel, etc.) previamente establecidas. Junto a la configuración y delegación del código de control en el CKM es necesario definir las actividades seguras para los procesos a controlar.

- Se deberá asegurar que las acciones de control del sistema sean entregadas a tiempo, aun si estas no son el resultado del cálculo de controladores sino en su lugar un *backup* de acciones de control o acciones de control seguras calculadas con datos de sensores previos.

- Distribución óptima del cómputo. Basándose en la delegación de código, los reguladores enviados al middleware se repartirán en el sistema distribuido en función de los requisitos de calidad de servicio acordados. Los reguladores enviados al middleware podrán delegarse en tiempo de ejecución entre los nodos del sistema distribuido para ser usados en diferentes situaciones.

- El middleware proporcionará transparencia en el acceso a datos distribuidos y en las comunicaciones entre aplicaciones de control, y entre estas y dispositivos externos (sensores y actuadores), de tal modo que se puedan invocar componentes distribuidos sin importar su localización.

- Se utilizará un monitor para analizar el *performance* de los bucles de control del sistema con el objeto de cumplir los niveles de calidad contratados con la aplicación, tomando para ello las decisiones oportunas tales como conmutación de controladores, cambio de modo, desconexión, etc. Y para ello se monitorizará igualmente la utilización de los recursos del sistema.

2.4. Modelos de Middleware de Control

Recordando que el *middleware de kernel de control* se enmarca en un marco de desarrollo empotrado en los que existe una gran variedad de sistemas empotrados heterogéneos con diversos *perfomance* de cómputo, es necesario establecer diversos niveles de servicios middleware a ser implementados en función de la restricción de recursos provista por cada sistema. En este sentido, se propondrán dos modelos de CKM compuestos por diversos grupos de servicios: *Tiny-Middleware* y *Full-Middleware*.

Las interfaces y relaciones de los modelos propuestos y referenciados en esta sección están recopilados en el apéndice A.

2.4.1. Full-Middleware

La versión más completa del middleware la denominaremos Full-Middleware (FCKM). Esta versión implementa las funcionalidades completas del middleware de control ofreciendo interfaces y mecanismos para la gestión de bucles complejos de control. Gestiona, organiza e interconecta toda la red distribuida de control. *El nivel operativo* esta compuesto por un sistema operativo de tiempo real.

Este modelo de midleware se utiliza en nodos con procesadores empotrados potentes con sistemas operativos completos de tiempo real y con capacidades altas de E/S y red.

2.4.2. Tiny-Middleware

La versión reducida del Full-Middleware la denominaremos Tiny-Middleware (TCKM). Este middleware se comunica con el FCKM mediante interfaces básicas para la gestión de sensores, actuadores y código de controladores. Además, incluye interfaces de red para la interconexión a sensores y actuadores, y para la descarga de código de controladores desde FCKM. *El nivel operativo* está formado por un conjunto de API's de control hardware con planificación cíclica.

Este modelo está dirigido a nodos con procesadores con capacidad de cómputo y red limitada, pero con características completas de E/S. Ejemplo: PIC (controlador de interfaz periférico), DSP,etc.

2.5. Componentes FullMiddleware

La versión del FCKM está conformada por un conjunto de componentes tal y como se muestra en la figura 2.7.

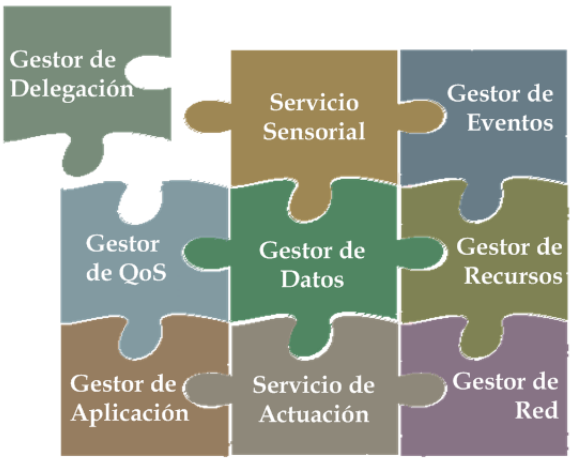

Figura 2.7: Componentes del Full Middleware de Kernel de Control

Estos componentes están configurados con las interfaces mostradas en la figura A.1 y figura A.2.

2.5.1. Gestor de aplicación

Define a una entidad software que ofrece una interfaz inmediata a las aplicaciones de control para acceder a los servicios ofrecidos por el CKM. A cada aplicación de control se le asignará un componentes de este tipo. Este componente tendrá el rol principal de gestionar el acceso de la aplicación al entorno middleware. Son funciones de este agente:

- Recibir las páginas de código de control transferidas por la aplicación y deserializarlas, y posteriormente, comenzar su ejecución mediante la inclusión de este en el planificador del sistema.

- Establecer y negociar directamente con la aplicación los factores de calidad de ejecución que se aplicarán al código delegado.
- Crear y coordinar conexiones entre los componentes internos del middleware y las interfaces externas a la aplicación, por ejemplo, la notificación de eventos, la coordinación entre las actividades optativas y las actividades obligatorias de control, entre otras.

2.5.2. Gestor de QoS

QoS (Calidad de servicio) determina el nivel de satisfacción de un servicio, garantizando un cierto grado de rendimiento en la ejecución del servicio. Partiendo de la definición anterior, este componente se encargará de negociar y gestionar que la ejecución de las tareas propias del middleware y las aplicaciones del sistema se ejecuten dentro determinados niveles de rendimiento.

Este componente interactúa con el gestor de recursos y con las aplicaciones de control a través del gestor de aplicación, de tal forma que se pueda asegurar una calidad óptima en la salida del sistema. Las aplicaciones son estructuradas en diferentes niveles de calidad (QL) dependiendo de la estimación de los recursos necesarios en su ejecución. Los niveles de calidad serán discretos y estarán caracterizados por la calidad necesaria en la salida y el grado de cumplimiento necesario, tales como precisión y cumplimientos estrictos de deadlines de ejecución en sistema de control de tiempo real. Entre mayor sea el nivel de calidad requerido mayor será la cantidad de recursos necesarios por la aplicación. Los QL de las aplicaciones pueden cambiar dinámicamente ya sea por petición propia o por decisión del gestor.

Este componente negociará con las aplicaciones el nivel de calidad que tendrá que proporcionarles, o lo que es lo mismo, la cantidad de recursos a proveer. El principal objetivo es utilizar los recursos de la forma más eficiente posible y maximizar el rendimiento global del sistema. La negociación depende de factores tales como la disponibilidad de recursos, la importancia de la aplicación: período, plazos y tiempos estrictos de ejecución, entre otros.

La interacción de este componente con el componente gestor de recursos es necesaria para asegurar que la reserva de recursos por parte de las aplicaciones estará garantizada. Además, monitorizará junto al gestor de recursos el comportamiento del sistema y analizará si las aplicaciones están utilizando los recursos solicitados o si hay recursos libres, para en tal caso, reasignar dinámicamente los recursos o negociar nuevos niveles de calidad.

2.5.3. Controlador de delegación

Este componente se encargará de coordinar la delegación de código entre los distintos nodos de cómputo del sistema distribuido. Las decisiones de este componente dependerán del componente de aplicación, del gestor de recursos y del gestor QoS, y más específicamente, de las peticiones propias de las aplicaciones, de la disponibilidad de recursos locales para el cumplimiento de las restricciones temporales de las aplicaciones (factores de comodidad de ejecución), de la distribución física de sensores y/o actuadores, de sobrecargas en redes de comunicaciones, entre otras.

Este componente igualmente se encargará de recibir el código delegado por parte de otro nodo, deserializandolo e instanciándolo dentro del middleware de control (CKM).

2.5.4. Gestor de datos

Este componente se basará en la especificación DDS (Data Distribution Service – Servicio de distribución de datos) para la distribución de datos de forma predecible dentro del sistema distribuido de tiempo real. Este componente utilizará un modelo de intercambio céntrico de datos y más específicamente el modelo DCPS (Data-Centric Publish-Subscribe – Datos céntrico Publicador/Subscriptor), el cual se construye sobre el concepto de espacio de datos global que es accesible por todas las aplicaciones interesadas.

En el entorno middleware, las aplicaciones que desean contribuir con información en el espacio global de datos comunican al componente gestor sus intenciones de ser publicadores, y a sí mismo, las aplicaciones que desean acceder a partes del espacio de datos solicitan al componente su intención de

convertirse en “subscriptores”. Cada vez que un publicador genera un nuevo dato dentro del “espacio global de datos”, el middleware, y más específicamente el componente gestor de datos, propaga la información a todos los subscriptores interesados.

En aplicaciones de control de tiempo real, ejemplos de publicadores son los sensores que generan información acerca del estado de un proceso, y ejemplo de subscriptores/publicadores son los controladores que utilizan la información generada por los sensores para calcular acciones de control que posteriormente son aplicadas al proceso.

Data

Hace referencia al objeto que es generado y consumido respectivamente por aplicaciones publicadoras y subscriptoras dentro del sistema middleware en lo que se conoce como “espacio global de datos”. Cada *Data* tiene asociado diversos subscriptores y es actualizado por aplicaciones publicadoras.

Este objeto de datos contiene meta-información acerca del publicador, de tiempos o instantes de actualización/modificación, tipo de dato y características propias del contenido (ej. unidades de medida, etc.).

2.5.5. Gestor de recursos

Este componente en coordinación con el gestor de QoS , se encarga de administrar el uso de los recursos físicos de los sistema de cómputo tales cómo memoria, cpu, ancho de banda, consumo de energía, entre otros.

2.5.6. Gestor de red

Este componente se encarga de administrar el hardware de red, accediendo a los controladores ofrecidos por el sistema operativo. Gestiona el transporte de mensajes en el sistema distribuido ofreciéndole al middleware servicios de transmisión y recepción de datos a través de diversos buses de comunicaciones.

2.5.7. Gestor de eventos

Este componente informa al nivel de aplicación acerca de los eventos que ocurren dentro del middleware de control. La notificación de estos eventos al nivel de aplicación se hace mediante interfaces preestablecidas por el gestor de aplicación. Las aplicaciones recibirán por defecto notificaciones de eventos propios del middleware tales como pérdida de plazos de ejecución, errores de ejecución, entre otros, y además podrán instalar notificadores de eventos específicos a través de *componentes de escucha*.

2.5.8. Relación de componentes

Los componentes que conforman la versión *full* del *middleware* de control estan relacionados en la forma en que se muestra en la figura A.3.

2.6. Componentes TinyMiddleware

Los servicios necesarios para el control de procesos con restricciones temporales y la incorporación del movimiento de código en sistemas middleware reducidos para sistemas empotrados con recursos limitados es mostrado en la figura 2.8.

Las interfaces de los componentes Tiny-Middleware son mostrados en la figura A.4.

Figura 2.8: Componentes para la versión Tiny-Middleware

2.6.1. Relación entre componentes

Los componentes de la versión reducida del CKM se relacionan entre sí como se muestra en la figura A.5.

2.6.2. Inicialización de las actividades

La inicialización de las actividades del sistema Tiny-Middleware es mostrado en la figura A.6.

2.6.3. Enlace con componentes de control

En la figura A.7 se muestra la secuencia necesaria para enlazar o contratar los sensores, actuadores o controladores gestionados por el Tiny-Middleware. Adicionalmente, se describe los pasos para descargar código en la versión reducida del sistema middleware de control.

2.7. Delegación y movimiento de código dentro del sistema Middleware de Control

La tecnología de código móvil unida a la de agentes móviles ha despertado en los últimos años un gran interés en la comunidad investigadora. Pero, su aplicación en entornos de tiempo real con el objeto de aumentar el performance y la flexibilidad en la ejecución de sistemas tales como el Control Distribuido de Procesos es relativamente reciente (ver figura 2.9).

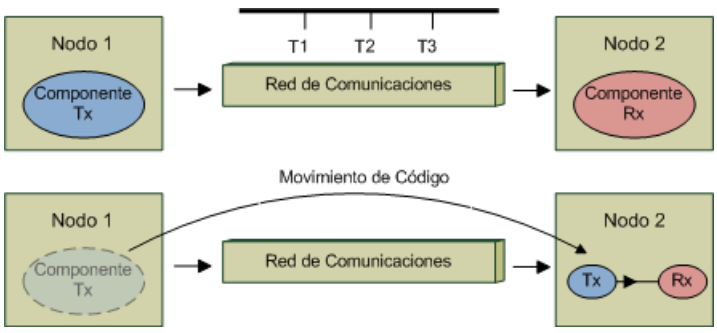

Figura 2.9: Movimiento de código

Se definirán tres tipos de movimiento que serán utilizados en el middleware de kernel de control:

- *Delegación:* Movimiento de “páginas de código” a ubicaciones remotas en donde se llevará a cabo su ejecución.
- *Migración:* Capacidad para cambiar dinámicamente la ubicación de ejecución de determinados “fragmentos de código”.
- *Clonación:* Duplicación de la ejecución de determinadas “páginas de código” en diferentes ubicaciones.

La inclusión del movimiento de código en sistemas de Control de Tiempo Real trae consigo las siguientes ventajas:

- *Flexibilidad y adaptación dinámica a cambios en el entorno*. Habilidad para distribuir código entre los distintos nodos de la red en función de configuraciones óptimas para la resolución de problemas determinados. Ejemplo: Cambios de modo.
- *Reducción de la latencia* en mensajes transmitidos a través de redes de comunicaciones. En sistemas distribuidos de control de tiempo real es necesario realizar la entrega de mensajes en plazos estrictos de tiempo, pero cuando las condiciones de tráfico en la red se vuelven adversas es posible que los mensajes sufran serios retardos que degraden el rendimiento del sistema. Para este tipo de situaciones los sistemas móviles ofrecen la posibilidad de mover el código al lugar de generación de datos en lugar de lo contrario, disminuyendo de esta forma los retardos introducidos por comunicaciones.
- *Balance de carga*. En sistemas empotrados de control de tiempo real en donde los recursos computacionales son limitados y las restricciones temporales de ejecución son exigentes, la distribución de componentes software debe llevarse a cabo de forma minuciosa. En este sentido, la movilidad de código ofrece la posibilidad de llevar a cabo redistribuciones de carga de cómputo cuando las condiciones de ejecución varían o se producen cambios de modo en los nodos de la red distribuida (ver figura2.10).

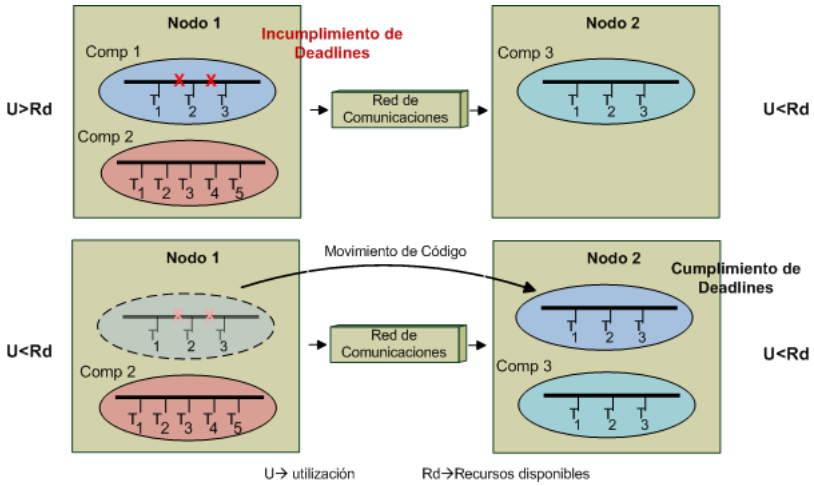

Figura 2.10: Balance de Carga

- Cuando en sistemas distribuidos de control surge la necesidad de actualizar, cambiar o ampliar los componentes del sistema software en tiempo de ejecución tales como controladores, métodos de actuación y adquisición de datos, algoritmos de fusión sensorial, protocolos de comunicación, etc., y ya sea por razones de eficiencia, seguridad o cambios de modo de trabajo, los sistemas móviles ofrecerán una solución versátil y atractiva en sistemas con restricciones temporales (ver figura2.11).

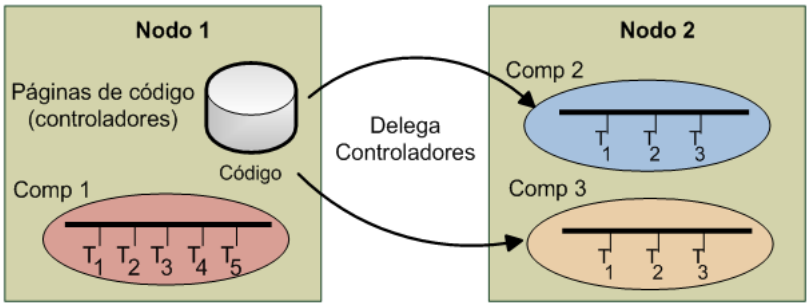

Figura 2.11: Delegación de código

- *Robustez y tolerancia a fallos*. En sistemas distribuidos de tiempo real uno de los requisitos de implementación indispensables es la abstracción de errores de tal forma que se pueda garantizar el correcto funcionamiento del sistema ante posibles fallos. En este aspecto, el movimiento de código permite complementar los sistemas tradicionales de tolerancia a fallos debido a que se pueden llevar a cabo redistribuciones dinámicas de componentes software ante eventuales fallos en sistemas de cómputo.

La implementación del movimiento de código mediante el uso de un sistema middleware de control (ver figura2.12 permite potenciar la funcionalidad y eficacia en el intercambio de componentes software, debido a que se pueden implementar de forma más robusta y autónoma mecanismos de control de delegación de componentes con restricciones temporales.

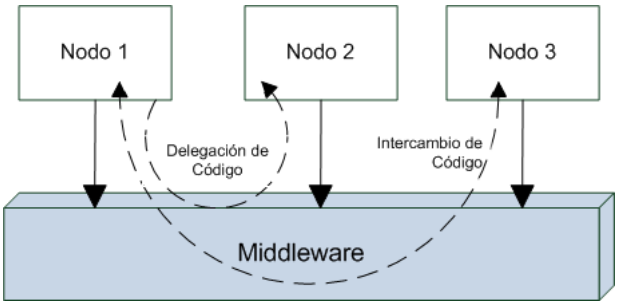

Figura 2.12: Delegación de código con sistemas middleware

El sistema middleware también puede simplificar la interdependencia en el acceso a datos remotos y locales cuando se producen movimientos de componentes, si en estos se implementan sistemas automáticos de distribución de datos (ver figura2.13).

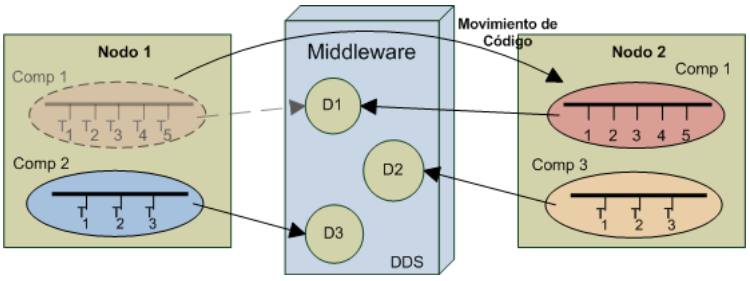

Figura 2.13: Delegación de código con sistemas middleware

En el sistema middleware de control se podrán implementar políticas de calidad de servicio de

forma implícita, que permitirán aplicar y relacionar directamente niveles de QoS con criterios de movilidad. Ejemplos de políticas de QoS:

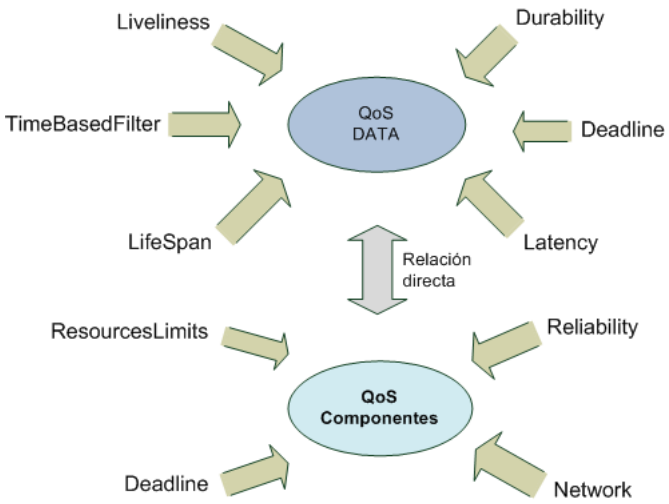

Figura 2.14: Políticas de QoS en Sistemas Middleware de Control

2.7.1. Criterios para la movilidad de código

En las siguientes situaciones el sistema podrá aprovecharse de las capacidades incorporadas por la delegación/movimiento de código:

- Inicio de cambios de modo.
- Actualización de componentes (corrección o mejoramiento de algoritmos).
- Incumplimiento de niveles de QoS contratados (incumplimiento de plazos de ejecución).
- Tolerancia a fallos (daños físicos, copias redundantes, etc).
- Mantener el rendimiento de control de los bucles de control en estados deseados.

2.7.2. Estructura del código móvil

Los componentes móviles están compuestos por meta-información (XML) que ofrece información dinámica y estática relevante para el movimiento de código.

- Estática: descripción del sistema multinivel, arquitectura hardware soportada (ej. arm, i386, etc), S.O., tipo de lenguaje de programación, dependencias, etc.
- Dinámica: Estructura y estado actual del componente, formato binario o interpretado o fuentes, y otros atributos no funcionales como: ancho de banda, tamaño, energía, urgencia o plazos de ejecución, etc.

El código de los algoritmos de control que se delegan al FCKM deben soportar la mayor parte de las arquitectura presentes en el sistema, es decir, el código binario delegado al middleware debe contener varias versiones del mismo código compilado para diferentes arquitecturas formando lo que llamaremos *paquete binario universal*. La información intercambiada entre los nodos que componen el sistema distribuido a través del middleware de control se estructura de la siguiente forma:

Información de capacidades de nodo:

- *Estática*: procesador, arquitectura, direcciones de red, disco, memoria del sistema, dispositivos periféricos a modo de sensor y actuador.
- *Dinámica*: frecuencia de funcionamiento de procesador, utilización promedio de CPU, memoria disponible, disco disponible.

Información de capacidades de sensores y actuadores:

- *Estática*: descripción del componente: tipo de sensor o actuador, descripción de funcionalidad, marca hardware, cantidad de elementos (si es aplicable), frecuencia máxima de muestreo, si es posible variarla, etc.
- *Dinámica*: período de muestreo actual, estado actual del sensor o actuador (activo o desactivo), etc.

Información de las capacidades de componentes software móviles tales como controladores:

- *Estática*: tipo de componente, tipo de programación (C/C++), tipo de código: compilado o interpretado, arquitectura para la que fue compilada, sistema operativo (si es aplicable), período máximo y mínimo (si es posible), *deadline*, tiempo de cómputo normalizado a una frecuencia determinada de reloj. En caso de ser un controlador: tipo de controlador, sensores y actuadores relacionados (id), si es posible variar la frecuencia, etc.
- *Dinámica*: período y plazos de ejecución actuales, tiempo de cómputo, plazos de ejecución perdidos en ejecución, rendimiento de ejecución del controlador, etc.

2.7.3. Ejemplos de delegación de código

Ejemplo de delegación de código entre dos tipos diferentes de middleware para el control de procesos:

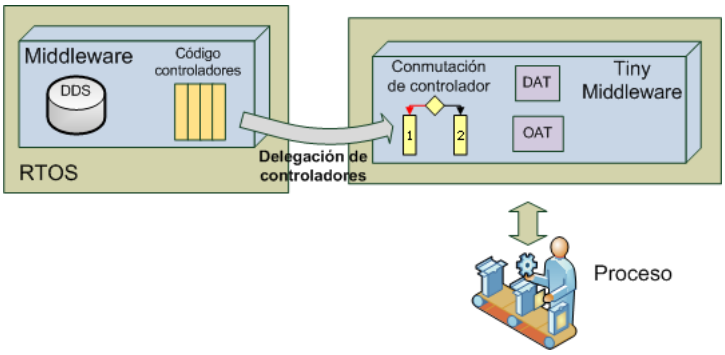

Figura 2.15: Delegación de código entre sistemas FCKM y TCKM

Coordinación en la migración de código entre componentes completos de control:

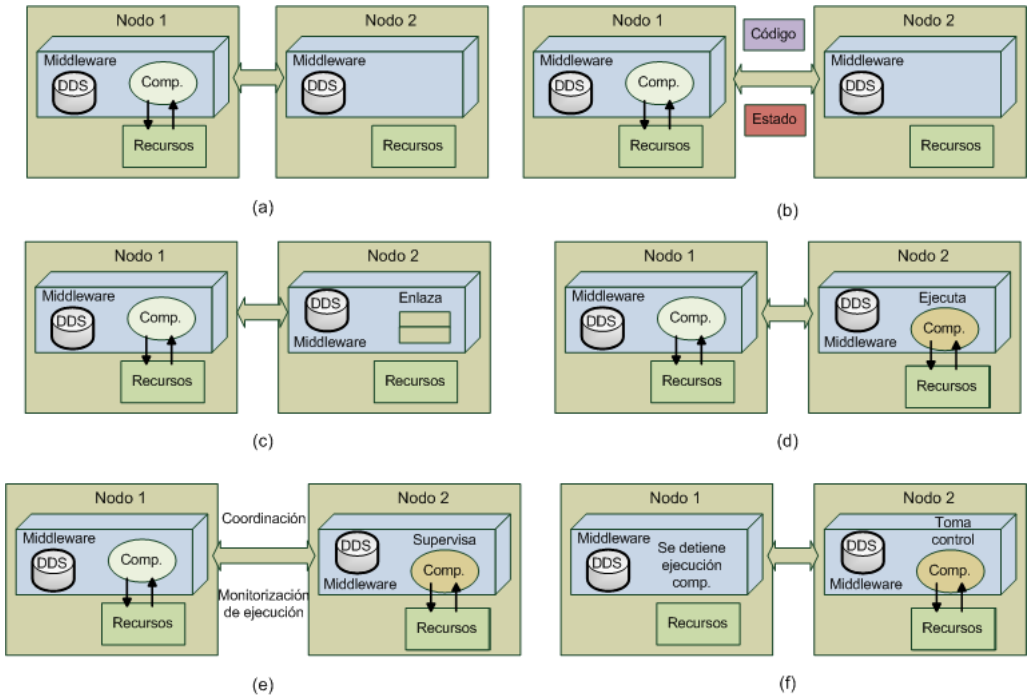

Figura 2.16: Migración de código entre sistemas FCKM

Capítulo 3

ESCALADO DE VOLTAJE/FRECUENCIA - FRECUENCIA ESTÁTICA

3.1. Introducción

El desarrollo de sistemas de cómputo en sectores industriales tales como el ferroviario, aeroespacial y automóvil está basado en Sistemas Empotrados Críticos de Tiempo Real (también conocido por sus siglas en inglés CRTES-Critical Real-Time Embedded Systems). Estos sistemas se enfrentan a nuevas demandas y exigencias relacionadas con el incremento de la fiabilidad, seguridad, mayor inteligencia, conectividad, mejoras del rendimiento, eficiencia en el consumo energético y reducción del volumen, tamaño y coste de los sistemas de cómputo. En los próximos años se espera que la tendencia en el desarrollo de CRTES esté dominada por la irrupción de los sistemas críticos mixtos (conocido también por sus siglas en inglés MCS - Mixed-Criticality Systems).

La arquitectura federada que se utiliza actualmente en estos dominios industriales, en los que cada sistema esta compuesto por sub-sistemas empotrados interconectados entre sí y en donde cada uno de ellos ofrece funcionalidades bien definidas, se está enfrentando a limitaciones futuras de escalabilidad debido a la incorporación continua de nuevas funcionalidades. Un ejemplo claro de esto es el sector del automóvil, donde los coche de alta gama están incorporando una gran cantidad de componentes software que traducido a líneas de código equivale a un total de 20 millones de líneas de código. Este código esta distribuido en 70 ECUs (por sus siglas en inglés Electronic Control Unit) que en conjunto software/hardware representan el 30 % del coste total de producción (Buttle (2012)).

Los MCS están fundamentados en la integración de diversas aplicaciones con diferentes niveles de criticidad tales como fiabilidad, seguridad, tiempo real y no tiempo real, en un mismo sistema físico empotrado. Algunas de las ventajas del uso de MCS son:

- La reducción del peso, consumo energético, costo, tamaño y peso.
- Incremento de la fiabilidad por la reducción de conexiones eléctricas, las cuales suelen ser una de las mayores fuentes de fallos en sistemas CRTES. En el sector de la aviación, los problemas de interconexión son la principal fuente de fallos en el equipamiento eléctrico de los aviones. Por tanto, la reducción del número de ECUs reduce el número de interconexiones eléctricas, conectores y cables.
- Mejora en la escalabilidad del sistema. Los sistemas MCS permiten agregar nuevas funcionalidades de valor agregado sin arriesgar la fiabilidad y reducir el impacto global del consumo energético, tamaño y costo. Sin embargo, la certificación o cualificaciones de estos sistemas son un nuevo reto porque se deberán proveer suficientes evidencias para demostrar que el sistema resultante es seguro para su propósito.

En los últimos años la mayoría de investigaciones en el área de MCS han estado direccionadas en temas relacionados a escalabilidad, certificación por diseño, reconfiguración, arquitectura, detección y aislamiento de fallos y redundancia.

Sin embargo, algunos retos importantes aún no han sido abordados, siendo uno de ellos la gestión y optimización del consumo energético en sistemas con criticidad mixta. El consumo de energía es otro recurso (que al igual que el tiempo y el espacio) que tiene que ser compartido entre las diferentes aplicaciones (criticas o no) y la energía disponible tiene que ser utilizada por todas las aplicaciones en el MCS. La ausencia de métodos de optimización y gestión del consumo energético en MCS puede llevar a una reducción en la disponibilidad del sistema así como a una disminución en el tiempo de vida del producto, lo cual puede afectar su aplicabilidad en la industria o disuadir el uso de la tecnología desarrollada para MCS. Esto es especialmente importante en sectores industriales basados en sistemas autónomos o con energía limitada tales como el espacial, donde la gestión del consumo energético es fundamental.

Para el desarrollo de este capítulo, vamos a considerar un entorno dinámico donde un sistema CRTES conectado a una red de comunicaciones, permite llevar a cabo la migración de tareas y modificar la frecuencia del procesador de forma dinámica. Suponiendo que el sistema distribuido conoce dónde y cuándo asignar las tareas (Biondi & Buttazzo, 2015; Briao et al., 2007; Emberson & Bate, 2007) entre las unidades de cómputo, es necesario realizar un análisis de planificabilidad o factibilidad en cada llegada y partida de tareas para cada sistema empotrado. De tal forma que se garantice que los requisitos temporales del sistema serán cumplidos durante la fase de re-asignación o distribución de tareas. Esto también implica que una nueva velocidad del procesador (escalamiento de frecuencia) deberá ser calculada para permitir la adaptación del sistema a la nuevas condiciones de carga computacional. Y es en este punto en el que el algoritmo propuesto en este capítulo tiene su importancia. Aunque algunos autores han llevado a cabo estas dos fases (análisis de planificabilidad y cálculo del escalado de frecuencia) separadamente (AlEnawy & Aydin, 2005), estos análisis están fuertemente relacionados y en algunos casos pueden ser ejecutados de forma conjunta.

En ese sentido, este trabajo se enfoca principalmente en la **propuesta de un nuevo algoritmo** para ser utilizado en tiempo de ejecución durante las **fases de cálculo de velocidad de procesador y asignación de tareas**, utilizando un esquema de planificación por prioridades fijas con plazos de ejecución (*deadline*) menor y/o igual que el periodo de las tareas cuando existen cargas de trabajo dinámicas en el procesador.

La principal motivación para el uso de algoritmos basados en prioridades fijas está dado por la demanda en el sector industrial, donde varios estándares recomiendan el uso de este tipo de planificadores. Un ejemplo de este tipo de estándares es el ARINC-653 (Interface, 2003), el cual es usado principalmente en la industria de la Aviación y actualmente también en el sector Aeroespacial (Windsor & Hjortnaes, 2009). Este estándar define un sistema temporal y espacialmente particionado, donde cada entorno de ejecución o partición deberá utilizar métodos de planificación basado en prioridades fijas. Un ejemplo del uso de este tipo de planificadores en el sector espacial son los trabajos realizados por el autor en (Galizzi et al., 2011, 2012), utilizando el sistema operativo RTEMS sobre el hipervisor XtratuM. Sin embargo, en este capítulo y el siguiente el planificador EDF basado en prioridades dinámicas es utilizado con el fin de proporcionar una comparativa con las propuestas. Como es bien sabido, el algoritmo EDF proporciona mayores cuotas de rendimiento comparado con la planificación basada en prioridades fijas, pero la principal premisa es cubrir una parte de las exigencias del sector industrial.

Esta propuesta será complementada en el siguiente capítulo con algoritmos dinámicos para maximizar el ahorro de energía global. Este método permitirá reutilizar los instantes ociosos del procesador consiguiendo un mayor ahorro de energía. Estos instantes ociosos son debidos a finalizaciones previas de las tareas y como resultado de características intrínsecas a la planificación.

3.1.1. Trabajos Relacionados

Pocos trabajos cubren la migración de tareas en el contexto de los sistemas empotrados e interconectados por red. Además muchos algoritmos están orientados únicamente a sistemas multiprocesador y con restricciones no estrictas de tiempo real (Yazdi et al., 2008; Emberson & Bate, 2007; Anderson et al., 2005). Algunas de estas publicaciones están relacionadas con la migración de tareas y con la minimización del consumo energético (Briao et al., 2007; Chen, 2005). Briao (Briao et al., 2007) analiza el impacto de la migración de tareas y justifica su uso desde el punto de vista de desempeño y ahorro energético en sistemas de cómputo.

Varias heurísticas han sido propuestas para abordar el problema de asignación de tareas con plazos de ejecución iguales que el período (Mejia-Alvarez et al., 2004; Chen, 2007) utilizando el esquema de planificación EDF (*Earliest-Deadline-First*). En (AlEnawy & Aydin, 2005), AlEnawy y Aydin usan algoritmos para la asignación de tareas de tiempo real utilizando el esquema de planificación RM (*Rate Monotonic*). En este último artículo se presenta una comparación de unos pocos algoritmos en función de parámetros tales como complejidad, factibilidad de planificación y consumo de energía. Sin embargo, se analizan únicamente tareas con plazos de ejecución igual que el período, lo cual puede resultar en una simplificación muy exigente para ser utilizada en *sistemas de control empotrados*, aunque en determinados casos pueden ser asumidos. Además, algunos aspectos tales como predicibilidad de ejecución, comportamiento en función de la llegada o partida de tareas y el costo computacional real de los algoritmos no son tenidos en cuenta.

A nivel de CPU, las técnicas DVS pueden ser clasificadas como *estática* y *dinámica*. Para el cálculo de la frecuencia mínima de procesador en sistemas de tiempo real estricto, los *algoritmos estáticos* utilizan parámetros tales como períodos o tiempos de activación mínimos y suponen que el tiempo de ejecución de cada tareas es el del peor caso. Adicionalmente, esta frecuencia es calculada de forma estática y antes de comenzar la ejecución del sistema. Puesto que muchas veces la diferencia en los tiempos de ejecución del mejor y el peor caso puede ser grande en determinadas aplicaciones, muchos autores han propuesto *algoritmos dinámicos*, los cuales se basan en la reclamación y reutilización en tiempo de ejecución de los tiempos ociosos resultantes de la finalización anticipada de tareas con el fin de reducir al máximo el consumo de energía.

Utilizando algoritmos estáticos se consiguen frecuencias de procesador sencillas que nunca cambian o frecuencias variables pero que han sido estáticamente decididas antes de comenzar la ejecución. Un ejemplo para el primer caso es (Pillai & Shin, 2001), en donde se calcula la frecuencia mínima de procesador para un conjunto de tareas dado, utilizando un esquema de planificación EDF y se propone además un método aproximado para RM. En (Saewong & Rajkumar, 2003) se propone un algoritmo que selecciona una frecuencia constante para un esquema de planificación con prioridades fijas. En (Bini et al., 2005) los autores presentan un método para conseguir una frecuencia promedio mínima de procesador cuando únicamente hay disponibles dos valores discretos de frecuencia, y para ello la velocidad del procesador se conmuta utilizando un método por modulación de ancho de pulso.

Desde el punto de vista de algoritmos con frecuencias variables pero estáticamente decididas, en (Mejia-Alvarez et al., 2004) y (Saewong & Rajkumar, 2003) los autores proponen un método en donde a cada tarea se le asigna una frecuencia diferente. Varios autores han publicado trabajos para lograr un ahorro de energía óptimo, entendiendo por óptimo que no existe otro algoritmo con el que se pueda conseguir un mayor ahorro de energía. Ejemplos de estos algoritmos para tareas periódicas son (Liu & Mok, 2003), (Yun & Kim, 2003), y para tareas periódicas y aperiódicas son (Zhong et al., 2007), (Scordino & Lipari, 2006). Adicionalmente, en (Liu & Mok, 2003) y (Gaujal & Navet, 2007) se aborda el problema mediante funciones de velocidad variables con el tiempo y presenta la caracterización de los puntos en el tiempo en donde deben llevarse a cabo cambios de frecuencia. Sin embargo, una desventaja de trabajar con frecuencias variables estáticamente establecidas es que si por alguna razón la activación de una tarea se pierde o se retrasa, todo el asignamiento de frecuencias puede resultar afectado, lo cual puede llevar a la perdida de plazos de ejecución.

Algunos autores (Piao et al., 2009) han propuesto algoritmos dinámicos en combinación con método estáticos. Estos algoritmos pueden dividirse en métodos inter-tareas y intra-tareas. En algoritmos inter-tareas, la frecuencia del procesador es determinada tarea por tarea, mientras que con los algoritmos intra-tareas se puede ajustar la frecuencia aun dentro de los límites de una tarea dada. En (Kim et al., 2002) se presenta una comparación del rendimiento de varias técnicas dinámicas. En (Pillai & Shin, 2001) los autores proponen un algoritmo dinámico para métodos de planificación EDF y RM. (Saewong & Rajkumar, 2003), (Quan, 2004) y (Leung, 2005) proponen algoritmos dinámico para planificadores por prioridades fijas. En (Aydin et al., 2004) se presenta un algoritmo genérico de reclamación dinámica de instantes ociosos, y un mecanismo adaptativo y especulativo para ajustar la frecuencia de procesador. En (Piao et al., 2009) se investiga sobre cambios de frecuencia dinámicos para tareas periódicas y aperiódicas.

Por otro lado, en (Marinoni & Buttazzo, 2007) se propone el uso de planificadores elásticos para mejorar la gestión DVS y en (Zhu et al., 2004), (Xia, 2008) la gestión de energía se basa en métodos de planificación por control realimentado (*feedback control scheduling*). El análisis de ahorro energético

en procesadores con capacidades DVS ha sido extendido por otros autores para su uso en la reducción de consumo a nivel de memoria (Cho & Chang, 2006)(Liang et al., 2008) y a nivel de red de comunicaciones (Yi et al., 2009)(Kumar et al., 2008).

Sin embargo, algunos trabajos mencionados con anterioridad están basados en el análisis del hiperperíodo, lo cual puede resultar en un cálculo con un coste computacional demasiado alto e inapropiado para ser utilizado en tiempo de ejecución cuando se lleva a cabo una migración de código. Además, algunos autores consideran únicamente plazos de ejecución igual al período, lo cual puede resultar en una simplificación demasiado estricta para ser utilizada en sistemas de control empotrados, aunque en casos específicos podría llegar a considerarse. Adicionalmente, algunos estudios utilizan heurísticas (Jejurikar & Gupta, 2004; Mejia-Alvarez et al., 2002) para hallar la frecuencia mínima, aunque algunas veces lejos de la solución óptima o mínima.

3.1.2. Contribuciones y organización del capítulo

El consumo de energía global del sistema puede ser utilizado o puede contribuir como criterio en el balance de cargas de procesamiento durante la fase de diseño y desarrollo de código. Este criterio puede ser utilizado combinado con otros criterios tales como planificabilidad, retardos de comunicación, estabilidad desde el punto de vista de aplicaciones de control, y para decidir sobre el movimiento dinámico de código y sobre el balance de cargas de trabajo en tiempo de ejecución. Sin embargo, nos centraremos en el problema de asignación de tareas, y más exactamente en el análisis de planificabilidad que debe realizarse en cada llegada o partida de tareas, y en la gestión de energía en los sistemas de cómputo.

En este capítulo, se presenta un nuevo algoritmo para el análisis de planificabilidad y para el cálculo de la frecuencia de procesador que minimiza el consumo energético del procesador. A través de amplias simulaciones llevadas a cabo en el capítulo 5, evaluaremos el rendimiento del método propuesto comparado con otros métodos de análisis de planificabilidad existentes, adaptados al cálculo de la frecuencia mínima de procesador. Esta frecuencia mínima será calculada en términos de consumo de energía, grado de aceptabilidad y coste de computación real, y al mismo tiempo se analizará la previsibilidad en la ejecución y el comportamiento de todos los algoritmos en relación con la llegada continua de tareas. Esta propuesta consiste en un *algoritmo estático* DVFS que halla una frecuencia de procesador mínima y constante para esquemas de planificación por prioridades fijas.

3.2. Modelo del sistema

En esta sección se presenta un conjunto de modelos para la definición del sistema, que serán utilizados como referencia en la enunciación de la propuesta: un modelo del computo de tareas, el modelo de procesador desde el punto de vista del consumo energético y los costes computacionales para la evaluación de los algoritmos.

3.2.1. Modelo de tareas

En este capítulo utilizaremos un modelo de tareas periódico. Supondremos un conjunto $\mathcal{T} = \{\tau_1, \tau_2, ..., \tau_n\}$ de *n* tareas expulsivas de tiempo real ejecutándose sobre un sistema uni-procesador, en donde cada tarea $\tau_i(T_i, D_i, C_i)$ está caracterizada por un período T_i, un plazo máximo de ejecución relativa D_i y un tiempo de ejecución para el peor caso C_i. Además supondremos instantes críticos de activación para las tareas, es decir desfaces iniciales de activación, iguales a 0. La planificación de las tareas se llevará a cabo utilizando un planificador por prioridades fijas (RM o DM) y estará ordenado de mayor a menor prioridad, siendo τ_1 la tarea de mayor prioridad y τ_n la tarea de menor prioridad. El conjunto de períodos de todas las tareas estará representado por la letra mayúscula y en negrita $\mathbf{T} = \{T_1, T_2, ..., T_n\}$, al igual que el tiempo de ejecución del peor caso para todas las tareas $\mathbf{C} = \{C_1, C_2, ..., C_n\}$.

Consideraremos que en la CPU se ejecutarán únicamente tareas de tiempo real, siendo esto no una restricción sino una simplificación para llevar a cabo el análisis.

3.2.2. Modelo del procesador

El consumo de potencia ($\mathcal{P}$) por ciclo de CPU es proporcional a $C_L.V_{dd}^2.f$, donde C_L es índice de capacitancia promedio de carga, V_{dd} es el voltaje de alimentación del procesador y f es la frecuencia de operación del procesador. La frecuencia de operación máxima depende directamente del voltaje de alimentación dado por $f = K\frac{(V_{dd}-Vth)^\sigma}{Vdd}$ donde K es una constante específica de la tecnología utilizada, Vth es el voltaje de *threshold*(Liu & Mok, 2003) y σ es el índice de saturación, donde $1 \leq \sigma \leq 2$ (Saewong & Rajkumar, 2003; Pouwelse et al., 2001).

Sin embargo, la relación exacta potencia/frecuencia depende específicamente de la tecnología del procesador y del hardware que lo acompaña, cuya función puede expresarse en términos de velocidad de CPU como una función polinomial de tercer o segundo grado (Li & Ding, 2001)(Aydin et al., 2004)(Zhong et al., 2007)(Marinoni & Buttazzo, 2007).

En este capítulo la curva de la función $\mathcal{P}$ (ver figura 3.1) es obtenida de una relación experimental entre el consumo de potencia en estado activo y ocioso del procesador respecto a la frecuencia del procesador Intel XScale (Zhong et al., 2007). Esta curva se utilizará como referencia en el análisis comparativo entre los diferentes algoritmos para el cálculo de la frecuencia óptima de operación del procesador que maximice el ahorro de energía.

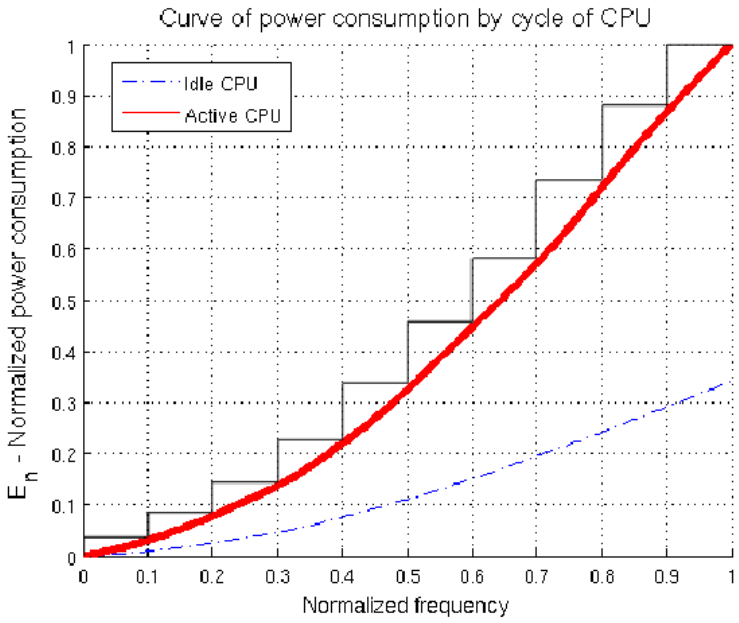

Figura 3.1: Modelo normalizado del consumo de potencia por ciclo de CPU del procesador Crusoe Transmeta

Considerando el hecho de que la mayoría de los procesadores comerciales disponibles únicamente proveen un número limitado de niveles de voltaje, en la figura 3.1 un modelo discreto ha sido trazado sobre el modelo continuo del procesador. Este modelo discreto está formado por 10 niveles de consumo de potencia, y cada nivel corresponde a una frecuencia de operación a la que el procesador puede ser escalado. De esta forma el modelo continuo puede ser mapeado al modelo discreto.

Para el análisis del comportamiento de varios algoritmos DVFS (Dynamic Voltage and Frecuency Scaling – Escalado dinámico de frecuencia y voltaje) en el apartado 5.2, se supondrá que el voltaje de alimentación del procesador puede ser cambiado continuamente, y por lo tanto, también la frecuencia. Si la frecuencia obtenida usando los métodos a evaluar está entre dos niveles discretos, la frecuencia a utilizar será la frecuencia más alta. De esta forma se garantiza que los requisitos temporales del sistema serán cumplidos. Aunque esta es una forma simple y rápida para extender los algoritmos a un modelo discreto, varios investigadores han propuesto algoritmos para mapear niveles de voltaje continuo a niveles discretos de una forma más óptima. Un ejemplo de estas propuestas es la enunciada

en (Bini et al., 2005).

Por consiguiente, en este documento no se explorará el efecto de números limitados de frecuencias de procesamiento. Sin embargo, gracias a los avances en el diseño de CPUs y en la electrónica de potencia (Duan & Khatri, 2006), varios sistemas pueden operar en amplios espectros de voltaje. Un ejemplo de esto es la evolución en los últimos años de los procesadores Intel que usan una tecnología SpeedStep (Intel, 2004). Esta tecnología tiene tres versiones: SpeedStep, SpeedStep II and SpeedStep III. En la primeras versiones la frecuencia de reloj podía ser escalonada en incrementos de 200Mhz y en la última versión la frecuencia puede variarse en incrementos de 100Mhz, lo cual incrementa el número de niveles discretos disponibles.

También supondremos que la frecuencia de operación será ajustada mediante un factor de escalado normalizado α ($0 < \alpha \leq 1$), lo cual equivale a escalar C por el factor $1/\alpha$ sin afectar el período ni los plazos de ejecución de las tareas. Cuando α es igual a 0, el procesador estará en modo apagado o "sleep" (ver sección 4.3.2 , y cuando α es 1, el procesador estará corriendo a la máxima frecuencia de reloj.

Inicialmente ignoraremos la sobrecarga por la conmutación entre las frecuencias de procesador, la cual generalmente es pequeña (Pillai & Shin, 2001), y consideraremos además que en la CPU sólo se ejecutarán tareas de tiempo real, no siendo esto último una restricción, sino una simplificación para llevar a cabo el análisis.

3.2.3. Evaluación de costes

Para la comparación de diferentes propuestas, algunos autores tales como (Bini & Buttazzo, 2004) y (Bini et al., 2003) han propuesto el análisis de complejidad de métodos de planificabilidad basándose en el número de pasos e iteraciones. Sin embargo, si se realiza esta comparación desde el punto de vista de ciclos de cómputo de ejecución, los resultados podrían verse fuertemente alterados. Por ejemplo, el coste computacional de 100 bucles de sumas no es el mismo que el de 100 bucles de divisiones. Por lo tanto, la complejidad de los algoritmos para calcular un apropiado *alpha* (α) en sistemas empotrados debería evaluarse basado en el número de ciclos máquina requeridos en cada operación matemática, especialmente si se considera la ejecución del algoritmo en tiempo de ejecución. Es bien sabido que en muchos sistemas empotrados sin FPU (unidad de coma flotante) las operaciones con divisiones tienen un alto coste computacional comparado con otras operaciones. Por ejemplo, un método muy utilizado para este tipo de operaciones es el Newton-Raphson, cuyas rutinas pueden tener un coste entre de 20 y 100 ciclos (Sloss et al., 2004) dependiendo de la implementación y del rango de los operandos de entrada. Por consiguiente, aspectos como estos serán considerados en la evaluación comparativa de algoritmos.

Para esto, se prestará especial atención al código ensamblador resultante de la implementación de los algoritmos y se contarán los ciclos máquina necesarios para su ejecución. La cache se deshabilitará de tal forma que se consiga una mayor predecibilidad en la ejecución de los algoritmos. Los costes se darán en ciclos máquina en lugar de unidades de tiempo porque esto permitirá expresar el coste de ejecución en función de diferentes frecuencias de procesador. Por ejemplo, el coste a una frecuencia de 100Mhz es 5 veces menos que cuando se utiliza un procesador a 500Mhz.

3.3. Escalado de frecuencia y consumo energético en sistemas con carga dinámica

En esta sección abordaremos la problemática en la movilidad de componentes de tiempo real entre diversas plataformas suponiendo que en estas se puede realizar una gestión de consumo. Uno de los principales inconvenientes al mover estos componentes entre procesadores es la reconfiguración dinámica de los parámetros que controlan el consumo debido a los cambios en el conjunto de tareas.

El consumo energético ha sido definido como la integral en el tiempo t de $\mathcal{P}$:

$$E(S_y, t) = \int_0^{t_a} \mathcal{P}(F(t, \mathcal{T}(t)))dt$$

donde S_y identifica el tipo de planificación, cuyo sufijo y puede ser FP o DP para planificación por prioridades fijas y por prioridades dinámicas respectivamente. $\mathcal{P}$ es la potencia consumida por ciclo de CPU, la cual depende de la función de frecuencia (F) de CPU. La frecuencia de reloj de procesador F es expresada como una función dependiente del tiempo t y del conjunto de tareas $\mathcal{T}$ activo en cada instante de tiempo t. En la figura 3.2 se muestra un ejemplo del escalado de frecuencia del procesador en función del tiempo y del conjunto de tareas. En esta figura se muestran dos conjuntos diferentes de tareas. Obviamente, el perfil del factor de escalado de frecuencia $\alpha(t)$ deberá cambiar justo cuando el procesador cambie el conjunto de tareas. Esto es debido a que cada α es calculado para un conjunto específico de tareas.

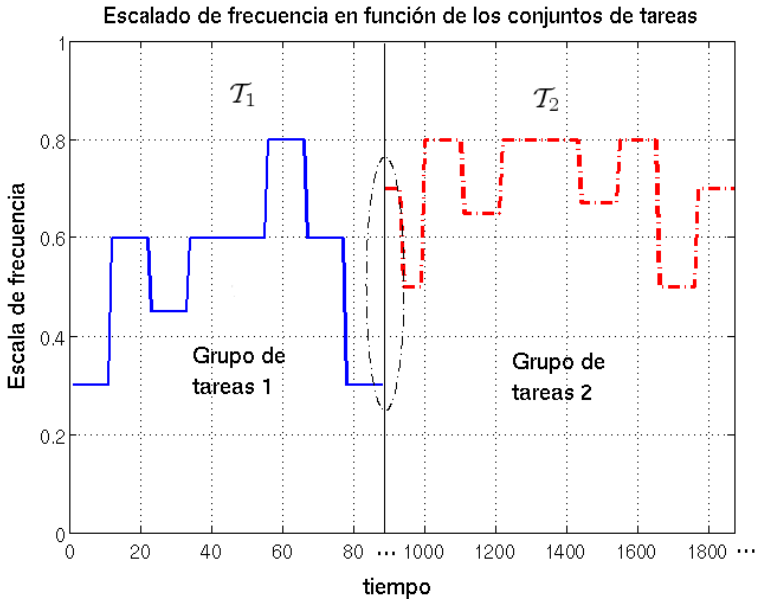

Figura 3.2: Ejemplo de escalados de frecuencia α en función de la variación del conjunto de tareas a lo largo del tiempo t

La conmutación entre dos conjuntos de tareas diferentes es mostrada en la figura 3.2 con una elipse punteada. El conjunto de tareas no es conocido de antemano, debido a que este conjunto es definido dinámicamente de acuerdo a las decisiones tomadas en el proceso de delegación de código (J.L.Posadas et al., 2008; Emberson & Bate, 2007). Sin embargo, existen protocolos de *cambio de modo* para sistemas de tiempo real (Real & Crespo, 2004) que pueden ser adaptados para ser utilizados en el movimiento de componentes de tiempo real. Estas modificaciones del conjunto de tareas demanda:

- volver a realizar el análisis de planificabilidad para todo el sistema de tiempo real,
- y además, se deberá recalcular una nueva función $\alpha(t)$ para el escalamiento de frecuencia del procesador.

Sin embargo estos análisis pueden llevarse a cabo de forma conjunta. Por otro lado, un aspecto importante a considerar en la migración de tareas, es el coste computacional involucrado en recalcular estos parámetros. Esto es debido a que este cálculo en sí puede significar un **incremento en el consumo de energía** así como un **incremento en el tiempo** para llevar a cabo la incorporación o el rechazo de tareas.

Después de una migración de componentes, un *conjunto de tareas resultante* $\mathcal{T}_{x+1}$ se entenderá como la modificación del conjunto de tareas actual $\mathcal{T}_x$ debido a nuevas inclusiones y expulsiones de tareas. De esta forma, el conjunto $\mathcal{T}_{x+1}$ será definido como:

$$\mathcal{T}_{x+1}^N = \mathcal{T}_x^n + \mathcal{T}_I^m - \mathcal{T}_E^p = \mathcal{T}_x^n + \Delta\mathcal{T}$$

donde el tamaño de $\mathcal{T}_{x+1}$ es $N = n + m - p$. $\mathcal{T}_I$ es el conjunto de tareas que migra al procesador y $\mathcal{T}_E$ es el conjunto de tareas que emigra del procesador, en donde $\mathcal{T}_E \subseteq \mathcal{T}_x$.

La función del consumo de energía suponiendo la movilidad de componentes en un instante de tiempo t_x esta dada por:

$$E(S_y, t_a) = \int_{t_0}^{t_1} \mathcal{P}(F(t, \mathcal{T}_0))dt \; + \; \ldots \; + \; \int_{tx}^{t_{x+1}} \mathcal{P}(F(t, \mathcal{T}_x))dt + \tag{3.1}$$

$$\ldots + \int_{t_{x+k}}^{t_a} \mathcal{P}(F(t, \mathcal{T}_{x+k}))dt$$

donde $F(t, \mathcal{T}_x)$ es la función de frecuencias del procesador obtenida para el conjunto de tareas $\mathcal{T}_x$ durante un periodo de tiempo.

3.4. Cálculo del factor de escalado del procesador

En esta sección se muestra – cómo a partir de algoritmos de planificabilidad presentados en el apartado 1.3 – es posible llegar a expresiones para el cálculo del factor de escalado de frecuencia α para conjunto de tareas con *D=T* y *D≤T*. Por otra parte, se establece que estas expresiones además de minimizar el consumo de energía también pueden ser utilizadas a su vez para comprobar la planificabilidad del sistema.

Note que no todos los ciclos de ejecución pueden ser escalados con la velocidad del procesador porque algunas operaciones interactúan con memoria u otros dispositivos de Entrada/Salida, cuyo tiempo de acceso podría ser siempre fijo (Bini et al., 2005). Sin embargo, el acceso a memoria puede también ser llevado a cabo a diferentes velocidades (Marvell, 2008), y por lo tanto se podrían usar dos factores de escalado: α para escalar el procesador; y β para escalar la velocidad de acceso al bus del sistema. Para simplificar el análisis, se supondrá β como un parámetro fijo igual a 1, de tal modo que el tiempo de ejecución para el peor caso C pueda expresarse como un parámetro dividido en dos partes: un parámetro C^f, el cual es dependiente de la frecuencia del procesador; y un parámetro fijo C^m no escalable.

Por consiguiente, el C_i de las tarea i-ésima puede expresarse como el tiempo de ejecución a la frecuencia máxima del procesador con respecto al factor de escalado de frecuencia α más una constante de tiempo de ejecución:

$$C_i = \frac{C_i^f}{\alpha} + C_i^m \tag{3.2}$$

3.4.1. Método LL

Teorema 3.1. *El cálculo del factor de escalado de frecuencia α usando la bien conocida prueba de planificación de Liu y Layland (ver el apartado 1.2.1) está dado por:*

$$\alpha^{LL} = \frac{U_f}{n(2^{1/n} - 1) - U_m} \tag{3.3}$$

en donde U_f es la suma de la utilización de la parte del conjunto de tareas cuyo cómputo depende de la frecuencia del procesador, U_m es la utilización de la parte de las tareas que no puede ser escalado con la frecuencia del procesador.

Adicionalmente, partiendo de que el factor de escalado está normalizado entre $0 < \alpha \leq 1$, el conjunto de tareas será planificable usando el método LL si α es menor o igual a 1. De lo contrario no se podrá comprobar la planificabilidad del sistema, teniendo en cuenta que el método LL es una prueba suficiente pero no necesaria.

Demostración. Partiendo de la definición en las ecuaciones 1.1 y de la ecuación 3.2, tenemos que:

$$\sum_{i=1}^{n} U_i \leq n(2^{1/n} - 1)$$

i	T_i	D_i	C_i^f	C_i^m
1	5	5	1	0
2	10	10	1	0
3	15	15	1	0
4	20	20	1	0
5	34	34	1	0

Tabla 3.1: Conjunto de tareas para el cálculo del factor α.

$$\sum_{i=1}^{n}\left(\frac{C_i^f}{\alpha \cdot T_i}+\frac{C_i^m}{T_i}\right)\leq n(2^{1/n}-1)$$

despejando α:

$$\sum_{i=1}^{n}\frac{C_i^f}{\alpha \cdot T_i}\leq n(2^{1/n}-1)-\sum_{i=1}^{n}\frac{C_i^m}{T_i}$$

$$\sum_{i=1}^{n}\frac{\frac{C_i^f}{T_i}}{n(2^{1/n}-1)-\frac{C_i^m}{T_i}}\leq \alpha$$

$$\alpha_{min}\triangleq \alpha \geq \sum_{i=1}^{n}\frac{\frac{C_i^f}{T_i}}{n(2^{1/n}-1)-\frac{C_i^m}{T_i}}$$

$$\alpha_{min}\triangleq \alpha \geq \frac{U_f}{n(2^{1/n}-1)-U_m} \tag{3.4}$$

por definición la expresión de la ecuación 3.11 determinará el valor de escalado de frecuencia mínimo que podrá utilizarse mediante el método LL.

Sin embargo, es bien sabido que la prueba LL es un método suficiente pero no necesario, y por consiguiente el factor de escalamiento α^{LL} podría no ser el menor factor α que minimiza el consumo de energía. Esta prueba puede usarse únicamente para tareas con plazos de ejecución iguales al período ($D = T$).

Migración de componentes

Por otro lado, para comprobar la planificabilidad del sistema después de una *migración de componentes* utilizando el método Liu-Layland, y suponiendo que el sistema inicial es planificable y que utiliza un factor α_x^{LL}, el nuevo factor LL α_{x+1}^{LL} estará definido como:

$$\alpha_{x+1}^{LL}=\frac{(U_f^{\alpha_x^{LL}}(t_x)+\Delta U_f^{\alpha_x^{LL}}(t_{x+1}))\alpha_x^{LL}}{n(2^{1/n}-1)-U_m(t_x)-\Delta U_m(t_{x+1})} \tag{3.5}$$

donde ΔU es la diferencia de utilizaciones de las nuevas tareas incorporadas al sistema y de las tareas expulsadas en el instante t_{x+1}. $U(t_x)$ es la utilización del conjunto de tareas en el sistema antes de la migración. α_x^{LL} es el factor de escalamiento de frecuencia antes de la migración de componentes. α_{x+1}^{LL} es el nuevo factor de escalamiento LL que puede ser utilizado después de completarse la migración de tareas.

Ejemplo

A continuación enunciaremos el cálculo del factor α^{LL} con un ejemplo sencillo. Para el ejemplo utilizaremos el conjunto de tareas de la tabla 3.4.1, en donde las tareas $i = 1, 2$ conformarán el conjunto inicial de tareas y el conjunto $i = 3, 4, 5$ corresponderá al conjunto de tareas que migran.

El cálculo del factor α para el conjunto inicial de tareas (α_x^{LL}) utilizando la ecuación 3.3 es igual a:

$$\alpha_x^{LL} = \frac{U_f}{n(2^{1/n}-1) - U_m} = \frac{0{,}3}{2(2^{1/2}-1)} = 0{,}3621$$

Sin embargo, el valor exacto de α_x, utilizando un método exacto como los propuestos más adelante, es igual a $0{,}3$ siendo en este caso el factor α^{LL} bastante aproximado. En términos de consumo de energía, esta diferencia representa un sobre consumo energético del $\%34$.

El cálculo incluyendo el conjunto de tareas restantes que migran utilizando la ecuación 3.5 es igual a:

$$\alpha_{x+1}^{LL} = \frac{(U_f^{\alpha_x}(t_x) + \Delta U_f^{\alpha_x}(t_{x+1}))\alpha_x}{n(2^{1/n}-1) - U_m(t_x) - \Delta U_m(t_{x+1})} = \frac{(U_f^{\alpha_x=0,3621}(t_x) + \Delta U_f^{\alpha_x=0,3621}(t_{x+1}))\,.\,0{,}3621}{5(2^{1/5}-1)} = 0{,}6$$

Siendo el valor exacto de $\alpha_{x+1} = 0{,}4667$, presentando por tanto una diferencia de $0{,}1333$, que en un procesador con $f_{max} = 700Mhz$ representa una diferencia de velocidad de $93Mhz$, lo cual dependiendo del procesador, puede llegar a implicar un sobre consumo energético del $\%56$ comparado con el consumo obtenido utilizando el valor exacto α.

3.4.2. Método HB

Teorema 3.2. *El factor de escalado de frecuencia basado en el método de planificación "límite Hiperbólico" (ver sección 1.2.2) puede calcularse de la siguiente forma:*

$$\alpha_x^{HB} = \text{máx}\{\alpha(n)\} \tag{3.6}$$

en donde $\alpha(n)$ representa los n valores posibles que pueden ser asignados a α como resultado de la resolución de la ecuación resultante del producto de las utilidades más uno $\left(\prod_{i=1}^{n}\left(\frac{U_i^f}{\alpha^{HB}} + U_i^m + 1\right) - 2 = 0\right)$ de todas las n tareas que conforman el sistema de computo.

Adicionalmente, partiendo de que el factor de escalado está normalizado entre $0 < \alpha \leq 1$, el conjunto de tareas será planificable usando el método HB si α es menor o igual a 1. De lo contrario no se podrá comprobar la planificabilidad del sistema, teniendo en cuenta que el método HB es una prueba suficiente pero no necesaria.

Demostración. Partiendo de la definición en las ecuaciones 1.2 y de la ecuación 3.2, tenemos que:

$$\prod_{i=1}^{n}(U_i^f + U_i^m + 1) \leq 2$$

$$\prod_{i=1}^{n}(\frac{C_i^f}{\alpha\,.\,T_i} + \frac{C_i^m}{T_i} + 1) \leq 2$$

$$\left(\left(\frac{U_1^f}{\alpha} + U_1^m + 1\right)\left(\frac{U_2^f}{\alpha} + U_2^m + 1\right)\left(\frac{U_3^f}{\alpha} + U_3^m + 1\right)...\left(\frac{U_n^f}{\alpha} + U_n^m + 1\right)\right) - 2 = 0 \tag{3.7}$$

en donde n es el número de tareas. Todas los factores en la ecuación son conocidos y constante excepto el valor para α. Resolviendo la ecuación se hallarán n valores posibles para α, cuyo valor válido será igual al máximo de los n valores resultantes (α_n).

Al igual que para el método LL, el método HB no puede ser usado para conjuntos de tareas con $D < T$.

Migración de componentes

Para el cálculo del factor de escalado después de una *migración de componentes*, y suponiendo que el sistema inicial es planificable, es necesario resolver los productos para poder aislar el factor α_{x+1}^{HB}

de la ecuación:

$$2 = \prod_{i=1}^{n} \left(\frac{U_i^{f\alpha_x^{HB}} \cdot \alpha_x^{HB}}{\alpha_{x+1}^{HB}} + U_i^m + 1 \right) \cdot \frac{\prod_{j=1}^{inc} \left(\frac{U_j^{f\alpha_x^{HB}} \cdot \alpha_x^{HB}}{\alpha_{x+1}^{HB}} + U_i^m + 1 \right)}{\prod_{k=1}^{exp} \left(\frac{U_k^{f\alpha_x^{HB}} \cdot \alpha_x^{HB}}{\alpha_{x+1}^{HB}} + U_i^m + 1 \right)} \tag{3.8}$$

donde inc y exp son respectivamente el número de tareas incorporadas y expulsadas del conjunto de tareas inicial ($\mathcal{T}(t_x)$). α_x^{HB} es el factor de escalado antes de la migración de componentes.

Ejemplo

La obtención del factor α_{x+1}^{HB} lo haremos de forma práctica a través de un ejemplo. Para el ejemplo nos valdremos del mismo conjunto de tareas expresado en la tabla 3.4.1 y asumiremos el conjunto de tareas 1 y 2 como el conjunto de tareas inicial, para posteriormente incorporar el grupo de tareas restante. El cálculo del factor α para el conjunto de tareas inicial (α_x^{HB}) utilizando la ecuación 3.6:

$$\left(\frac{U_1^f}{\alpha_x} + U_1^m + 1 \right) \left(\frac{U_2^f}{\alpha_x} + U_2^m + 1 \right) - 2 = 0$$

$$\left(\frac{U_1^f}{\alpha_x} + 1 \right) \left(\frac{U_2^f}{\alpha_x} + 1 \right) = 2$$

Llevando a cabo la multiplicación del binomio:

$$\alpha_x^2 - (U_1^f + U_2^f)\alpha_x - U_1^f U_2^f = 0$$

resolviendo esta ecuación cuadrática nos queda:

$$\alpha_x^{HB} = \frac{(U_1^f + U_2^f) \pm \sqrt{(U_1^f + U_2^f)^2 + 4U_1^f U_2^f}}{2}$$

$$\alpha_x^{HB}(1) = 0{,}3562$$

$$\alpha_x^{HB}(2) = -0{,}0562$$

aplicando la ecuación 3.6 obtenemos finalmente que el factor de escalado será igual a:

$$\alpha_x^{HB} = \text{máx}\{\alpha_x(n)\} = 0{,}3562$$

El factor de escalado mínimo exacto es igual a $0{,}3$, con el método de LL obtuvimos $\alpha_x^{LL} = 0{,}3621$ y con este método se obtuvo $\alpha_x^{HB} = 0{,}3562$, por tanto el cálculo del valor α utilizando el método HB es más aproximado al exacto. Este α resultante con HB supone un sobre consumo energético del $\%30$ comparado con el método exacto.

Suponiendo una migración de tareas compuesta por el conjunto tareas restante de la tabla 3.4.1, el nuevo factor de escalado α_{x+1}^{HB} se puede calcular utilizando la ecuación 3.8 y 3.6:

$$2 = \left(\left(\frac{\alpha_x U_1^{f\alpha_x^{HB}}}{\alpha_{x+1}^{HB}} + 1 \right) \left(\frac{\alpha_x U_2^{f\alpha_x^{HB}}}{\alpha_{x+1}^{HB}} + 1 \right) \right) \left(\left(\frac{\alpha_x U_3^{f\alpha_x^{HB}}}{\alpha_{x+1}^{HB}} + 1 \right) \left(\frac{\alpha_x U_4^{f\alpha_x^{HB}}}{\alpha_{x+1}^{HB}} + 1 \right) \left(\frac{\alpha_x U_5^{f\alpha_x^{HB}}}{\alpha_{x+1}^{HB}} + 1 \right) \right)$$

$$\alpha_{x+1}^{HB} = \text{máx}\{\alpha_{x+1}(1), \alpha_{x+1}(2), \alpha_{x+1}(3), \alpha_{x+1}(4), \alpha_{x+1}(5)\} = 0{,}5829$$

El valor exacto para α_{x+1} es $0{,}4667$, por lo que en comparación con el método LL ($\alpha_{x+1}^{LL} = 0{,}6$) la diferencia es menor, pasando esta de $0{,}1333$ a $0{,}1162$, lo cual en un procesador con $f_{max} = 700Mhz$ representa pasar de una diferencia de $93Mhz$ a $81Mhz$ ($408Mhz$) de la frecuencia exacta ($327Mhz$). El sobre consumo energético con este método es del $\%50$ comparado con el consumo usando el factor exacto α.

3.4.3. Método LLM

En esta sección se presenta el cálculo del factor α^{LLM}, basado en la extensión del método LL y descrito en el apartado 1.2.4. El método LLM es válido para tareas con plazos de ejecución menores e iguales al período.

Teorema 3.3. *El factor de escalado de frecuencia basado en la prueba de utilización acotada está dada por:*

$$\alpha^{LLM} = \max_{i=1...n} \frac{f_i^f}{U(p, \Delta_i) - f_i^m} \tag{3.9}$$

donde f_i^f y f_i^m son respectivamente:

$$f_i^f = \sum_{j \in H_p} \frac{C_j^f}{T_j} + \sum_{k \in H_1} \frac{C_k^f}{T_i} + \frac{C_i^f}{T_i} \qquad f_i^m = \sum_{j \in H_p} \frac{C_j^m}{T_j} + \sum_{k \in H_1} \frac{C_k^m}{T_i} + \frac{C_i^m}{T_i}$$

donde H_1 está formado por el conjunto de tareas de mayor prioridad que pueden expulsar la tarea τ_i una única vez antes de su plazo de ejecución D_i, es decir, el conjunto de tareas de mayor prioridad con períodos mayores o igual a D_i. H_p está formado por el conjunto de tareas de mayor prioridad que puede expulsar la tarea τ_i más de una vez antes de su plazo de ejecución D_i, o en otras palabras, el conjunto de tareas de mayor prioridad con períodos de ejecución menores que D_i.

$U(p, \Delta_i)$ es definido como:

$$U(p, \Delta_i) = \begin{cases} p((2\Delta_i)^{1/p} - 1) + 1 - \Delta_i & 0{,}5 \leq \Delta_i \leq 1 \\ \Delta_i & 0 \leq \Delta_i < 0{,}5 \end{cases}$$

donde Δ_i y p son:

$$\Delta_i = \frac{D_i}{T_i} \qquad p = num(H_p) + 1$$

$num(H_p)$ corresponde al número de tareas en el conjunto H_p.

Demostración. Partiendo de la definición de la ecuación 1.4 tenemos que:

$$f_i \leq U(p, \Delta_i)$$

donde f_i es:

$$f_i = \sum_{j \in H_p} \frac{C_j}{T_j} + \sum_{k \in H_1} \frac{C_k}{T_i} + \frac{C_i}{T_i}$$

y utilizando la ecuación 3.2,

$$f_i = \sum_{j \in H_p} \frac{C_j^f}{\alpha T_j} + \sum_{j \in H_p} \frac{C_j^m}{T_j} + \sum_{k \in H_1} \frac{C_k^f}{\alpha T_i} + \sum_{k \in H_1} \frac{C_k^m}{T_i} + \frac{C_i^f}{\alpha T_i} + \frac{C_i^m}{T_i}$$

lo cual puede ser agrupado como:

$$f_i = \frac{f_i^f}{\alpha} + f_i^m$$

reemplazando en la ecuación inicial y despejando α:

$$\frac{f_i^f}{\alpha} + f_i^m \leq U(p, \Delta_i)$$

$$\alpha \geq \frac{f_i^f}{U(p, \Delta_i) - f_i^m}$$

donde $i = 1, ..., n$, donde n es el número de tareas del sistema. De los valores resultantes para cada tarea i, el resultado más grande será el correspondiente para $alpha^{LLM}$:

$$\alpha^{LLM} = \max_{i=1...n} \frac{f_i^f}{U(p, \Delta_i) - f_i^m}$$

Ejemplo

Utilizando el teorema 3.3, el factor de escalado α^{LLM} para el conjunto de tareas $\mathcal{T}$ propuesto en la tabla 3.4.1 es igual a $0{,}6$. Este factor de escalamiento de frecuencia es aproximado y da como resultado un sobre consumo de energía igual a $\%56$, si se compara el consumo del procesador utilizando el factor exacto.

3.4.4. Método RTA

El método recurrente de análisis por tiempo de respuesta (RTA) presentado en la sección 1.2.3 es un test exacto, al que haciéndole algunas modificaciones es posible llegar a un algoritmo para encontrar un valor de factor de escalado de frecuencia α, que minimice el consumo de energía en el sistema. Basado en el artículo (Saewong & Rajkumar, 2003) y haciendo algunas modificaciones se puede obtener una solución exacta y estática para un conjunto de tareas con $D = T$ y $D < T$. Un algoritmo para el cálculo del factor de escalado es el mostrado en la figura 3.3.

Ejemplo

Utilizando el algoritmo de la figura 3.3, el factor de escalado α^{RTA} para el conjunto de tareas $\mathcal{T}$ propuesto en la tabla 3.4.1 es igual a $0{,}4667$. Este factor de escalado determina la mínima frecuencia a la que puede ejecutarse el procesador de forma estática garantizando al mismo tiempo la planificabilidad del sistema.

3.4.5. Método EDF-U

En este apartado se presenta el cálculo del factor α^{EDF}, basado en el análisis de planificabilidad para un sistema basado en prioridades dinámicas descrito en el apartado 1.3.1. Esta prueba es válida para tareas con plazos de ejecución menores e iguales al período.

Teorema 3.4. *El cómputo del factor α usando la prueba de planificabilidad EDF (ver sección 1.3.1) cuando $D = T$ esta dada por:*

$$\alpha^{EDF} = \frac{U_f}{1 - U_m} \tag{3.10}$$

donde U_f es la suma de la utilización de la parte del conjunto de tareas cuyo cómputo depende de la frecuencia del procesador, y U_m es la parte de la utilización de las tareas cuyo cómputo no puede ser escalado con la frecuencia del procesador.

Cuando $D < T$, una prueba simple y aproximada para el cálculo de α puede ser utilizada. α se hallaría de la misma forma como en la ecuación 3.10, pero en el cálculo de la utilización en lugar de usar el período (T_i) se utilizaría el plazo de ejecución, esto es: $U = \frac{C}{D}$.

Demostración Partiendo de la definición en las ecuaciones 1.24 y de la ecuación 3.2, tenemos que:

$$\sum_{i=1}^{n} U_i \leq 1$$

$$\sum_{i=1}^{n} \left(\frac{C_i^f}{\alpha \cdot T_i} + \frac{C_i^m}{T_i} \right) \leq 1$$

```
alpha_RTA:
  for Task(i)
     alpha_tmp(i)=Min_Freq(Cf,Cm,T,D,i);
  end
  alpha=max(alpha_tmp);
  return alpha;

Min_Freq(Cf,Cm,T,D,i):
  S=I=B=Delta=0; alpha=1; IN_BZP=TRUE; w'=0;
  w=(Cf(i)/fmax)+Cm(i);
  While (w<D(i))
    if (IN_BZP==TRUE)
        Delta=D(i)-w;
        While (w<D(i)) && (Delta>0)
           j pertenece hp(Task(i))
              w'=sum(((Cf(j)/fmax)+Cm(j))*(floor(w/T(j))+1))+S;
           Delta=w'-w; w=w';
        end
        IN_BZP=FALSE;
    else
        for j pertenece hp(Task(i))
           I=min(T(j)*ceil(w/T(j))-w , D(i)-w);
        end
        S=S+I; w=w+I; B=w-S;
        if ((B/t) < alpha)
           alpha=B/t;
        end
        IN_BZP=TRUE;
    end
  end
  return alpha;
```

Figura 3.3: Algoritmo para el cálculo del factor de escalado de frecuencia utilizando el método RTA

despejando α:

$$\sum_{i=1}^{n} \frac{C_i^f}{\alpha \cdot T_i} \leq 1 - \sum_{i=1}^{n} \frac{C_i^m}{T_i}$$

$$\sum_{i=1}^{n} \frac{\frac{C_i^f}{T_i}}{1 - \frac{C_i^m}{T_i}} \leq \alpha$$

$$\alpha_{min}^{EDF} \triangleq \alpha \geq \sum_{i=1}^{n} \frac{\frac{C_i^f}{T_i}}{1 - \frac{C_i^m}{T_i}}$$

$$\alpha_{min}^{EDF} \triangleq \alpha \geq \frac{U_f}{1 - U_m} \tag{3.11}$$

Ejemplo

El factor de escalado α^{EDF} para el conjunto de tareas $\mathcal{T}$ propuesto en la tabla 3.4.1 es igual a $0{,}4461$. Este factor de escalado determina la mínima frecuencia a la que puede ejecutarse el procesa-

dor garantizando al mismo tiempo la planificabilidad del sistema, cuando se utiliza un esquema de planificación por prioridades dinámicas.

Si se compara con el α obtenido utilizando un método exacto para un sistema basado en prioridades fijas, tal como RTA (ver 3.4.4), la diferencia es igual a $0{,}4667 - 0{,}4461 = 0{,}0206$. Esto determina que utilizando un esquema de planificabilidad por prioridades dinámicas garantiza una menor frecuencia de procesador. Sin embargo, y tal como se defendió en secciones anteriores, el principal objetivo de esta tesis es el uso de un esquema basado en prioridades fijas. Este método se utiliza únicamente por motivos comparativos de los resultados.

3.4.6. Método basado en la región de planificabilidad

Partiendo de lo expuesto en la sección 1.2.5, en relación al establecimiento de regiones acotadas en el espacio C (C-*Espacio*) para las tareas del sistema, y teniendo en cuenta el ejemplo de la figura 1.3, es fácil pensar en una sintonización de los valores de las variables C_i de tal modo que estos queden dentro de una región acotada de planificabilidad. En este caso, es de especial interés la sintonización de la variable C_i, ya sea de forma independiente o proporcional a todos los C, para que el sistema siga siendo o resulte planificable.

En (Bini et al., 2006) se aborda este aspecto desde una perspectiva de análisis de sensibilidad, en la que se analiza si es posible sumarle o restarle a cada uno de los valores C_i. Sin embargo, ellos no abordan la problemática desde la perspectiva que nos ocupa, relacionada con en el escalamiento de frecuencia del procesador.

Para nuestro propósito el tiempo de cómputo de cada tarea de la ecuación 1.20, dejará de ser la componente variable para convertirse en un parámetro constante, pasando a ser el factor de escalado $\alpha(\mathcal{T})$ nuestro componente de interés a partir del siguiente teorema:

Teorema 3.5. *El factor de escalado mínimo de frecuencia de procesador que minimiza el consumo de energía mientras previene la pérdida de plazos para un conjunto de tareas $\mathcal{T}$ está dado por:*

$$\alpha^{\mathcal{P}}_{min}(\mathcal{T}) = \max_{i=1\ldots n} \min M^*_i(\mathcal{P}_{i-1}) \tag{3.12}$$

en donde P_{i-1} son los puntos de planificación definidos en la ecuación 1.17. Esta expresión puede ser aplicada igualmente para los puntos de planificación S_i:

$$\alpha^{\mathcal{S}}_{min}(\mathcal{T}) = \max_{i=1\ldots n} \min M^*_i(\mathcal{S}_i) \tag{3.13}$$

donde $M^*_i(\mathcal{S}_i)$ está determinada por:

$$M^*_i(\mathcal{S}_i) = \sum_{j=1}^{i} \frac{\left\lceil \frac{s_{kl}}{T_j} \right\rceil C^f_j}{s_{kl} - \left\lceil \frac{s_{kl}}{T_j} \right\rceil C^m_j} \quad / \quad s_{kl} \in \mathcal{S}^*_i$$

s_{kl} es definida en la ecuación 1.7. C^f y C^m son definidas en la ecuación 3.2.

*Partiendo de que el factor de escalamiento de frecuencia está normalizado entre $0 < \alpha \leq 1$, el sistema de cómputo será planificable **sí y solo sí** el valor de α es menor igual a 1, de lo contrario, el procesador no podrá planificar el conjunto de tareas aun funcionando a su máxima velocidad.*

Demostración. Partiendo de la definición de las ecuaciones 1.10 y 3.2, y siguiendo varios cambios para simplificar el análisis, tenemos que:

$$\max_{i=1\ldots n} \min M_i(\mathcal{S}_i) \leq \mathcal{S}_i$$

$$\max_{i=1\ldots n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j \leq s_{kl} \quad / \quad s_{kl} \in \mathcal{S}^*_i$$

$$\max_{i=1\ldots n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil \left(\frac{C^f_j}{\alpha} + C^m_j \right) \leq s_{kl} \quad / \quad s_{kl} \in \mathcal{S}^*_i$$

$$\max_{i=1\ldots n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil \frac{C_j^f}{\alpha} \leq s_{kl} - \max_{i=1\ldots n} \min \sum_{j=1}^{i} \left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j^m$$

despejando α de la expresión nos queda:

$$\max_{i=1\ldots n} \min M_i^*(\mathcal{S}_i) \leq \alpha \triangleq \alpha_{min}$$

donde la matriz modificada M (M_i^*) será:

$$M_i^*(\mathcal{S}_i) = \{m_{kl}^*\}_i \tag{3.14}$$

$$/ \ \{m_{kl}^*\}_i = \sum_{j=1}^{i} \frac{\left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j^f}{s_{kl} - \left\lceil \frac{s_{kl}}{T_j} \right\rceil C_j^m} \ / \ s_{kl} \in \mathcal{S}_i^*$$

Estas expresiones utilizadas en la demostración pueden ser extendidas a los puntos $\mathcal{P}_{i-1}$.

El factor de escala del procesador ($\alpha^{\mathcal{P}_i}$ y $\alpha^{\mathcal{S}_i}$) debe ser mayor e igual que la expresión anterior para que el sistema sea planificable, siendo a la vez el resultado de esta expresión el correspondiente al valor de escalado de frecuencia mínimo que deberá utilizarse en el procesador para que el consumo de energía sea mínimo mediante un esquema de planificación basado en prioridades fijas.

3.5. Propuesta de un método reducido $\mathcal{A}$

La acotación de la región de planificabilidad en sistemas periódicos basado en prioridades fijas en el *C-Espacio* está basado principalmente en el conjunto de puntos de planificabilidad $\mathcal{S}_i$ o $\mathcal{P}_i$ (Teorema 1.1 y 1.2). La estructura y el número de puntos de planificabilidad que conforman estos conjuntos determinan la precisión en el análisis y el número de operaciones matemáticas a realizar o en otras palabras el coste.

En la sección 1.2.5,la región de planificabilidad está basada en las expresiones de $\mathcal{S}_i$ y $\mathcal{P}_{i-1}$, las cuales difieren en el número de elementos que conforman los conjuntos (ecuaciones 1.9 y 1.18):

$$\textit{Tamaño}(\mathcal{S}_i) >> \textit{Tamaño}(\mathcal{P}_{i-1})$$

El tamaño del conjunto de puntos $\mathcal{S}$ puede llegar a ser mucho mayor que el del conjunto $\mathcal{P}$, donde el tamaño del primero dependerá de la separación entre los períodos de las tareas. Por tanto, el número de operaciones a llevar a cabo por uno u otro método también puede llegar a tener grandes diferencias. Sin embargo, la precisión del análisis de las dos propuestas es la misma, ambos son suficientes y necesarios.

En este sentido, en esta sección se propone una reducción en el número de puntos de planificación necesarios para el cálculo de la región de planificabilidad. El objetivo de esta simplificación es disminuir sustancialmente el coste computacional del cálculo del algoritmo DVS, pero intentando al mismo tiempo preservar la precisión del análisis de planificabilidad.

La reducción de estos puntos de planificación se basará en la reducción de los puntos del conjunto $\mathcal{P}_{i-1}$, los cuales a su vez son el resultado de una reducción de puntos del conjunto $\mathcal{S}_i$ ($\mathcal{P}_{i-1} \subseteq \mathcal{S}_i$), dando como resultado de esta forma un nuevo conjunto de puntos al que denominaremos $\mathcal{A}_{i-1}$. Este conjunto $\mathcal{A}_{i-1}$ será un $\mathcal{A}_i \subseteq \mathcal{P}_{i-1}$ que a su vez será $\mathcal{A}_i \subseteq \mathcal{S}_i$.

El cálculo del factor de escalado de frecuencia usando el nuevo conjunto de puntos de planificación estará definido por el siguiente teorema:

Teorema 3.6. *El factor de escalado mínimo de frecuencia de procesador que permite la minimización del consumo de energía garantizando a su vez la planificación de un conjunto determinado de tareas $\mathcal{T}$ está dado por:*

$$\alpha_{min}^{\mathcal{A}}(\mathcal{T}) = \max_{i=1\ldots n} \min M_i^*(\mathcal{A}_i) \tag{3.15}$$

en donde M_i^* está definido en la ecuación 3.14, pero aplicado al conjunto de puntos de planificabilidad $\mathcal{A}$. Este conjunto de puntos $\mathcal{A}_i$ es igual a:

$$\mathcal{A}_i(D_i) = \{D_i \ \cup \ sched\mathcal{A}_i(D_i)\} \tag{3.16}$$

en donde $sched\mathcal{A}(D_i)$ es la matriz característica en el cómputo de los puntos $\mathcal{A}_i$. $sched\mathcal{A}(D_i)$ corresponde a la siguiente expresión matricial:

$$sched\mathcal{A}(D_i) = \begin{bmatrix} \left\lfloor \frac{D_i}{T_1} \right\rfloor T_1 & \left\lfloor \frac{D_i}{T_2} \right\rfloor T_2 & \left\lfloor \frac{D_i}{T_3} \right\rfloor T_3 & \dots & \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \\ & \left\lfloor \left\lfloor \frac{D_i}{T_2} \right\rfloor \frac{T_2}{T_1} \right\rfloor T_1 & \left\lfloor \left\lfloor \frac{D_i}{T_3} \right\rfloor \frac{T_3}{T_2} \right\rfloor T_2 & \dots & \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor T_{j-1} \\ & & \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_3} \right\rfloor \frac{T_3}{T_2} \right\rfloor \frac{T_2}{T_1} \right\rfloor T_1 & \dots & \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor T_{j-2} \\ & & & \dots & \dots \\ & & & & \left\lfloor \dots \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor \dots \frac{T_{j-k-1}}{T_{j-k}} \right\rfloor T_{j-k} \end{bmatrix}$$

$$\forall j \in [1, \dots, i-1] \wedge \forall k \in [0, \dots, i-2] \tag{3.17}$$

donde la dimensión de la matriz está dada por $(i-1)x(i-1)$.

Teniendo en cuenta que el factor de escalado α del procesador está normalizado entre $0 < \alpha \leq 1$, el sistema será planificable si el factor α es menor igual que 1. Si α es mayor que 1, habrá una baja probabilidad de que el sistema pueda ser realmente planificable, por el contrario, si α es menor o igual que 1, el sistema será siempre planificable.

Demostración. Para la demostración es necesario partir del principio del método de la región de planificabilidad. Este método está basado principalmente en el análisis de peor caso de la carga de trabajo (*workload*) para el conjunto de tareas que componen el sistema. Cuya carga no puede ser superior a los plazos de ejecución establecidos como máximos (D_i) para cada una de las tareas.

Por tanto, partiendo de que la carga de trabajo del peor caso $W_i(D_i)$ para la tarea τ_i, corresponde al tiempo total en que el procesador está ocupado atendiendo las tareas de mayor prioridad en el intervalo $[0, D_i]$, más el tiempo de ejecución de la propia tarea τ_i, podemos expresar la condición de planificabilidad para un tarea específica τ_i como:

$$C_i + W_{i-1}(D_i) \leq D_i \tag{3.18}$$

La demanda de procesador en el intervalo $[0, t]$, o lo que es lo mismo, el tiempo requerido por las tareas para ser ejecutadas en ese intervalo, está definido por $\sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$. Por consiguiente, y suponiendo que el procesador no tiene pendiente la ejecución de otra tarea de mayor prioridad (ver sección 3.2.1) que las tareas del subconjunto $\{\tau_1, \tau_2, ..., \tau_n\}$, se puede expresar que:

$$\forall t \;\; W_i(t) \leq \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j$$

Por otro lado, para que la planificación sea factible, la carga de trabajo en el intervalo $[t, D]$ debe ser más pequeña o igual que la longitud de ese intervalo, ya que de lo contrario significaría que el tiempo requerido por las tareas para ser ejecutadas es superior al plazo establecido. Por tanto se tiene que:

$$\forall t \in [0, D_i] \quad W_i(D_i) - W_i(t) \leq (D_i - t)$$

La ecuación anterior también puede ser expresada de la siguiente forma:

$$\forall t \in [0, D_i] \quad W_i(D_i) \leq \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \tag{3.19}$$

Adicionalmente, vamos a definir un término $\psi_i(D_i)$ que representará el último instante en el intervalo $[0, D_i]$ en el cual el procesador está desocupado (en estado *ocioso*), que por definición sería:

$$\psi_i(D_i) = max\{t \in [0, D_i] \;\wedge t \notin \text{ a instantes activos de las tareas}\}$$

De esta expresión podemos concluir que ninguna de las i tareas será activa en el instante $\psi_i(D_i)$ y que todos los trabajos ejecutados hasta ese instante se habrán completado a tiempo. Además de esto,

se puede concluir que el procesador siempre estará ocupado en el intervalo $[\psi_i, D_i]$, y por tanto la carga de trabajo en tal intervalo para el conjunto de tareas con mayor prioridad $\tau_1, ..., \tau_i$ será igual a: $(D_i - \psi_i(D_i))$.

Por tanto, la carga de trabajo $W_i(D_i)$ también puede ser calculada con exactitud por esta función si conocemos el último instante ocioso de procesador $\psi_i(D_i)$, siendo:

$$W_i(D_i) = \sum_{j=1}^{i} \left\lceil \frac{\psi_i(D_i)}{T_j} \right\rceil C_j + (D_i - \psi_i(D_i)) \tag{3.20}$$

Sin embargo, usando esta expresión pasamos de la complejidad de la estimación de la carga de trabajo en sí misma en un intervalo continuo $[0, D_i]$ (ver ecuación 3.19), a la complejidad de la búsqueda de $\psi_i(D_i)$.

No obstante, es posible restringir un conjunto de valores probables para ψ_i. Con este propósito, un primer conjunto de posibles valores puede estar formado por todos los tiempos de activación de las tareas con prioridades mayores que τ_i antes de D_i, junto con el plazo de ejecución de la tarea τ_i, y esto es porque *el último instante posible en que el procesador puede estar ocioso coincidirá justo cuando comience una nueva tarea*.

Y este conjunto de puntos posibles para ψ_i, se corresponde con el conjunto de puntos $\mathcal{S}_i$ enunciados en la sección 1.2.5. Una vez se calcula la carga de trabajo con el conjunto de posibles valores para ψ_i, el valor mínimo resultante de la evaluación de la función 3.19 se corresponderá con la carga exacta de trabajo $W_i(D_i)$, siendo:

$$W_i(D_i) = \min_{t \in \psi_i} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \tag{3.21}$$

Donde $\psi_i = \mathcal{S}_i$ y $\mathcal{S}_i$ está definido en la ecuación 1.6.

Pero este conjunto de posibles valores para ψ_i puede ser acotado aún más mediante el conjunto de valores propuestos en (Bini & Buttazzo, 2004), en donde se enuncia un subconjunto $\mathcal{P}_i(t)$ al que pertenece $\psi_i(t)$. Este subconjunto acotado de puntos de planificabilidad puede ser definido como $\psi_i^*(t)$, donde $\psi_i^*(t) = \mathcal{P}_i(t)$ y $\psi_i^*(t) \subseteq \psi_i(t)$. Este subconjunto $\psi_i^*(t)$ está determinado por:

$$\begin{cases} \psi_0^*(t) = \{t\} \\ \psi_i^*(t) = \psi_{i-1}^* \left(\left\lfloor \frac{t}{T_i} \right\rfloor T_i \right) \cup \psi_{i-1}^*(t) \} \end{cases} \tag{3.22}$$

Esta expresión puede ser probada por inducción sobre i:

Paso inicial. Para $i = 1$, es necesario probar que para una tarea planificable τ_1, $\psi_1(t) \in \psi_1^*(t)$, evaluando la ecuación anterior nos queda:

$$\psi_1^*(t) = \psi_0^* \left(\left\lfloor \frac{t}{T_1} \right\rfloor T_1 \right) \cup \psi_0^*(t) = \left\{ \left\lfloor \frac{t}{T_1} \right\rfloor T_1, t \right\}$$

Puesto que asumimos que τ_1 es planificable, entonces el último instante ocioso del procesador $\psi_1(t)$ puede ser únicamente:

1. en $\left\lfloor \frac{t}{T_1} \right\rfloor T_1$, si el último instante en $[0, t]$ de τ_1 es activo en t.
2. o por el contrario en t.

ambos valores pertenecen al conjunto $\psi_1^*(t)$ y por tanto el paso inicial es probado.

Paso inductivo. Si $\psi_i(t) \in \psi_i^*(t)$ para todo t, es necesario probar que para un conjunto de tareas planificable $\{\tau_1, ..., \tau_i, \tau_{i+1}\}$, entonces también $\psi_{i+1}(t) \in \psi_{i+1}^*(t)$ para todo t. Vamos a considerar el intervalo de tiempo $[\lfloor t/T_{i+1} \rfloor T_{i+1}, t]$. En este intervalo, dos cosas pueden pasar:

1. El procesador esté ocupado en todo el intervalo;
2. existe un instante en el cual el procesador no está ocupado.

Para el primer caso, $\psi_{i+1}(t) = \psi_{i+1}(\lfloor t/T_{i+1} \rfloor T_{i+1})$ porque el procesador siempre corre una tarea en $[\lfloor t/T_{i+1} \rfloor T_{i+1}, t]$. Además para este caso:

$$\psi_{i+1}(\lfloor t/T_{i+1} \rfloor T_{i+1}) = \psi_i(\lfloor t/T_{i+1} \rfloor T_{i+1})$$

de lo contrario el último instante de τ_{i+1} en el intervalo $[0, \lfloor t/T_{i+1} \rfloor T_{i+1}]$ podría perder su plazo de ejecución en $\lfloor t/T_{i+1} \rfloor T_{i+1}$, contradiciendo la hipótesis de que τ_{i+1} es planificable.

Para el segundo caso, en el que existe un instante en el cual el procesador no está ocupado, partiremos de que $y \in [\lfloor t/T_{i+1} \rfloor T_{i+1}, t]$ siendo este un instante de tiempo en donde no habrán tareas activas. Puesto que en el tiempo y el $\lfloor t/T_{i+1} + 1 \rfloor$'s trabajo de τ_{i+1} tiene que haber terminado, τ_{i+1} nunca será activo en $[y, t]$. Esto implica que $\psi_{i+1}(t) = \psi_i(t)$.

Uniendo los dos casos, conseguimos:

$$\psi_{i+1}(t) = \psi_i \left(\left\lfloor \frac{t}{T_{i+1}} \right\rfloor T_{i+1} \right) \vee \psi_{i+1}(t) = \psi_i(t)$$

Por hipótesis inductiva tenemos que:

$$\psi_{i+1}(t) \in \psi_i^* \left(\left\lfloor \frac{t}{T_{i+1}} \right\rfloor T_{i+1} \right) \cup \psi_i^*(t)$$

y finalmente, por la definición de $\mathcal{P}_{i+1}(t)$:

$$\psi_{i+1}(t) \in \psi_{i+1}^*(t)$$

probando con ello la definición de ψ_i^*.

Con esto, podemos expresar la carga de trabajo $W_i(D_i)$ como:

$$W_i(D_i) = \min_{t \in \psi_i^*(D_i)} \sum_{j=1}^{i} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \tag{3.23}$$

y reemplazando esto en la ecuación 3.18, establecemos la condición de planificabilidad para una **determinada** i **tarea**:

$$C_i + W_{i-1}(D_i) \leq D_i$$

$$C_i + \min_{t \in \psi_{i-1}^*(D_i)} \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j + (D_i - t) \ \leq \ D_i$$

$$\min_{t \in \psi_{i-1}^*(D_i)} \ C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \ \leq \ t \tag{3.24}$$

y cuya condición de planificabilidad para **todas las tareas** puede ser claramente expresada como:

$$\max_{i=1 \ldots n} \ \min_{t \in \psi_{i-1}^*(D_i)} \ C_i + \sum_{j=1}^{i-1} \left\lceil \frac{t}{T_j} \right\rceil C_j \ \leq \ t$$

donde ψ_{i-1}^* corresponde con los puntos definidos por $\mathcal{P}_{i-1}$.

Una vez establecido el principio de la región de planificabilidad y de las condiciones de planificabilidad del conjunto de puntos reducido $\mathcal{S}$ y $\mathcal{P}$, nos enfocaremos en la enunciación del conjunto reducido $\mathcal{A}$. Para obtener el nuevo conjunto reducido de puntos $\mathcal{A}_i$ a partir del conjunto $\mathcal{P}$, se propone extraer una función *característica* de los puntos $\mathcal{P}_{i-1}$, de tal forma que se puedan obtener los puntos

más significativos que permiten acotar y delimitar todas las zonas en el tiempo en el que se encontrarán los puntos $\mathcal{P}_{i-1}$. Esta matriz característica, basada en las expresiones matriciales propuestas por esta tesis en 1.2.5 y 1.2.6, está definida como:

$$sched\mathcal{A}(D_i) = \begin{bmatrix} \left\lfloor \frac{D_i}{T_1} \right\rfloor T_1 & \left\lfloor \frac{D_i}{T_2} \right\rfloor T_2 & \left\lfloor \frac{D_i}{T_3} \right\rfloor T_3 & \dots & \left\lfloor \frac{D_i}{T_j} \right\rfloor T_j \\ & \left\lfloor \left\lfloor \frac{D_i}{T_2} \right\rfloor \frac{T_2}{T_1} \right\rfloor T_1 & \left\lfloor \left\lfloor \frac{D_i}{T_3} \right\rfloor \frac{T_3}{T_2} \right\rfloor T_2 & \dots & \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor T_{j-1} \\ & & \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_3} \right\rfloor \frac{T_3}{T_2} \right\rfloor \frac{T_2}{T_1} \right\rfloor T_1 & \dots & \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor T_{j-2} \\ & & & \dots & \dots \\ & & & & \left\lfloor \dots \left\lfloor \left\lfloor \left\lfloor \frac{D_i}{T_j} \right\rfloor \frac{T_j}{T_{j-1}} \right\rfloor \frac{T_{j-1}}{T_{j-2}} \right\rfloor \dots \frac{T_{j-k-1}}{T_{j-k}} \right\rfloor T_{j-k} \end{bmatrix}$$

$$\forall j \in [1, \dots, i-1] \wedge \forall k \in [0, \dots, i-2] \ / \ i = 1, .., n$$

de donde sólo hace falta tener en cuenta el propio plazo de ejecución de la i-ésima tarea, dando como resultado un conjunto total de puntos definido como:

$$\mathcal{A}_i(D_i) = \{D_i \ \cup \ sched\mathcal{A}(D_i)\}$$

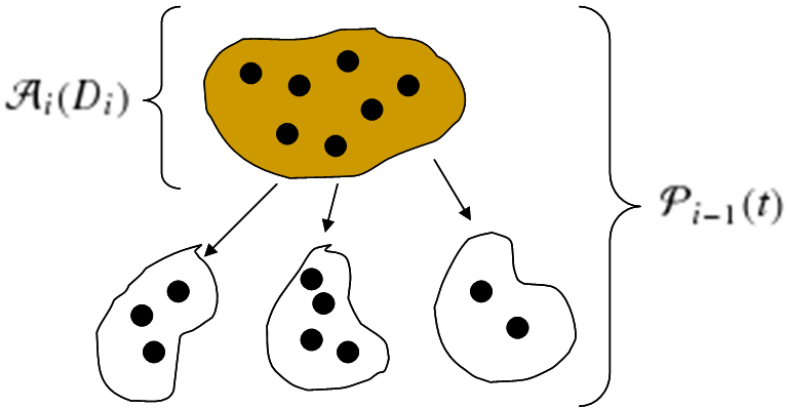

Figura 3.4: De la matriz característica derivan un subconjunto de puntos que conforman en total el conjunto $\mathcal{P}_i(D_i)$

La matriz obtenida en la ecuación3.17 puede considerarse como la matriz característica del conjunto de puntos $\mathcal{P}_{i-1}$. Del conjunto $sched\mathcal{A}(D_i)$ se pueden obtener subconjuntos adicionales de puntos que en total constituyen el conjunto $\mathcal{P}_{i-1}(D_i)$ (ver figura 3.4). Igualmente, esta matriz $sched\mathcal{A}(D_i)$ acota y delimita todas las posibles zonas en el tiempo en el que se concentrarán la totalidad de puntos. En la figura 3.5 se comparan todos los elementos de los conjuntos $\mathcal{P}_{i-1}$ y $\mathcal{A}_i$ para un grupo de 15 tareas. En esta figura se observa que las zonas de concentración de puntos $\mathcal{P}_{i-1}$ están delimitadas por los puntos obtenidos con la matriz característica $sched\mathcal{A}(D_i)$.

Por definición la matriz característica $sched\mathcal{A}(D_i)$ utiliza menos puntos de planificación para la estimación de los posibles valores de ψ_i, por tanto, al utilizar el conjunto de puntos pueden presentarse dos casos:

1. Que el valor exacto para $\psi_i(D_i)$ se encuentre dentro de los valores obtenidos en el cálculo de $\mathcal{A}_i(D_i)$.

2. Que haya una diferencia entre el valor real $\psi_i(D_i)$ y el valor obtenido mediante los puntos $\mathcal{A}_i(D_i)$.

Para el primer caso, obtendríamos un resultado exacto para $W_i(D_i)$ y por consiguiente, la condición de planificabilidad de la ecuación 3.24 sería verificada con exactitud, y por ende, de esta misma forma se calcularía el factor de escalado de frecuencia con un margen de error de 0.

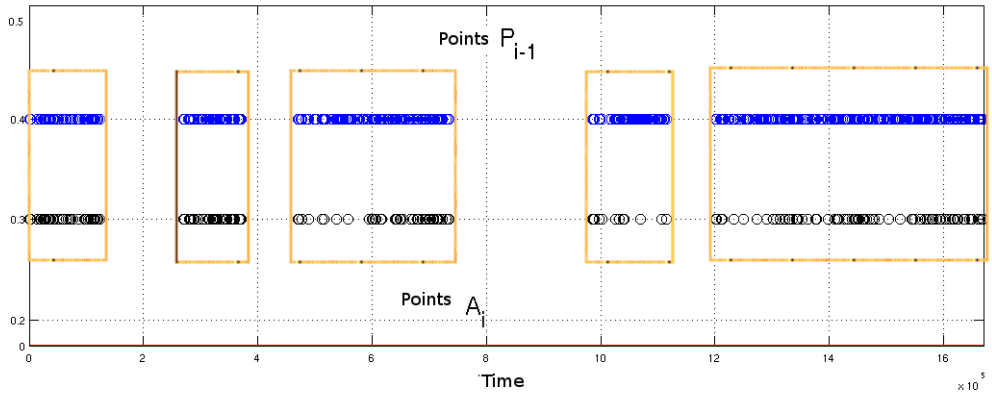

Figura 3.5: Caracterización del conjunto de puntos $\mathcal{P}_{i-1}$ a través del conjunto $\mathcal{A}_i$

Para el segundo caso, partiendo de que la carga de trabajo W_i más pequeña corresponde con la evaluación de $t = \psi_i(D_i)$, y suponiendo que $\psi_i(D_i)$ no se encuentra entre el conjunto $\mathcal{A}_i(D_i)$, entonces la carga de trabajo cuando $t \in \{\mathcal{A}_i(D_i)\}$ será siempre mayor que la real:

$$\psi_i(D_i) \notin \mathcal{A}_i(D_i) \Rightarrow W_i(D_i)^{t=\mathcal{A}_i(D_i)} \geq W_i(D_i)^{t=\psi_i(D_i)} \tag{3.25}$$

Con esto probamos que el resultado del cálculo del factor de escalado de frecuencia utilizando el conjunto $\mathcal{A}$ nunca nos dará un valor por debajo del factor de escala real, sino en todo caso, por encima del valor exacto, asegurando con esto un resultado *suficiente* aunque en algunos casos no *necesario*.

Ahora nos queda por cuantificar cual puede llegar a ser el máximo error al utilizar del conjunto $\mathcal{A}_{i-1}$ el valor más aproximado ($\psi_i^*(D_i)$) al valor exacto de $\psi_i(D_i)$. El error en el cálculo de la carga de trabajo será:

$$error_w = W_i(D_i)^{t=\psi_i^*(D_i)} - W_i(D_i)^{t=\psi_i(D_i)}$$

$$error_w = \sum_{j=1}^{i} \left\lceil \frac{\psi_i^*(D_i)}{T_j} \right\rceil C_j + (D_i - \psi_i^*(D_i)) - \sum_{j=1}^{i} \left\lceil \frac{\psi_i(D_i)}{T_j} \right\rceil C_j - (D_i - \psi_i(D_i))$$

$$error_w = \sum_{j=1}^{i} \left(\left\lceil \frac{\psi_i^*(D_i)}{T_j} \right\rceil - \left\lceil \frac{\psi_i(D_i)}{T_j} \right\rceil \right) C_j + \psi_i(D_i) - \psi_i^*(D_i)$$

y para el caso específico del cálculo del factor de escalado tenemos que el error es:

$$error_\alpha = \sum_{j=1}^{i} \left(\left\lceil \frac{\psi_i^*(D_i)}{T_j} \right\rceil \frac{1}{\psi_i^*(D_i)} - \left\lceil \frac{\psi_i(D_i)}{T_j} \right\rceil \frac{1}{\psi_i(D_i)} \right) C_j$$

$$error_\alpha = \sum_{j=1}^{i} \left(\left\lceil \frac{\psi_i^*(D_i)}{T_j} \right\rceil \psi_i(D_i) - \left\lceil \frac{\psi_i(D_i)}{T_j} \right\rceil \psi_i^*(D_i) \right) \frac{C_j}{\psi_i(D_i) \cdot \psi_i^*(D_i)}$$

Una vez demostrado el origen del conjunto de puntos $\mathcal{A}$, podemos llegar a la fórmula final del factor de escalado de frecuencia α. Partiendo de la definición en las ecuaciones 1.19 y 1.20, del teorema 3.5 y de algunas pequeñas modificaciones para simplificar el análisis, tenemos que:

$$\max_{i=1\ldots n} \min M_i(\mathcal{A}_{i-1}) \leq \mathcal{A}_{i-1}$$

$$M_i(\mathcal{A}_{i-1}) = \{m\}_i \;\; / \;\; \{m\}_i = \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil \frac{\mathcal{C}_j}{\alpha \cdot f_{max}} \;\; / \;\; a \in \mathcal{A}_{i-1}^*$$

donde a es el conjunto de elementos que conforman el conjunto $\mathcal{A}^*_{i-1}$, y $\mathcal{A}^*_{i-1} = \mathcal{A}_{i-1}\ /a\ \exists!a$.

Despejando de $\{m\}_i$ el factor α nos queda que:

$$\max_{i=1\ldots n} \min M_i^*(\mathcal{A}_{i-1}) \leq \alpha \triangleq \alpha_{min} \tag{3.26}$$

donde,

$$M_i^*(\mathcal{A}_{i-1}) = \{m^*\}_i \quad / \quad \{m^*\}_i = \sum_{j=1}^{i} \left\lceil \frac{a}{T_j} \right\rceil \frac{C_j}{a} \quad / \ a \in \mathcal{A}^*_{i-1}$$

Capítulo 4

ESCALADO DE VOLTAJE/FRECUENCIA - FRECUENCIA VARIABLE

4.1. Introducción

En el capítulo anterior presentamos un método para el cálculo de la frecuencia mínima de procesador, cuyo cálculo garantiza que si se usa esta frecuencia en el procesador de forma estática en el tiempo se cumplirán todos los plazos de ejecución del conjunto de tareas del sistema. En el cómputo del factor de escalamiento α se utilizan los parámetros temporales de peor caso para las tareas. Sin embargo, y puesto que en la mayoría de aplicaciones la diferencia en los tiempos de ejecución del mejor y el peor caso puede ser grande, el uso de esta frecuencia mínima podría no ser eficiente desde el punto de vista de ahorro energético.

Por tanto, esa propuesta inicial del capítulo anterior será complementada con un algoritmo dinámico que calculará las frecuencias de procesador en tiempo de ejecución, maximizando de esta forma el ahorro de energía global. Este método se basará en la reclamación y reutilización de los tiempos ociosos resultantes de la finalización anticipada de tareas y como resultado de características intrínsecas del planificador.

4.2. Motivación

Como se comenta previamente, para intentar ahorrar la mayor cantidad de energía no basta con encontrar la frecuencia mínima con la que se asegura la planificabilidad del sistema y un consumo promedio bajo, sino que además es necesaria la utilización de algoritmos que ajusten dinámicamente el consumo energético del procesador mediante cambios de frecuencia para compensar instantes ociosos (o *idle*) del procesador que sólo pueden detectarse en tiempo de ejecución.

Utilizando un algoritmo de planificación RM o DM, los tiempos ociosos pueden ser el resultado de:

1. **Espaciado en las ejecuciones entre tareas** como resultado del escalamiento de los tiempos de ejecución utilizando un factor α constante, calculado mediante el análisis de cargas de trabajo del peor caso para un conjunto determinado de tareas.

   En el escalamiento de un conjunto de tareas utilizando un factor α normalmente no se consigue una utilización del 100 % debido a características propias de algoritmos basados en asignaciones de prioridades RM/DM. Y con esto no estamos diciendo que con algoritmos por prioridades fijas no se pueda planificar un sistema con una utilidad del 100 %. Lo que exponemos es que en la mayoría de los casos cuando se escala un conjunto de tareas con una frecuencia mínima constante, la utilización resultante del conjunto de tareas escalado no será del 100 %, aunque sí estará alrededor de esa utilización.Por tanto, quedarán algunas separaciones ociosas a lo largo de todo el hiper-período entre los tiempos de ejecución asumiendo aun el peor caso de las tareas. Estos tiempos ociosos inherentes se hacen mayores cuando $D < T$.

Estos tiempos ociosos no aparecen en el escalamiento de frecuencia para esquemas de planificación basado en prioridades dinámicas, tales como EDF, cuando $D = T$, pero si están presentes cuando $D < T$.

2. **Terminaciones anticipadas de la ejecución de tareas**. La mayoría de métodos utilizados para encontrar el factor de escalado de frecuencia mínimo o óptimo deben basar su análisis en los tiempos de ejecución de peor caso de las tareas, sin embargo, en la práctica las aplicaciones de tiempo real pueden presentar variaciones significativas entre los tiempos de ejecución para el peor caso y el mejor de los casos y ejecuciones promedio (Aydin et al., 2006), pudiéndose desperdiciar por tanto ciclos de CPU que podría ser utilizados para ahorrar energía.

El primer punto puede ser corregido mediante el análisis completo del hiper-período del conjunto de tareas (Liu & Mok, 2003) del sistema. De este análisis se obtiene un perfil de escalado de frecuencia estático que varia con el tiempo $\alpha(t)$, con cuyo análisis es posible optimizar el consumo de energía. Sin embargo, en entornos donde el conjunto de tareas del sistema puede variar dinámicamente después del movimiento de componentes, tal como es el caso que planteamos, el análisis de todo el hiper-período en tiempo de ejecución resulta sencillamente inviable. Para dichos escenarios es más factible el cálculo de un α constante y una corrección en tiempo de ejecución de estos tiempos ociosos.

La segunda suposición es bastante difícil de corregir de forma predeterminada o lo que es lo mismo, corregir mediante análisis previos fuera de línea en la fase de diseño del sistema. La única forma de calcular el tiempo de ejecución de una tarea en cada activación pasa por conocer cual será el comportamiento en el tiempo de todo el sistema ciber-físico, que incluye las interacciones entre el mundo físico y de cómputo, y cuyas interacciones definirán los eventos que podrían fijar la cantidad de cómputo requerido en cada tarea y en cada instante de activación. Por tanto, en la práctica un análisis previo es sencillamente inviable y más aún si se plantea un entorno con condiciones cambiantes. Es por ello que para abordar esta segunda suposición en la mayoría de la literatura (Zhong et al., 2007) se utilizan algoritmos en tiempo de ejecución para compensar este tipo de desperdicio de ciclos de CPU.

Para ajustar el desperdicio de energía debido a estos dos tipos de instantes ociosos de procesador, y teniendo en cuenta el entorno con soporte para la delegación de código propuesto en esta tesis, se propone como requisito de aplicación la utilización de métodos para el reajuste de la frecuencia de procesador en tiempo de ejecución.

En este capítulo se presentará una visión en este asunto y se propondrá un método nuevo para realizar este ajuste energético, el cual se comparará posteriormente con otros algoritmos encontrados en la literatura.

4.2.1. Trabajos relacionados

En la literatura la mayoría de algoritmos para su aplicación en tiempo de ejecución **Culver05**, está dirigida a esquemas de planificación basado en prioridades dinámicas. Esto es debido a que en estos esquemas es más simple esta recuperación.

Sin embargo, en esquemas de planificación por prioridades fijas hay dos algoritmos dinámicos bastante conocidos en la literatura para la recuperación de tiempos ociosos , estos son **ccRM** (Pillai & Shin, 2001; Tsai et al., 2007) y **lppsRM** (Shin et al., 2000; Culver & Khatri, 2005). Estos dos métodos son iguales en el sentido de que ambos se basan en el mismo punto de partida: la frecuencia de procesador mínima constante. Pero se diferencian en que el primero ajusta la frecuencia del procesador en presencia de instantes ociosos cuando hay una sola tarea activa, y el segundo método ajusta el procesador aún en presencia de varias tareas activas.

Sin embargo, la utilización del método **ccRM** en tiempo de ejecución puede resultar inviable, debido principalmente a su complejidad, ya que requiere una gran cantidad de información futura para que realmente pueda ser aplicado eficazmente. Este algoritmo debe analizar el sumatorio de los tiempos de ejecución restantes de todas las tareas que se encuentren activas en el momento de empezar el análisis y los tiempos de ejecución de todas las tareas que se activen posteriormente antes del *deadline* más próximo, de tal forma que se pueda establecer cuanto hay que reducir la velocidad de la CPU de forma proporcional para todas las tareas, y esto repetido sucesivamente en cada activación. Si se analizan la mayoría de implementaciones de planificadores, la mayor parte de la información

requerida por este método podría no estar disponible o requerir un gran esfuerzo para obtenerla. Además, toda esta sobrecarga es dependiente del número de tareas, lo que se puede traducir en cambios de contexto de sistema no deterministas, característica no deseable en sistemas de tiempo real.

El método **lppsRM** puede resultar más sencillo y rápido porque sólo necesita información del instante de activación justo después de la tarea que está analizando. Se han realizado simulaciones comparativas de estos dos métodos en artículos como el de (Kim et al., 2002), en donde el algoritmo **ccRM** resulta más eficiente a la hora de ahorrar energía que el **lppsRM**, aunque no se tienen en cuenta particularidades como las anteriormente mencionadas. En la enunciación original de ambos métodos se proponen acciones poco realistas o ideales, como es el caso de Shin (Shin et al., 2000), que propone el uso del modo *power-down* del procesador en tiempos ociosos, cuya utilización puede resultar inviable en la mayoría de aplicaciones. Este tipo de suposiciones lleva a que los dos métodos se encuentren bastante alejados de su aplicabilidad real para obtener un ahorro teórico óptimo. Por lo tanto, resulta interesante la propuesta de nuevos algoritmos (Kim et al., 2002) que mejoren la recuperación de energía de forma dinámica en RM y DM, teniendo en cuenta además detalles de implementación.

En este capítulo y en el posterior de evaluación comparativa, se supondrá que el algoritmo *lpps* en lugar de usar el modo *power-down* usará el "modo *idle*", en cuyo modo de operación el reloj que va al *core* de la CPU es detenido, de tal forma que la CPU pueda reactivarse rápidamente.

4.3. Modelo del sistema

Para la enunciación de la propuesta en esta sección se presenta un conjunto de modelos para la definición del sistema. Para el modelo de tareas se utilizará el mismo modelo presentado en la sección 3.2.1. El modelo del procesador estará en la misma línea que el presentado en la sección 3.2.2, pero se ampliarán las características de consumo energético y los modos de operación.

4.3.1. Modelo de procesador

Lo primero que hay que tener en cuenta es la curva característica de consumo del procesador que es específica para cada hardware. Sin embargo, en todos los procesadores la relación entre el consumo de potencia y la frecuencia de operación del procesador puede expresarse mediante polinomios de tercer y segundo grado ((Li & Ding, 2001) (Jejurikar & gupta, 2002) (Aydin et al., 2004) (Zhong et al., 2007) (Marinoni & Buttazzo, 2007)). En la figura 4.1 puede verse la curva normalizada de consumo para un procesador Crusoe, la cual es igual que la utilizada en el modelo presentado en el capítulo anterior (figura 3.1) y que será utilizada como referencia en esta sección y simulaciones posteriores.

Partiendo de esta caracterización del consumo, donde el consumo del procesador se incrementa de forma convexa a la frecuencia del procesador, es sencillo llegar a la conclusión de que se ahorra mayor energía en cuanto más se extienda el tiempo de ejecución de las tareas, o lo que es lo mismo, entre mayor sea el escalamiento de C_n dentro de su plazo de ejecución, lo cual se logra reduciendo la frecuencia de operación del procesador. En un principio, también es posible llegar a pensar que se ahorraría mayor energía si se ejecuta lo más rápidamente posible la tarea y después se lleva el procesador a un estado de reposo, pero esto no es así.

En la figura 4.2 se representa el consumo energético por unidad de tiempo para una tarea con $C = 1$, $T = 10$ y $D = 10$, y se evalúa a diversas frecuencias de procesamiento. En esta figura se puede observar que el menor consumo energético se presenta cuando el tiempo de cómputo se extiende a lo largo de su plazo de ejecución y el mayor consumo se exhibe cuando se utiliza la frecuencia máxima del procesador, lo cual es debido a que en el eje vertical (E_n) los valores de consumo varían de forma cúbica, al tiempo que C lo hace de forma lineal en el eje horizontal. Si la curva de consumo mostrada en la figura 4.2 tuviera un comportamiento lineal, el consumo para la tarea sería el mismo tanto si se utiliza la máxima frecuencia y después se lleva a reposo el procesador, como si se utiliza la frecuencia más pequeña posible a lo largo de D.

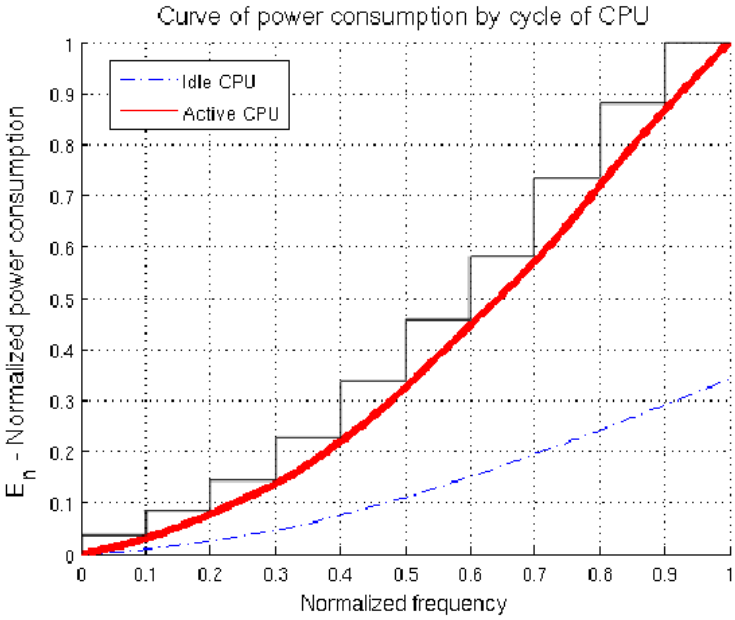

Figura 4.1: Modelo normalizado del consumo de potencia por ciclo de CPU

Aunque sabemos que lo ideal es extender lo máximo posible el tiempo de ejecución de las tareas, esto no siempre es posible, especialmente cuando $D \leq T$, lo que da lugar a instantes ociosos de procesador como los comentados previamente. Para estos instantes es necesario definir procedimientos con el fin de conseguir el máximo ahorro posible de energía.

Pero para la definición de estos procedimientos, es necesario presentar los modos de operación más frecuentes en los procesadores, utilizados para gestionar el ahorro de energía en instantes de inactividad. Esto modos se describirán en la sección 4.3.2.

Variación de la frecuencia de procesador en base al voltaje de alimentación

En esta sección se describe cómo se lleva a cabo en la práctica el escalado de frecuencia en procesadores reales. En la mayoría de procesadores el cambio de frecuencia está relacionado directamente con el voltaje de alimentación (Vdd). En función del voltaje que se aplique al procesador este tendrá una frecuencia de operación, a mayor voltaje mayor frecuencia y a medida que baje la tensión baja la frecuencia. En este tipo de procesadores la frecuencia no es una variable independiente, esta acoplada al valor de Vdd. Un ejemplo claro de este tipo de procesadores es el Transmeta Crusoe TM5400(Corporation, 2000), el cual fue uno de los primeros procesadores de la marca Transmeta con gestión de energía, el cual tenía la siguiente correspondencia entre frecuencia y voltaje (tabla 4.1):

Frecuencia (Mhz)	Voltaje Vdd	Consumo Relativo (%)
700	1.65	100
600	1.60	80.59
500	1.50	59.03
400	1.40	41.14
300	1.25	24.60
200	1.10	12.70

Tabla 4.1: Frecuencia de reloj versus voltaje de alimentación para el procesador Crusoe TM5400

Sin embargo, en los últimos procesadores de la familia Transmeta Crusoe (Corporation, 2008) es

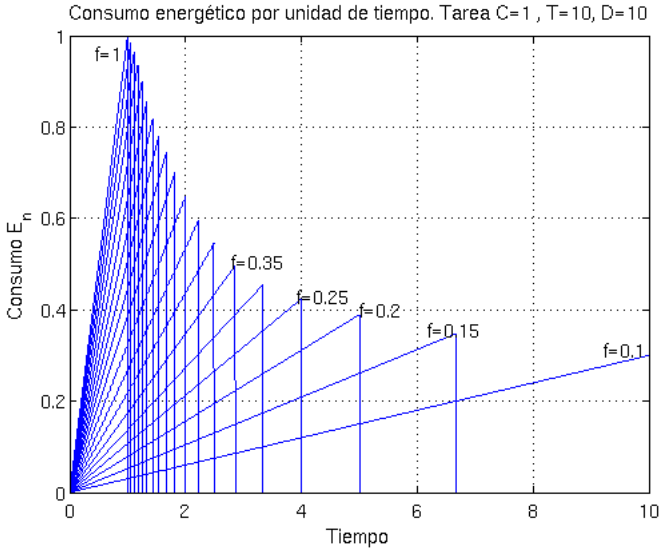

Figura 4.2: Evaluación del consumo energético de una tarea escalada en diversas frecuencias de CPU

posible cambiar el voltaje de alimentación en pasos de 25mV lo que equivale en frecuencia a pasos de 33Mhz(Kadayif et al., 2004), lo cual da la posibilidad de realizar cambios de frecuencia casi continuos.

Otra de las inclinaciones de la tecnología DVFS aplicada a los procesadores, es la utilización de PLL's (Phase-Locked Loop) para cambios de frecuencia en combinación con cambios de voltaje de alimentación, para obtener más niveles de operación o más frecuencias de trabajo para el procesador. Adicionalmente, también es posible realizar cambios de frecuencia en el *bus del sistema* y en la *velocidad de acceso a la SDRAM*. Un ejemplo de esta configuración es la utilizada en los procesadores XScale PXAxxx (Corporation, 2003), (Corporation, 2006) y (Marvell, 2008). A continuación mostraremos las posibles frecuencias que pueden ser asignadas al procesador PXA255:

1 PLL	2 PLL	Normal 1	Turbo 1	Turbo 2	Turbo 3	Bus sistema	Frec SDRAM
27	1	99.5@1.0V	-	199.1@1.1V	298.6@1.3V	50	99.5
36	1	132.7@1.0V	-	-	-	66	66
27	2	199.1@1.0V	298.6@1.1	398.1@1.3V	-	99.5	99.5
36	2	265.4@1.1V	-	-	-	132.7	66
45	2	331.8@1.3V	-	-	-	165.9	83
27	4	398.1@1.3V	-	-	-	196	99.5

Tabla 4.2: Frecuencia de reloj para el procesador PXA255

En la tabla podemos ver distintas configuraciones de PLL y como en algunas de estas es posible subir la frecuencia del procesador por el aumento del voltaje de alimentación, a lo que más comúnmente se conoce como *overclocking*.

4.3.2. Modos de operación de bajo consumo en los procesadores

Los procesadores modernos tienen varios modos de operación del procesador adicionales a la capacidad de cambiar de forma dinámica la frecuencia de procesamiento. Estos modos son parte del sistema de gestión energética que incorporan los sistemas de cómputo. A continuación enunciaremos los modos de operación más comunes, los cuales serán utilizados como complemento al establecimiento de las frecuencias de procesador.

Modo "standby"

En este modo todas las actividades del procesador son detenidas, excepto el reloj de tiempo real (RTC) y el gestor de potencia y relojes. La CPU y todos los periféricos son detenidos pero retienen su estado de contexto. La memoria SRAM puede retener su estado opcionalmente. Todos los PLL son automáticamente deshabilitados. Puesto que toda las actividades internas han sido detenidas, el procesador sólo podrá salir de este estado mediante eventos del RTC o eventos externos programados (ejemplo el teclado). En (Marvell, 2008) se proporcionan algunas latencias para los procesadores XScale, las cuales pueden ser tomadas como referencias orientativas para el uso en otros procesadores. Un ejemplo de las latencias para entrar y salir de este modo son mostradas en la tabla 4.3.

Cambio de modo	**Latencia**
Activo $\rightarrow$ Standby	1230μs – 1410μs
Standby $\rightarrow$ Activo	670μs - 865μs

Tabla 4.3: Latencias para cambios de modo *standby*

Modo "sleep"

En este modo se le quita la alimentación a la CPU y a todos los periféricos del procesador. Todos los relojes y PLL son deshabilitados excepto los contadores y el RTC. Este modo requiere reiniciar el procesador ya que el *contador de programa* deja de ser válido. El estado del procesador es borrado y nuevamente recuperado cuando comienza el reinicio del procesador. El procesador sólo podrá salir de este modo mediante eventos externos o eventos del RTC. En la tabla 4.4 se muestran algunas latencias orientativas para el paso del procesador de un modo activo a un modo *sleep*: entre 1230μs y 1410μs; y de un modo *sleep* a un modo activo: entre 990μs(para despertar) $+$ 865μs (la salida del modo es completada).

Cambio de modo	**Latencia**
Activo $\rightarrow$ Sleep	1415μs – 1800μs
Sleep $\rightarrow$ Activo	990μs + 865μs

Tabla 4.4: Latencias para cambios de modo *sleep*

Modo "idle"

Cuando el procesador entra en *modo idle* el reloj que va al *core* de la CPU es detenido. En este modo no se cambia la generación del reloj, de tal forma que cuando se generen peticiones del servicio de interrupciones solicitando la atención de la CPU, esta pueda reactivarse rápidamente en el estado previo al *mode idle*. Durante este modo el reloj de tiempo real, los temporizadores del sistema operativo, el controlador de interrupciones, las entradas/salidas de propósito general (GPIO), las unidades de periféricos, los controladores de memoria, entre otros, permanecen activos.

Sin embargo, en la mayoría de procesadores con capacidades DVFS, la frecuencia que este seleccionada en el momento de entrar en *modo idle* afecta directamente al consumo de potencia del procesador para este modo. Como es de esperarse, el consumo para este modo es mucho menor que en estado activo pero crece igualmente de forma cúbica en función de la frecuencia. Es por ello que en la figura 4.1 aparecen dos curvas de consumo en función de la frecuencia, una para cuando el procesador está en estado activo y otra para el estado *idle*. Esto igualmente puede verse reflejado en pruebas prácticas, tales como las realizadas en (Zhong et al., 2007), en donde se muestra el consumo de potencia medido para una plataforma BitsyXb (ADS) equipada con un procesador PXA270 (Corporation, 2006), del que se obtienen consumos globales como los mostrados en la figura 4.3. En esta figura se comparan los consumos de la plataforma cuando el procesador está en estado activo y estado

idle configurado para las diferentes frecuencias que soporta, igualmente se puede observar como el consumo puede ser modelado por un polinomio de tercer orden.

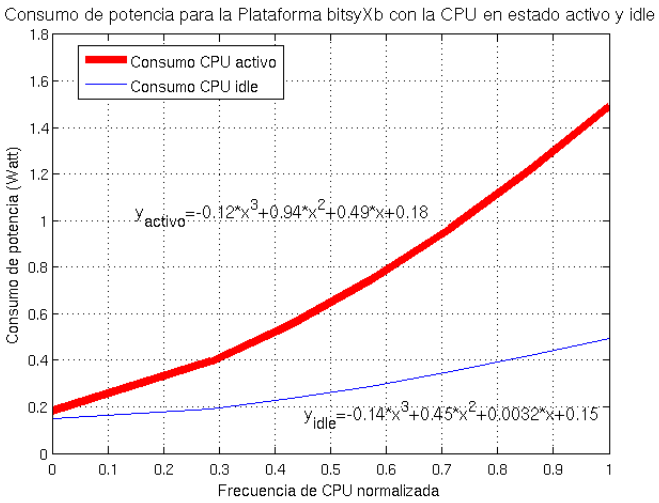

Figura 4.3: Consumo de potencia para la plataforma BitsyXb cuando la CPU está en estado activo y en estado *idle*

Modos especiales "idle"

En el intento por disminuir aún más el consumo de energía en instantes ociosos, en varias familias de procesadores se han creado modos especiales "*idle*". En el procesador PXA255 (Corporation, 2003), por ejemplo, se ha creado un modo denominado "*33-Mhz idle*", el cual tiene el consumo de potencia más bajo comparado con cualquiera de los modos normales *idle* a diferentes frecuencias. Este modo configura al procesador en una frecuencia especialmente baja antes de entrar en modo ocioso. Este modo guarda similitudes con un modo *idle* normal, en que el reloj que va al *core* de la CPU es detenido durante el periodo de inactividad del procesador e igualmente monitoriza peticiones del servicio de interrupción. Pero este modo se diferencia del modo normal *idle*, en que las unidades de periféricos deben ser deshabilitadas porque estas no funcionan a esta frecuencia, la memoria debe ser puesta en un modo refresco y en que para entrar y salir de este modo se debe activar una *secuencia de cambio de frecuencia*. Los módulos tales como reloj de tiempo real, temporizadores del sistema operativo, controlador de interrupción, GPIO, entre otros, permanecen funcionales.

En otros procesadores como los de la familia PXA27x (Corporation, 2006), se ha creado un modo parecido al anterior denominado "*deep-idle*", el cual es una mezcla entre un modo *idle* y un modo *13-Mhz*, en el cual se apaga el PLL que alimenta el *core* de la CPU, forzando a todos los relojes internos que derivaban de este PLL a conectarse a un oscilador de 13Mhz. La diferencia principal del modo *deep-idle* con el modo *33-Mhz idle* del procesador PXA255, es que los periféricos mantienen su funcionalidad cuando se entra en este modo porque derivan de un PLL diferente. En los procesadores PXA3xx (Marvell, 2008) este modo especial se denomina "*ring oscillator*", en este modo la CPU funciona a 60Mhz, la memoria externa a 30Mhz y la SRAM interna funciona a 60Mhz. En el modo *ring oscillator* no todos los periféricos funcionan. Las latencias aproximadas en este modo pueden ser vistas en la tabla 4.5, aunque a cada una hay que sumarle el tiempo que lleva un cambio de frecuencia.

4.4. Recuperación y gestión de instantes ociosos

Del modelo y modos de operación del procesador presentados previamente es posible extraer varias conclusiones. Algunos modos de operación tales como *standby* y *sleep* no pueden utilizarse para gestionar los tipos de tiempos ociosos de procesador mencionados en la sección 4.2, primero, porque

Cambio de modo	Latencia
Activo → *ring oscillator*	18.5μs
ring oscillator → Activo	8μs

Tabla 4.5: Latencias para cambios de modo *ring oscillator*

el coste temporal para entrar o salir de cada uno de estos modos es bastante alto; y segundo, porque para salir de estos modos se necesitan eventos externos como los del teclado o eventos del reloj RTC, cuya resolución normalmente es de aproximadamente 31μs. Tales circunstancias resultan inviables para nuestro propósito ya que se puede comprometer el cumplimiento de las restricciones temporales de las tareas. Adicionalmente, en estos modos la mayoría de periféricos se desactivan, por lo que si se esperan eventos provenientes de estos, por ejemplo de la red de comunicaciones, estos seguramente no serán detectados y se perderán.

El modo *idle* y los modos especiales *idle* resultan más adecuados para el escenario planteado en esta tesis. Por lo que en este capítulo únicamente tendremos en cuenta estos modos de operación.

4.5. Propuesta de procedimientos para la gestión de instantes ociosos

Tal como se expuso previamente, a pesar de que en el *modo idle* el consumo energético es mucho menor que en modo activo, el consumo en este modo sigue dependiendo de forma cúbica de la frecuencia del procesador. Por ello se propone un conjunto de pasos para gestionar esta situación: cuando se detecta un instante ocioso, antes de cambiar el modo del procesador a *idle* se debe *bajar la velocidad del procesador a la mínima* posible, y una vez se reanuda la actividad, se debe recuperar la frecuencia anterior. Con esto se ahorra una mayor cantidad de energía y a su vez, todos los periféricos estarán disponibles. Sin embargo, hay que tener en cuenta que esta propuesta esta limitada por el coste temporal de los cambios de frecuencia, de tal forma que únicamente es aplicable si los instantes ociosos son mayores al coste de dos secuencias de cambios de velocidad.

A continuación, mostramos el pseudo-código de una primera propuesta (ver figura 4.4).

```
last_freq;
idle_mode_detected()
{
   last_freq=current_freq;
   if (next_activation()>2*cost_seq_change_freq)
      next_activation=next_activation-cost_seq_change_freq;
      seq_change_freq(min_freq);
   end
   entry_idle_mode();
}
out_idle_mode()
{
   if (last_freq!=current_freq)
      seq_change_freq(last_freq);
   end
}
```

Figura 4.4: Algoritmo 1: Gestión de tiempos ociosos mediante cambios de frecuencia

Por otro lado, los modos especiales *idle* también podrían ser utilizados en los instantes ociosos para aumentar aún más el ahorro energético. Sin embargo, para esto hay que tener en cuenta que no en todos los procesadores están disponibles, que en este modo no todos los periféricos funcionarán

correctamente, por lo que sólo se podrán utilizar en el caso que no se dependa fuertemente de eventos externos provenientes de los periféricos que se desactivan. Igualmente se deberá tener en cuenta los costes temporales derivados del uso estos modos especiales, de tal forma que se pueda determinar los instantes apropiados para su aplicación El coste temporal del uso de estos modos especiales se incrementa con respecto al uso del modo normal *idle*. A continuación, mostramos el pseudo-código de una segunda propuesta (ver figura 4.5). Este pseudo-código es una ampliación de la primera propuesta.

```
last_freq;
idle_mode_detected()
{
    last_freq=current_freq
    if (next_activation()>(cost_special_idle+2*cost_seq_change_freq))
        desactive_pheripherical();
        next_activation=next_activation-cost_seq_change_freq-cost_special_idle;
        entry_special_idle_mode();
    else if (next_activation()>2*cost_seq_change_freq)
        next_activation=next_activation-cost_seq_change_freq;
        seq_change_freq(min_freq);
        entry_idle_mode();
    else
        entry_idle_mode();
    end;
}
out_idle_mode()
{
    if (current_freq!=last_freq)
        seq_change_freq(last_freq);
    end
}
out_special_idle_mode()
{
    seq_change_freq(last_freq);
    active_pheripherical();
}
```

Figura 4.5: Algoritmo 2: Gestión de tiempos ociosos mediante cambios de frecuencia, incluyendo modos de operación *idle* especiales del procesador.

4.6. Propuesta del algoritmo *rmit* para la recuperación de instantes ociosos del procesador

Para el diseño de esta nueva propuesta nos basaremos en los planteamientos básicos de secciones anteriores, y especialmente en la sección 4.3.1, que en resumidas cuentas sugieren que para maximizar el ahorro de energía después de identificar la frecuencia mínima de procesador, es necesario expandir al máximo la ejecución de las tareas a lo largo de los tiempos ociosos en cada instante disponible, y cuando no sea viable, intentar gestionar esta inactividad de una forma eficiente. Y esto cuidando a su vez, características de implementación tales como el sobre coste computacional que conlleve cada acción y la disponibilidad o facilidad con la que se puede obtener la información requerida desde el planificador. Y como objetivos intrínsecos de la propuesta, el determinismo y el cumplimiento de las restricciones temporales del sistema.

A continuación proponemos un método dinámico coordinado para la "*recuperación y gestión de instantes ociosos*", al que nos referiremos con la abreviatura ***rmit*** (*recovery and management of idle time*).

El método ***rmit*** ajustará la frecuencia de operación del procesador en tiempo de ejecución basado en la propuesta de los algoritmos 1 o 2 (figuras 4.4 y 4.5, cuya utilización dependerá del tipo de procesador), en el análisis del estado de las tareas en el planificador, en el análisis de los tiempos de activación y en los parámetros propios de la tarea a evaluar. Es evidente que el coste de implementación del algoritmo dependerá principalmente del esquema específico en cada planificador, de la forma en que este organice las colas de tareas y de los estados que mantenga de estas. Por tanto, para la formulación de este algoritmo vamos a suponer un planificador como ese encontrado en el sistema operativo de tiempo real RT-Linux ((Coronel et al., 2004),(Martínez et al., 2007)), para el cual el autor de esta tesis realizó una adaptación a la arquitectura XScale (Coronel, 2005). En ese planificador de prioridades fijas se mantiene un control de la información del estado de todas las tareas: la tarea que comienza una ejecución, de la cola de tareas que están *activas* esperando por el uso del procesador y de la cola de tareas que están suspendidas esperando por su próxima activación, esta información se obtiene del estado de *flags* en estructuras de datos. Entre los tiempos de *activación* de las tareas que se encuentran suspendidas esperando por su próximo período y de las tareas que acaban de terminar su ejecución, el planificador escoge el instante de activación más próximo para programar el temporizador. Por tanto, muchas veces con sólo comprobar la programación del temporizador se podrá obtener información.

Las invocaciones al planificador son debidas principalmente a interrupciones provocadas por vencimientos en la programación de temporizadores o por la terminación anticipada de la ejecución de tareas. En estas invocaciones además de programar la próxima activación del temporizador, se comprueba la cola de tareas que están esperando por el uso del procesador y se escoge la tarea siguiente con mayor prioridad.

El algoritmo propuesto ***rmit*** basa su optimización *en línea* mediante acciones en el comienzo y finalización de las tareas. Por ello, el algoritmo será dividido en dos partes, una fase de *arranque* y una fase de *terminación*.

4.6.1. Método *rmit* — Arranque

Esta fase centra su optimización de consumo en el comienzo de las tareas. Este método se integra de forma transparente en el esquema del planificador, es decir, no hay que hacer modificaciones excepcionales en la secuencia del planificador ni en la estructura de datos de las tareas. En la figura 4.6 se representa el funcionamiento de esta primera fase mediante un diagrama de flujo.

Cuando una tarea τ_i es asignada al procesador después de un cambio de contexto y antes de comenzar su ejecución, el algoritmo *rmit* entra en acción y comprueba si hay tareas activas en espera de ser atendidas. Esta comprobación se hace consultando el número de tareas en la cola de ejecución mantenida por el planificador. En caso de que si hayan tareas activas, el algoritmo comprueba si la frecuencia configurada en ese momento es la establecida como la mínima constante, en caso contrario, esta es fijada en el procesador. De esta forma, el comienzo de la ejecución de una tarea siempre se hace a la frecuencia mínima (α_{min}), frecuencia con la que se ha calculado que el sistema será planificable en condiciones de peor caso.

En caso de que **no** hayan tareas adicionales activas en el momento de comenzar la ejecución de la tarea τ_i, se procederá a comprobar si se puede ampliar el tiempo de ejecución de la tarea reduciendo la frecuencia del procesador. Lo primero que comprobamos es si la próxima activación programada ($t(pa)$) es menor o igual que el plazo de ejecución absoluto ($t_{abs}(D_i)$) de la tarea τ_i. La próxima activación puede obtenerse del tiempo programado en el temporizador utilizado por el planificador en ese instante. En caso de que se cumpla que $t(pa) \leq t_{abs}(D_i)$, se procederá a una nueva comprobación para verificar si la próxima activación programada ($t(pa)$) será menor igual que el tiempo absoluto de ejecución del peor caso ($t_{abs}(C_i^{\alpha_{min}})$) calculado a la frecuencia mínima. Si se cumple que $t(pa) \leq t_{abs}(C_i^{\alpha_{min}})$, esto significará que otra tarea solicitará el uso del procesador incluso antes de que la tarea τ_i termine su ejecución. En tal caso, no habrá tiempo disponible y no se podrá escalar el procesador por debajo de la frecuencia mínima (α_{min}). Por tanto, para asegurar la planificabilidad del sistema, la frecuencia del procesador debe fijarse a α_{min} antes de comenzar la ejecución de τ_i.

En caso de que $t(pa) \leq t_{abs}(D_i)$ y $t(pa) \leq t_{abs}(C_i^{\alpha_{min}})$ **no** se cumpla, significaría que habrá un intervalo de tiempo ocioso que puede recuperarse y se procederá al cálculo de una nueva frecuencia de

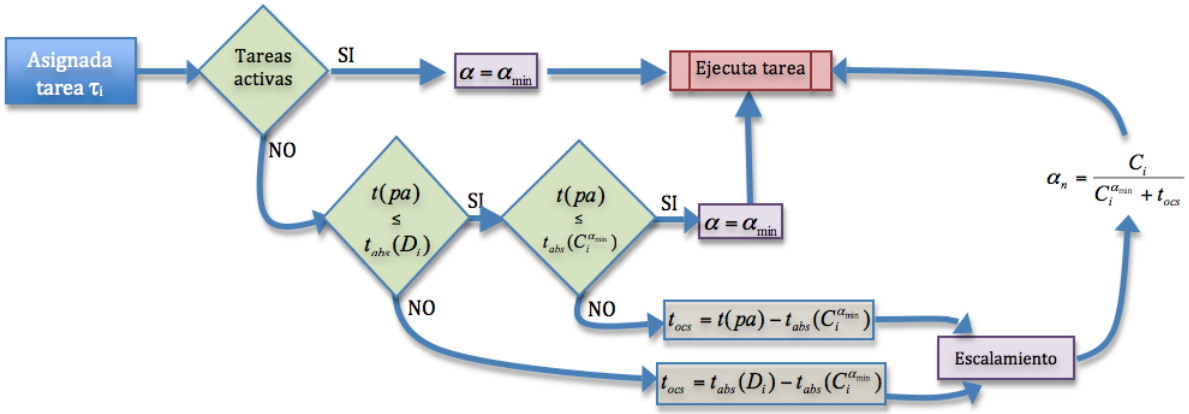

Figura 4.6: Algoritmo *rmit* cuando comienza a ejecutarse una nueva tarea

procesador. Cuando $t(pa) \leq t_{abs}(D_i)$ es invalida, el tiempo ocioso que puede aprovecharse será igual a:

$$t_{ocs} = t_{abs}(D_i) - t_{abs}(C_i^{\alpha_{min}}) \tag{4.1}$$

En caso de que $t(pa) \leq t_{abs}(D_i)$ sea valido, pero no se cumpla que $t(pa) \leq t_{abs}(C_i^{\alpha_{min}})$, el tiempo libre aprovechable será igual a:

$$t_{ocs} = t(pa) - t_{abs}(C_i^{\alpha_{min}}) \tag{4.2}$$

Utilizando este tiempo ocioso, el cálculo de la **nueva frecuencia** de procesador será igual a:

$$\alpha_n = \frac{C_i}{C_i^{\alpha_{min}} + t_{ocs}} \tag{4.3}$$

Donde C_i es el tiempo de ejecución de peor caso sin escalar (a la máxima frecuencia del procesador), y $C_i^{\alpha_{min}}$ es el tiempo de ejecución de peor caso escalado a la frecuencia de operación mínima constante calculada con ayuda del algoritmo presentado en el capítulo anterior.

Demostración. Suponiendo que se ha detectado un tiempo ocioso futuro que puede ser aprovechado por la tarea i, el nuevo tiempo de ejecución deseado para τ_i deberá incrementarse en ese valor de t_{ocs} (esto es $C_i^{\alpha_{min}} + t_{ocs}$). Para obtener ese valor es necesario escalar el tiempo de ejecución de peor caso actual, que es $C_i^{\alpha_{min}}$, por un nuevo valor α_n. El tiempo de ejecución de peor caso actual deberá normalizarse al valor inicial de τ_i antes de que fuera escalado. Esto último se consigue multiplicando ese valor por α_{min}. Esto es:

$$C_i^{\alpha_{min}} + t_{ocs} = \frac{C_i^{\alpha_{min}}}{\alpha_n}.\alpha_{min}$$

Despejando α_n de esta ecuación tenemos:

$$\alpha_n = \frac{C_i^{\alpha_{min}}}{C_i^{\alpha_{min}} + t_{ocs}}.\alpha_{min} = \frac{C_i}{C_i^{\alpha_{min}} + t_{ocs}}$$

Una vez terminadas estas comprobaciones y posibles cambios se procede a la ejecución de la tarea τ_i.

Con la finalización de alguna de las tareas del conjunto $\mathcal{T}$ arranca la segunda fase del algoritmo.

4.6.2. Método *rmit* — Terminación

La optimización del consumo de esta fase se enfoca en el final de ejecución de las tareas. Al igual que la fase anterior, esta parte se integra de forma transparente en el esquema del planificador sin necesidad de incorporar modificaciones excepcionales en la secuencia principal del planificador ni en la estructura de datos de las tareas. En la figura 4.7 se representa el funcionamiento de esta primera fase mediante un diagrama de flujo.

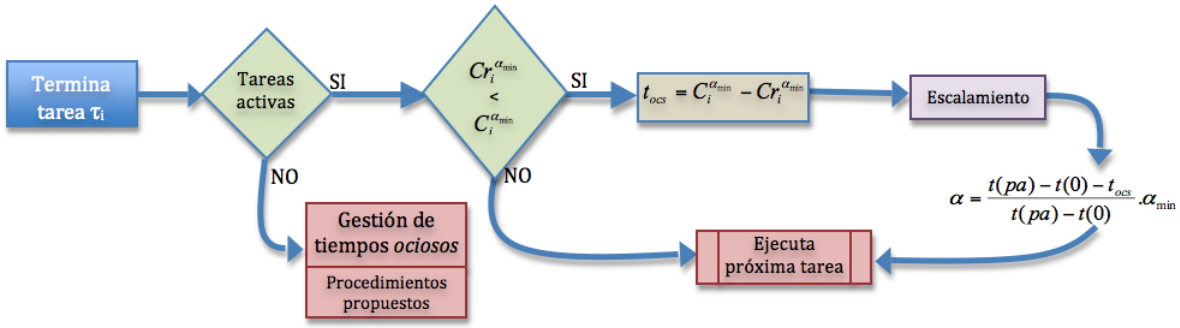

Figura 4.7: Algoritmo *rmit* cuando termina la ejecución de una tarea

Cuando una tarea τ_i termina su ejecución o es expulsada de la planificación, se comprueba si hay tareas activas esperando por el procesador. En caso de que **no** hayan tareas en la cola de ejecución del planificador, se procederá a realizar una gestión de tiempos ociosos basado en los algoritmos propuestos en las figuras 4.4 o 4.5. Pero en caso de que **sí** hayan tareas en la cola de espera, entonces se comparará el tiempo que realmente ha estado en ejecución la tarea τ_i (tiempo de ejecución real $Cr_i^{\alpha_{min}}$) con el tiempo de ejecución del peor caso $C_i^{\alpha_{min}}$. Si $Cr_i^{\alpha_{min}}$ **no** es menor que $C_i^{\alpha_{min}}$, entonces se continuará con la ejecución de la siguiente tarea sin ninguna modificación, ya que no hay tiempo extra disponible. Pero si por el contrario, $Cr_i^{\alpha_{min}}$ **sí** es menor que $C_i^{\alpha_{min}}$, entonces se calculará el tiempo no consumido por τ_i y que puede ser aprovechado por otras tareas en ejecución. Este tiempo ocioso es igual a:

$$t_{ocs} = C_i^{\alpha_{min}} - Cr_i^{\alpha_{min}} \tag{4.4}$$

Utilizando este tiempo ocioso, el cálculo de la ***nueva frecuencia*** de procesador será igual a:

$$\alpha_n = \frac{t(pa) - t(0) - t_{ocs}}{t(pa) - t(0)} . \alpha_{min} \tag{4.5}$$

Donde $t(pa)$ es el tiempo de la próxima activación programada. $t(0)$ corresponde al tiempo actual. t_{ocs} es el tiempo sobrante en la última ejecución de la tarea i y α_{min} es la frecuencia mínima constante de procesador, frecuencia con la que se ha calculado que el sistema será planificable en condiciones de peor caso.

Demostración. Suponiendo que se ha detectado un tiempo sobrante como resultado de la última ejecución de la tarea i, ese tiempo adicional deberá incrementarse en el tiempo de cómputo del resto de tareas que se encuentren ya en ejecución entre el tiempo actual $(t(0))$ y la próxima activación $(t(pa))$. Por tanto, suponemos ese intervalo de tiempo como si fuera la ejecución de una única tarea y esperamos a que tenga un tiempo de ejecución igual al intervalo: $t(pa) - t(0)$. Cuyo intervalo de cómputo también incluirá este tiempo extra con el que no se contaba previamente. Pero para incluir ese tiempo extra hay que escalar el procesador durante ese intervalo. Para conseguir ese nuevo factor de escalamiento tenemos:

$$C_d = \frac{C_p^{\alpha_{min}}}{\alpha_n} . \alpha_{min}$$

Donde el tiempo de cómputo deseado (C_d) se obtendrá de aplicar un factor de escalado al tiempo de computo previo $C_p^{\alpha_{min}}$ y normalizado por la frecuencia mínima α_{min}. Que se traducirá en:

$$(t(pa) - t(0) = \frac{t(pa) - t(0) - t_{ocs}}{\alpha n}.\alpha_{min}$$

Donde el nuevo cómputo deseado es igual a $t(pa) - t(0)$, y el tiempo de cómputo previamente previsto para el peor caso es $(t(pa) - t(0))$ menos el tiempo ocioso t_{ocs} obtenido.

Despejando αn de la ecuación anterior se obtendrá la ecuación 4.5.

4.7. Ejemplo de la variación de la frecuencia de procesador en tiempo de ejecución

A modo de ilustración, en esta sección se presenta una serie de ejemplos que muestran la aplicabilidad y el impacto de variar la frecuencia del procesador en tiempo de ejecución para aumentar el ahorro energético. Sin embargo, es en el capítulo 5 donde se llevará a cabo la evaluación de los algoritmos presentados en este capítulo y donde se compararan con otros presentes en la literatura.

4.7.1. Ejemplo 1: Recuperación de instantes ociosos

Para este primer ejemplo, se propone un escenario con un conjunto de tareas como el de la tabla 4.6.

T	D	C
10	10	2
15	15	3
18	15	4

Tabla 4.6: Características temporales del conjunto de tareas para el escenario 1

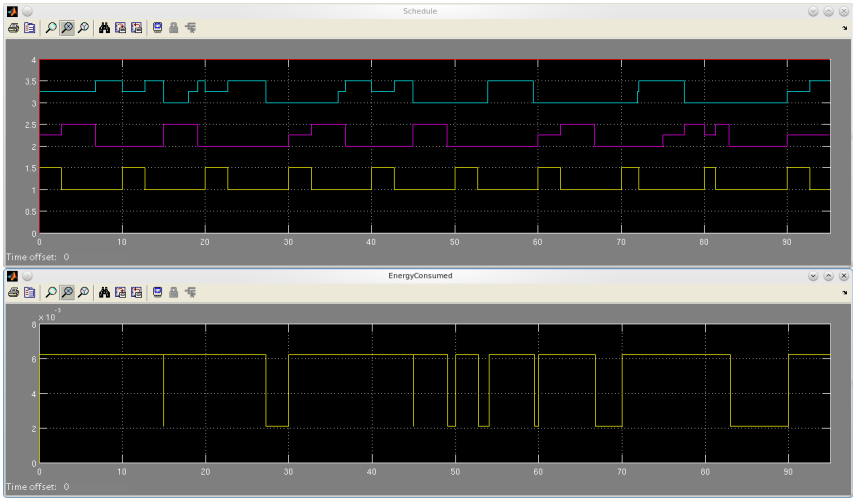

Figura 4.8: Escenario 1: Planificación y consumo utilizando una frecuencia de procesador mínima y constante

Este conjunto de tareas tiene una utilización $U = 62\,\%$. Utilizando el método $\mathcal{A}$ para el cálculo de la frecuencia mínima constante, obtenemos un $\alpha_{min} = 0{,}733$. Aplicando este factor de escalado, la utilización del sistema sube hasta $U^{\alpha_{min}} = 84{,}85\,\%$.

En la figura 4.8 y 4.9 se presenta la planificación del conjunto de tareas en el hiper-período, el cual es igual a 90. En este ejemplo, se supondrán que los tiempos de ejecución (C_i) de las tareas serán siempre constantes, excepto en las activaciones 8 y 9 de la tarea $i = 1$, en donde C_1 será $1{,}54$ y 1 respectivamente. Estas activaciones corresponden con los instantes de tiempo 70 y 80 de la tarea 1.

En cada figura aparecen dos ventanas, la ventana superior en las imágenes representa la planificación temporal de las tareas. La venta inferior representa el consumo de energía del procesador, cambios de nivel en los gráficos de esa ventana representan cambios en la frecuencia del procesador.

En la figura 4.8, el sistema utiliza una frecuencia mínima constante durante toda la ejecución. En los tiempos ociosos el procesador se lleva a estado "*idle*" desde la frecuencia mínima.

En la figura 4.9, la ejecución del sistema utiliza una frecuencia mínima constante, pero utiliza además el método *rmit* presentado en este capítulo, para la recuperación dinámica de los tiempos ociosos.

En la ventana de consumo de energía de la figura 4.8, se puede observar que el consumo de energía se mantiene de forma constante y este sólo disminuye cuando coincide con tiempos libres en la planificación de las tareas.

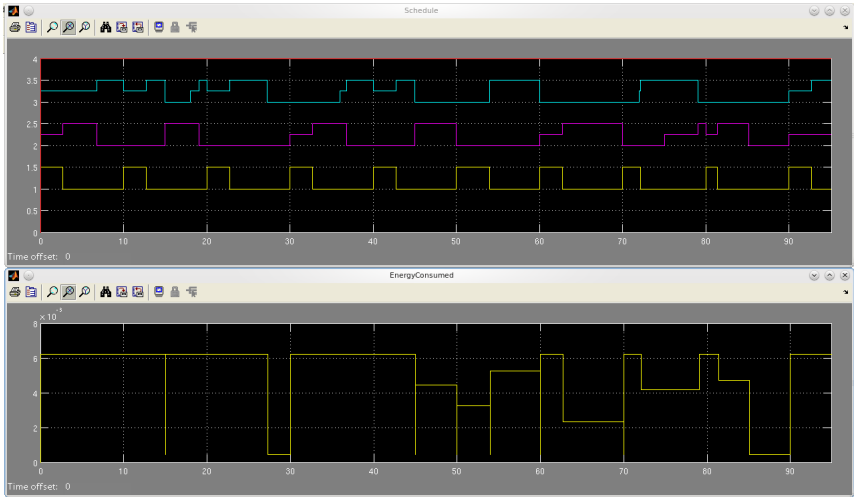

Figura 4.9: Escenario 1: Planificación y consumo utilizando el algoritmo propuesto *rmit*

Sin embargo, en la ventana de consumo de energía de la figura 4.9, se puede observar que la frecuencia de procesador cambia varias veces intentando recuperar los tiempos libres del procesador. Los cambios de frecuencia en el intervalo de tiempo entre 45 y 70 son debidos a la fase de "*Arranque*" del método *rmit*. Las reducciones de frecuencia entre los rangos 70 y 85 son debidos a la fase de "*Terminacion*" del método *rmit*.

4.7.2. Ejemplo 2: Consumo energético

En esta sección se presenta un ejemplo centrado en el consumo energético. En la tabla 4.7 se presenta el escenario a analizar.

T	D	C
45	23	7
50	36	10
60	38	9
120	98	29

Tabla 4.7: Características temporales del conjunto de tareas para el escenario 2

Este conjunto de tareas tiene una utilización $U = 74{,}72\,\%$. Utilizando el método $\mathcal{A}$ para el cálculo de la frecuencia mínima constante, obtenemos un $\alpha_{min} = 0{,}8980$. Aplicando este factor de escalado, la utilización del sistema sube hasta $U^{\alpha_{min}} = 83{,}21\,\%$.

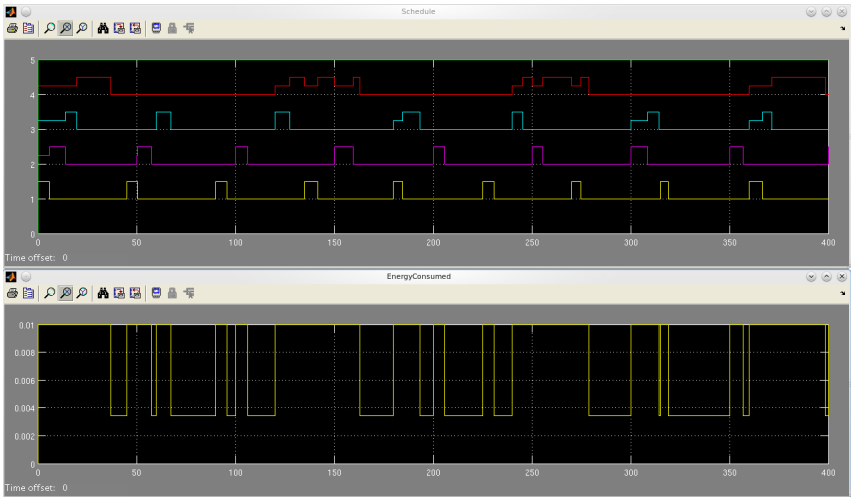

Figura 4.10: Escenario 2: Planificación y consumo en un sistema en el que no se hace ningún tipo de gestión de energía.

En la figura 4.10, 4.11 y 4.12 se presenta la planificación del conjunto de tareas de la tabla 4.7 en una parte de su hiper-período, esto con el fin de ver en más detalle la evolución de consumo. En la figura 4.13 se mostrará la planificación y consumo usando el algoritmo *rmit* para el conjunto de tareas en todo el hiper-período, el cual es igual a 1800. En este ejemplo, los tiempos de ejecución (C_i) de las tareas varían entre un $0\,\%$ y un $50\,\%$, por tanto, en algunas activaciones C_i no tendrá variaciones y en otras podrá variar máximo hasta la mitad de su tiempo de ejecución.

Al igual que en el ejemplo anterior, en cada figura aparecen dos ventanas, la ventana superior en las imágenes representa la planificación temporal de las tareas. La venta inferior representa el consumo de energía del procesador, cambios de nivel en los gráficos de esa ventana representan cambios en la frecuencia del procesador.

En la figura 4.10, la planificación del sistema no implementa ningún mecanismo de gestión energética y se utiliza la frecuencia máxima de operación. En los momentos ociosos el procesador se lleva a estado "*idle*" desde la frecuencia máxima.

En la figura 4.11, el sistema utiliza una frecuencia mínima constante durante toda la ejecución. En los tiempos ociosos el procesador se lleva a estado "*idle*" desde la frecuencia mínima.

En la figura 4.12, la ejecución del sistema utiliza una frecuencia mínima constante, pero utiliza además el método *rmit* presentado en este capítulo, para la recuperación dinámica de los tiempos ociosos.

En la ventana de consumo de energía de la figura 4.10 y 4.11, se puede observar que el consumo de energía se mantiene constante y este sólo disminuye cuando coincide con tiempos libres en la planificación de las tareas.

Sin embargo, en la ventana de consumo de energía de la figura 4.13, se puede observar que la frecuencia de procesador cambia varias veces intentando recuperar los tiempos libres del procesador. Estos cambios son debidos a ambas fases del algoritmo *rmit*: "*Arranque*" y "*Terminacion*".

Finalmente, en la figura 4.13, se presenta el consumo completo para el conjunto de tareas de la tabla 4.7. La evolución en la planificación y el consumo de energía de las tareas es mostrado para el hiper-período (1800).

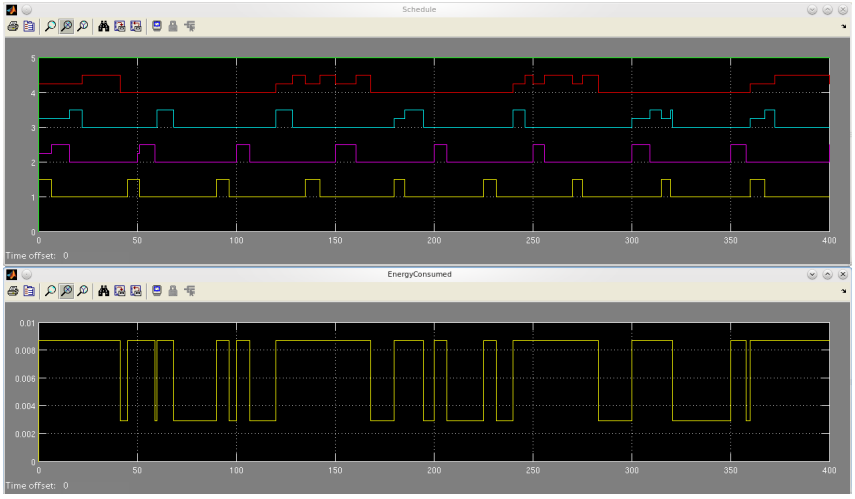

Figura 4.11: Escenario 2: Planificación y consumo utilizando una frecuencia de procesador mínima y constante

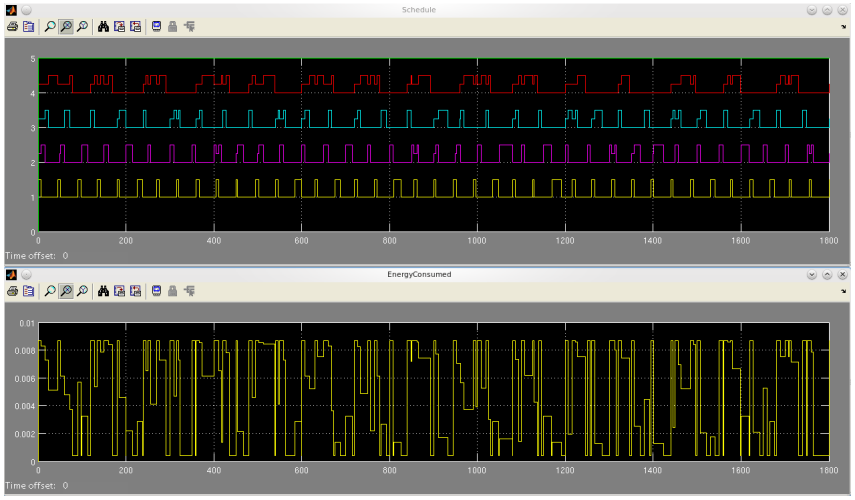

Figura 4.13: Escenario 2: Planificación y consumo utilizando el algoritmo propuesto *rmit* en todo el hiper-período

4.7.3. Conclusiones

Centrándonos en el ejemplo 2, si se compara el consumo total alcanzado con el algoritmo *rmit* respecto a los otros dos casos en el hiper-período, obtendremos los siguientes resultados:

- Utilizando la frecuencia máxima se obtiene un sobre consumo igual al $46,59\,\%$
- Utilizando la frecuencia mínima constante se obtiene un sobre consumo igual al $34,02\,\%$

El ahorro conseguido utilizando el algoritmo de gestión energética *rmit* es de casi el doble con respecto a un sistema con el mismo conjunto de tareas pero sin gestión energética.

Observando estos resultados, también se puede llegar a la conclusión de que usando únicamente la frecuencia mínima constante no se garantiza necesariamente una gestión óptima del gasto energético. En el ejemplo, la diferencia entre el consumo con frecuencia máxima y mínima, únicamente fue del $9\,\%$.

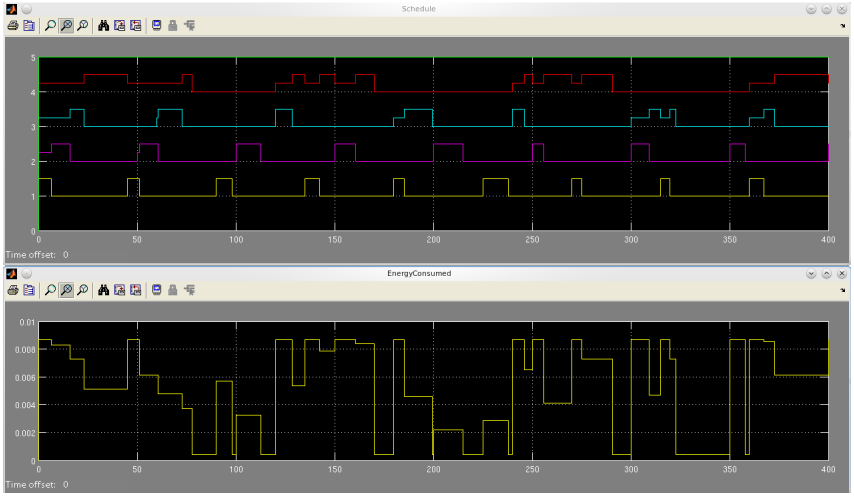

Figura 4.12: Escenario 2: Planificación y consumo utilizando el algoritmo propuesto *rmit*

Los resultados de consumo energético totales son dependientes de las características de cada conjunto de tareas analizado. Por tanto, los porcentajes presentados son específicos para el escenario planteado, para otros conjuntos de tareas estos diferenciales de consumo podrían ser mayores o menores.

Capítulo 5

EVALUACIÓN DE ALGORITMOS DE ESCALADO DE FRECUENCIA

5.1. Evaluación experimental cuando $D = T$ y $D \leq T$ - Frecuencia estática

En esta sección se propone una serie de simulaciones para el algoritmo propuesto (método $\mathcal{A}$) junto con los algoritmos presentados (métodos $\mathcal{P}$, RTA, LLM, HB, LL y $EDF - U$). El objetivo principal es evaluar experimentalmente la eficiencia de los algoritmos en la delegación dinámica de código en escenarios con conjuntos de tareas con $D = T$ y $D \leq T$.

Para la construcción de tales escenarios, a continuación se propone un método para la generación de perfiles dinámicos de carga. La formación de estos perfiles esta basado en características básicas de tiempo real, tales como: períodos, plazos de ejecución, utilización de CPU, etc.

5.1.1. Método para la generación de perfiles de cargas dinámicas

El objetivo principal de este método es automatizar la creación de perfiles de cargas, basado en la generación de conjuntos de tareas a partir de una serie de parámetros de entrada. Esta automatización es recopilada en una librería de Matlab. Esta librería tiene una interfaz genérica que no es dependiente de los algoritmos presentados en esta tesis, por lo que puede ser usada de forma independiente en otros tipo de evaluaciones.

Para la generación de estos perfiles se definen tres grupos de tareas aleatorios: A, B y C. El grupo A contiene tareas con períodos cortos, el grupo B contiene tareas con períodos intermedios, y el grupo C esta compuesto de tareas con períodos largos. Cada grupo de tareas consiste de 20 tareas. El intervalo de los períodos de tareas es mostrado en la tabla 5.1.

Grupo	Cota inferior	Cota superior
A	$2000\mu s$	$40000\mu s$
B	$40001\mu s$	$600000\mu s$
C	$600001\mu s$	$4000000\mu s$

Tabla 5.1: Intervalo de períodos para grupos de tareas A, B y C.

Adicionalmente a estos intervalos de períodos, se han definido tres tipos de llegada de tareas ($Ll1$, $Ll2$ y $Ll3$) que serán utilizados en la simulación. Para el conjunto $Ll1$ las tareas de mayor prioridad llegaran primero, en $Ll2$ las tareas con prioridad media llegaran antes y en el grupo $Ll3$ las tareas de menor prioridad serán las primeras en llegar. Cada grupo de llegada consta de un conjunto de 20 tareas.

Estos perfiles de carga tratan de representar tareas con características temporales usadas típicamente en aplicaciones reales de control, tales como en el control de robots móviles y humanoides. En

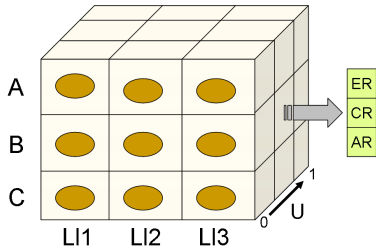

Figura 5.1: Conjunto de pruebas para el análisis de los métodos para el cálculo del factor de escalado de frecuencia.

este tipo de aplicaciones generalmente convergen varios niveles jerárquicos de control. Estos niveles jerárquicos están compuestos por conjuntos de tareas con características temporales diferentes, que dependen de la criticidad de las actividades a llevarse acabo. Por ejemplo, en un sistema robótico jerárquico, con niveles reactivos, tácticos y de misión, la asignación de tareas es bastante semejante a la distribución de los grupos de tareas propuestos. El primer nivel (nivel reactivo) esta constituido principalmente por tareas de prioridad alta, i.e. tareas con períodos cortos, tal como el Grupo A. El nivel táctico puede compararse con el Grupo B, el cual esta formado por tareas con períodos más relajados, con criticidad media. En el nivel de misión las restricciones temporales son reducidas. En este nivel las tareas usadas son las menos prioritarias y con períodos grandes, análogamente con el grupo C.

Los grupos de tareas A, B y C se combinan con los tipos de llegada $Ll1$, $Ll2$ y $Ll3$, y esto resulta en 9 conjunto de tareas distintos. Estos a su vez serán evaluados con 5 utilizaciones U discretas (0.3 - 0.5 - 0.7 - 0.8 - 0.95). Dando lugar a 45 conjuntos con 20 tareas cada uno.

5.1.2. Contexto de evaluación

Para este análisis, cada una de las pruebas se caracterizará según el grado de aceptación **AR**, coste computacional **CR** y consumo energético **ER** (ver figura 5.1). Los resultados de estas simulaciones serán presentados sobre tres componentes de referencia:

- El *coste computacional* de cada algoritmo es calculado de acuerdo con el número y tipo de operaciones (suma, resta, multiplicación, división o potencia,etc.). Cada operación es caracterizado con un número predeterminado de ciclos máquina y basado en la arquitectura ARM (Sloss et al., 2004). Por consiguiente, las unidades en las cuales el coste computacional es medido son ciclos máquina.

- El componente de *sobre-consumo de energía* se refiere al exceso de consumo de energía de un determinado algoritmo debido a desviaciones en el cálculo de la frecuencia de operación mínima en comparación con la frecuencia calculada por un algoritmo exacto para prioridades fijas.

- El *grado de rechazo* hace referencia a 1 menos la relación entre el número de tareas que un determinado método predice como planificable $\mathcal{T}_{pt}$ y el número total de tareas que realmente es planificable utilizando una algoritmo necesario y suficiente $\mathcal{T}_p$: $1 - (\mathcal{T}_{pt}/\mathcal{T}_p)$.

Estas componentes fueron escogidas de tal forma que una mayor magnitud en los ejes puede entenderse como un peor comportamiento del algoritmo. En las pruebas, el resultado de peor caso es dibujado para cada grupo de tareas y tipo de llegada. El cálculo de α usando el método presentado $\mathcal{S}$ (1.2.5,3.4.6) no fue usado debido a los altos costes computaciones involucrados.

En el análisis energético únicamente se refleja el consumo causado por la ejecución del conjunto completo de tareas en las simulaciones. Este consumo depende de la frecuencia de procesador obtenida para cada algoritmo. El consumo energético derivado de la ejecución del propio algoritmo no es incluido en el consumo global presentado en las simulaciones. Esto es porque este consumo depende de la frecuencia a la que el procesador esta funcionando en el instante en que el algoritmo requiere

ser ejecutado. Por tanto, se ha considerado más apropiado comparar el coste de los algoritmos en unidades independientes de la frecuencia de procesador.

Esto considera que el número de ciclos máquina es proporcional al consumo energético. Protocolos de cambio de modo deberían establecer la frecuencia del procesador a la cual el algoritmo debería llevarse a cabo cuando el cálculo de una frecuencia nueva es requerido. Sin embargo, estos protocolos no están dentro del objetivo de esta tesis.

5.1.3. Grado de rechazo en comparación con coste computacional

Las figuras 5.2 y 5.3 muestran el resultado de simulación para el porcentaje de rechazo de tareas en comparación con el coste computacional en el cálculo de algoritmos. Note que en las figuras el porcentaje de rechazo es dibujado entre valores de 0 y 0.6.

En las figuras se puede ver que exceptuando los métodos LLM, LL y HB, todos los algoritmos para el cálculo de α tienen un grado de rechazo igual a 0 %. El método LL tiene a grado de rechazo que se incrementa con la utilización y el cual se extiende desde 0 % hasta cerca del 45 %. Para el caso del método HB el rechazo es ligeramente inferior al de LL. Cuando $D \leq T$, el grado de rechazo máximo para el método LLM se incrementa hasta cerca del 50 %. El planificador basado en EDF-U tiene un comportamiento igual que LLM.

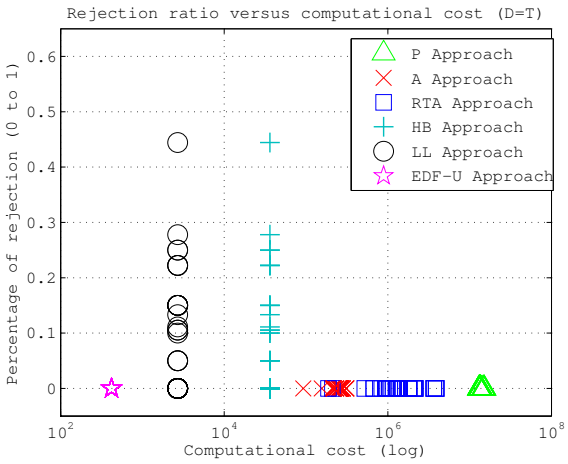

Figura 5.2: Grado de rechazo en función del coste computacional cuando $D = T$.

Desde el punto de vista de coste,aunque los métodos EDF-U (cuando $D \leq T$), LLM, LL, y HB tienen mayores porcentajes de rechazo, en contrapartida estos algoritmos tienen costes computacionales bajos. En ambos casos, $D = T$ y $D \leq T$, el coste computacional utilizando el método RTA es bastante disperso y impredecible, y su coste es de los más altos en la mayoría de los casos comparado con todos los métodos evaluados, excedido únicamente por el método $\mathcal{P}$.

En la evaluación, el método $\mathcal{A}$ tiene un coste intermedio y un grado de rechazo de 0 para $D = T$ y $D \leq T$. Además, el método propuesto tiene un coste predecible, y su coste computacional es casi 20 veces menor que el coste para $\mathcal{P}$, aún teniendo el mismo grado de rechazo. Note que el hueco en el coste entre el método propuesto y el método $\mathcal{P}$ se incrementa con la llegada de nuevas tareas.

5.1.4. Porcentaje de sobre-consumo contra coste computacional

En las figuras 5.4 y 5.5 se muestra el sobre consumo energético en función del coste computacional para el peor caso. El sobre-consumo es debido a desviaciones en el calculo del factor de escalado α en comparación con el calculo usando un método exacto. Hay que tener en cuenta que desviaciones en α equivalen a sobre consumos cuadráticos – dependiendo de la expresión polinomial que describa el consumo de potencia de CPU.

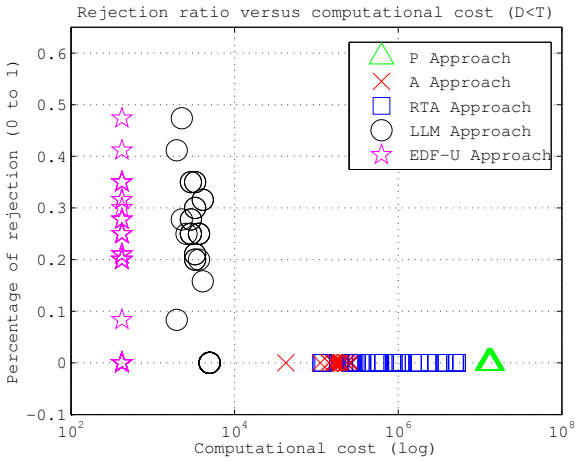

Figura 5.3: Grado de rechazo en función con el coste computacional cuando $D < T$.

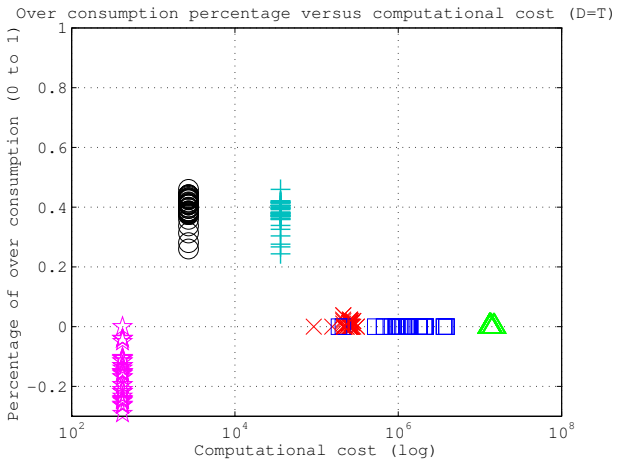

Figura 5.4: Porcentaje de sobre consumo en función del coste computacional cuando $D = T$.

En la figura 5.4 se puede observar que LL tiene un coste computacional pequeño, pero un sobre consumo cercano al 45 %. Cuando $D = T$, EDF-U ahorra más energía que el resto de métodos. Por esta razón los puntos para EDF-U están localizados en valores negativos. HB tiene un sobre consumo ligeramente menor que LL. Cuando $D \leq T$, el método LLM muestra un sobre consumo para el peor caso entre el 30 % y 60 %. Cuando $D \leq T$, el método EDF-U tiene un comportamiento similar que LLM, pero este primero tiene un menor coste computacional.

Adicionalmente, se obtiene un coste computacional intermedio cuando se usa el método reducido $\mathcal{A}$. Cuando $D = T$, el método propuesto tiene una mayor concentración de puntos de peor caso de sobre-consumo alrededor de 0 %, y con algunos valores máximos por encima de 0 pero por debajo del 2.5 %. Cuando $D \leq T$, se obtiene un sobre consumo mayoritariamente igual a 0 usando el método $\mathcal{A}$. Un 2 % de los puntos dibujados para $\mathcal{A}$ están por encima de 0. Para ambos casos, $D = T$ y $D \leq T$, el método $\mathcal{P}$ y RTA muestran un sobre consumo igual a 0, pero con un coste computacional bastante alto y disperso para este último algoritmo.

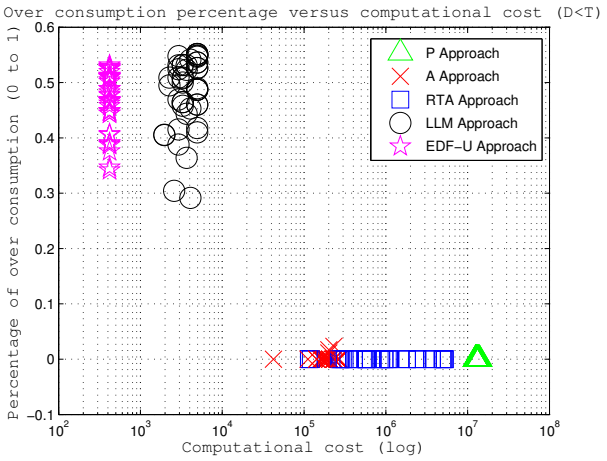

Figura 5.5: Porcentaje de sobre consumo en función del coste computacional cuando $D < T$

5.1.5. Grado de rechazo comparado con porcentaje de sobre-consumo

Las figuras 5.6 y 5.7 muestra el porcentaje de rechazo de tareas en comparación con el porcentaje de sobre consumo de energía.

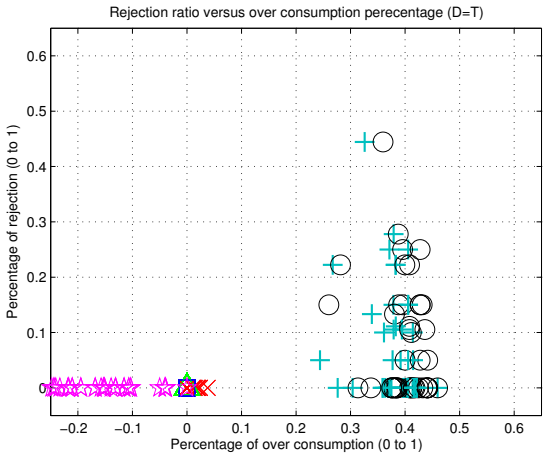

Figura 5.6: Grado de rechazo versus porcentaje de sobre consumo energético cuando $D = T$.

Cuando $D = T$, los puntos para los métodos LL y HB se dispersan en la intersección de las áreas comprendidas entre el 20 % y 48 % del eje de sobre consumo y el área entre 0 % y 45 % del eje de grado rechazo de tareas. El incremento del porcentaje en estos métodos esta determinado por la utilización y el número de tareas. El método $\mathcal{A}$ no tiene puntos en el eje de grado de rechazo de tareas y en el eje de sobre consumo, la mayoría de puntos están ubicados en 0, únicamente el 5 % de los puntos dibujados para $\mathcal{A}$ están entre 1.2 % y 2.5 %. El método EDF-U tiene un grado de rechazo igual a 0 % y el ahorro de energía de este método es mayor que el resto de métodos por prioridades fijas. Por tal razón, los puntos de EDF-U están localizados en la parte negativa del eje. Los otros métodos tiene un sobre consumo y grado de rechazo igual a 0.

Cuando $D \leq T$, el método LLM esta disperso en la intersección de las áreas entre 30 % y 55 % del sobre consumo y el área entre el 0 % y 50 % del eje de rechazo. El método EDF-U tiene un comportamiento igual a LLM. Los métodos RTA, $\mathcal{P}$ y $\mathcal{A}$ tienen un sobre consumo igual a 0 y muy cerca de 0 respectivamente y un grado de rechazo igual a 0.

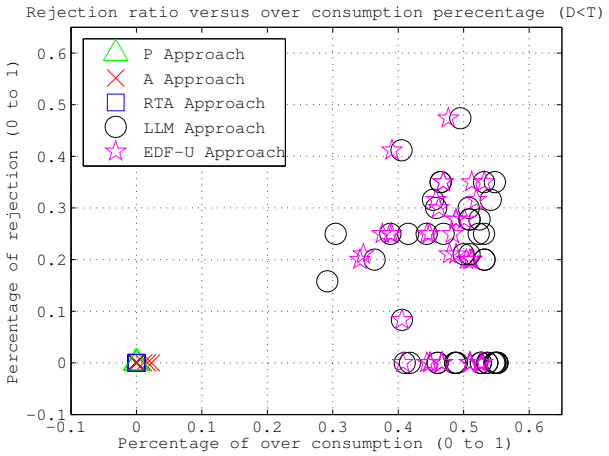

Figura 5.7: Grado de rechazo versus porcentaje de sobre consumo energético cuando $D < T$.

5.1.6. Otras simulaciones

En la figura 5.8 se muestra la evolución del coste computacional del algoritmo propuesto junto con otros algoritmos de prioridades fijas respecto a la llegada de tareas nuevas, para esta figura se han utilizado todos los resultados obtenidos en las pruebas llevadas a cabo cuando $D = T$. La figura muestra principalmente la variabilidad del coste para el método RTA, el cual varia dependiendo del conjunto de tareas y se incrementa con el número de llegada de tareas.

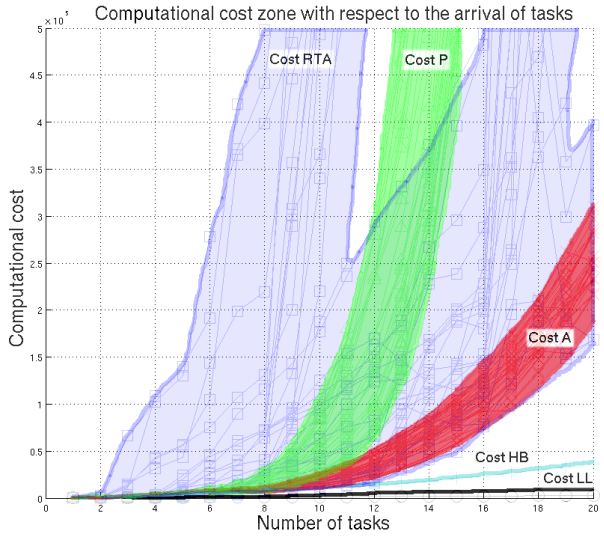

Figura 5.8: Evolución de el coste computacional del algoritmo respecto a la llegada de nuevas tareas al sistema

La figura 5.8 también muestra la evolución del método $\mathcal{P}$, cuyo coste se incrementa dependiendo del número de tareas en orden de 2^n, donde n es el número de tareas. El coste del algoritmo estático propuesto es más pequeño comparado con los algoritmos $\mathcal{P}$ y RTA, y su evolución con respecto al número de llegadas es más amortiguado que los métodos anteriormente mencionados.

En la figura 5.9 se muestra de forma independiente la evolución del ahorro de energía respecto al ahorro que se conseguiría con un algoritmo exacto, y esto en función de la llegada de tareas para cada

conjunto (A, B y C) y sus respectivas llegadas (LL1, LL2 y LL3). En la figura se utiliza una utilización del 95 % para los conjunto de tareas con $D = T$ y $D \leq T$.

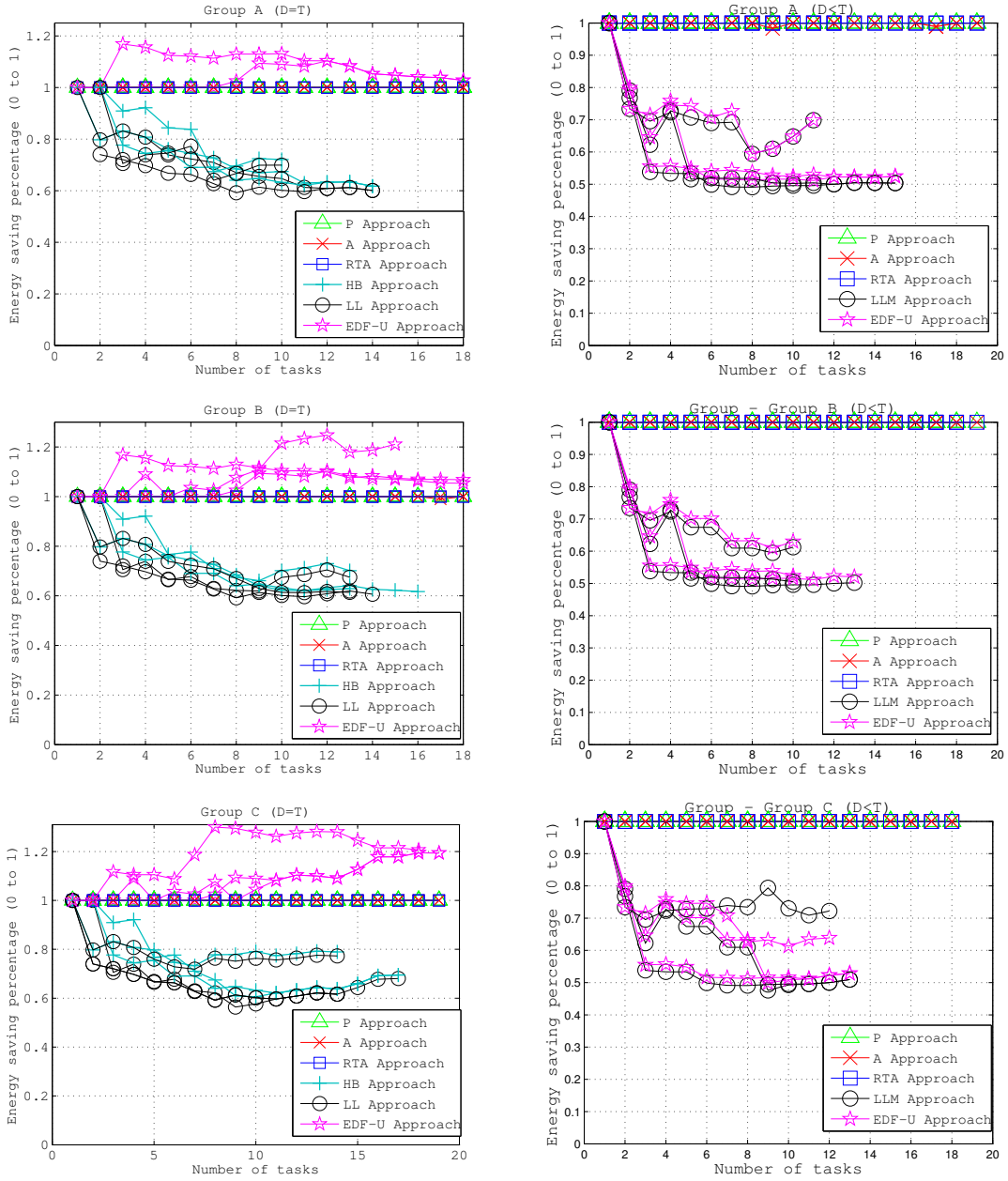

Figura 5.9: Evolución del ahorro de energía en comparación con el ahorro que se lograría con un algoritmo exacto, y esto en función de la llegada progresiva de tareas. Utilización igual a 95 %.

.

5.1.7. Conclusiones

En la sección 3.5 se propuso un nuevo algoritmo (método $\mathcal{A}$) para el cálculo de un factor de escalado de frecuencia de procesador para políticas de planificación basadas en prioridades fijas con tareas con $D = T$ y $D \leq T$.

En este capítulo se han presentado los resultados de extensivas simulaciones donde se compara

el comportamiento del método propuesto $\mathcal{A}$ con varios métodos de análisis de factibilidad para el cálculo del factor de escalamiento de frecuencia α cuando las tareas tienen $D = T$ y $D \leq T$. El comportamiento de los algoritmos ha sido analizado desde el punto de vista de consumo de energía, grado de rechazo y coste de computo real del algoritmo.

Los resultados de las simulaciones muestran que el algoritmo propuesto aventaja a otros algoritmos en el cálculo de la constante de escalado de frecuencia desde el punto de vista de coste computacional, predecibilidad y grado de aceptación de tareas. Sin embargo, se han observado pequeñas desviaciones en las pruebas de ahorro de energía. El método propuesto permite no únicamente la verificación con una gran precisión de la factibilidad del sistema, si no que además permite el cálculo del factor α, por lo que el algoritmo puede utilizarse como método de prueba de aceptación de tareas en tiempo de ejecución en entornos con cargas de procesamiento dinámicas. El algoritmo también puede ser usado en sistema operativos de tiempo real porque la sobrecarga derivada de la ejecución del algoritmo es reducida y predecible, y se minimiza el consumo de energía a la vez que se garantiza la no perdida de plazos de ejecución de tareas.

En cuanto a los resultados de las pruebas en comparación con el algoritmo EDF, es bien sabido que los algoritmos con esquemas por prioridades dinámicas tienen un mayor rendimiento que la planificación basada en prioridades fijas. Sin embargo, en muchos casos pueden haber restricciones de diseño que obliguen a la utilización de algoritmos por prioridades fijas, restricciones tales como las impuestas por estándares, por ejemplo, el estándar ARINC-653 (Interface, 2003).

Las principales características de los métodos analizados son presentadas en la tabla 5.2.

Alg.	Coste	Grado de rechazo	Grado de Sobre-consumo	D / T
$\mathcal{S}$	Muy Alto	Ninguno	Ninguno	$D \leq T$
$\mathcal{P}$	Alto	Ninguno	Ninguno	$D \leq T$
RTA	Impredecible Medio-Alto	Ninguno	Ninguno	$D \leq T$
$\mathcal{A}^{\dagger}$	Bajo-Medio	Ninguno	Muy bajo	$D \leq T$
LLM	Bajo	Muy Alto	Muy Alto	$D \leq T$
HB	Bajo	Alto	Alto	$D = T$
LL	Muy Bajo	Alto	Alto	$D = T$
$EDF - UAprox.$	Muy Bajo	Muy Alto	Muy Alto	$D \leq T$
$EDF - U$	Muy Bajo	Ninguno	Ninguno	$D = T$

Tabla 5.2: Comparación de los métodos para el cálculo del factor de escalamiento de frecuencia constante α. (†: Método propuesto $\mathcal{A}$).

5.2. Evaluación experimental cuando $D \leq T$ - Frecuencia dinámica

En esta sección se propone una serie de simulaciones para el algoritmo propuesto *rmit*. El objetivo principal es evaluar experimentalmente la eficiencia del algoritmos en su aplicación en tiempo de ejecución, en entornos con tareas con $D = T$ y $D \leq T$.

Para la construcción de tales escenarios, a continuación se propone un escenario para la generación de perfiles dinámicos de carga.

5.2.1. Métodos para la simulación

Se utilizará la herramienta de simulación True-Time (Ohlin et al., 2007), empleada en la simulación de múltiples aplicaciones de tiempo real tales como en (Cervin et al., 2007) y (Farias et al., 2007).

A esta herramienta se le ha realizado *algunas modificaciones* para poder simular *cambios de frecuencia en tiempo de ejecución* utilizando un análisis *en tiempo de ejecución* (ver figura 5.10). Se ha

adaptado para permitir el acceso a las colas de espera, tareas activas, tiempos de activación, plazos de ejecución, etc.

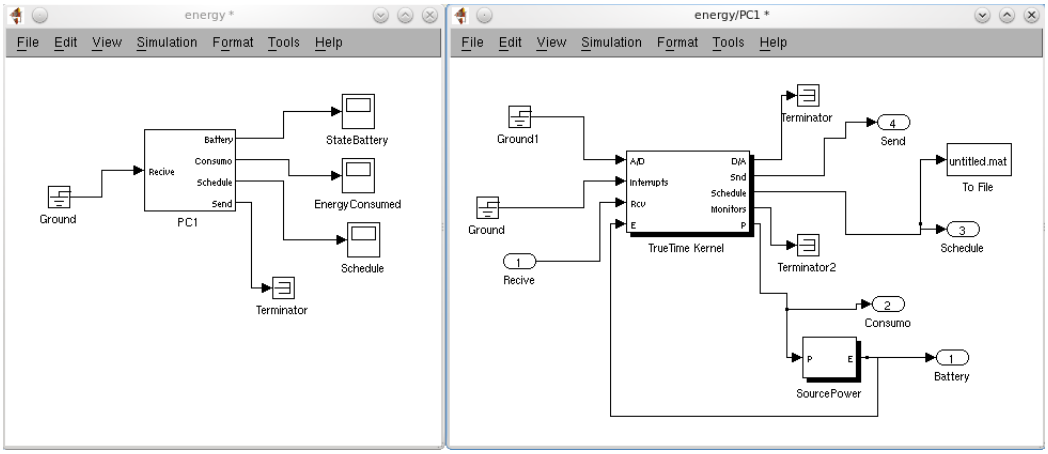

Figura 5.10: Escenario en simulink

5.2.2. Contexto de evaluación

Los grupos de tareas formados para la simulación siguen la misma configuración que la descrita en la sección 5.1.1 (ver figura 5.1). Para esta simulación se han utilizado tareas con $D < T$, con plazos D entre un 50 % y un 80 % del período T. Además, utilizaremos la curva de consumo de la figura 4.1, suponiendo consumos diferentes en función de si se esta en un modo activo o en un modo *idle*.

5.2.3. Evaluación

Lo primero que haremos es abordar el caso más sencillo, suponiendo que únicamente se presenta el primer tipo de tiempo ocioso propuesto, inherente en el escalado con esquemas de planificación RM y DM, donde supondremos que el tiempo de ejecución de las tareas en todas sus activaciones es el de peor caso. Los grupos de tareas formados para la simulación siguen la misma configuración que la descrita en la sección 5.1.1. Para esta simulación se han utilizado tareas con $D < T$, con plazos D entre un 50 % y un 80 % del período T. Además, utilizaremos la curva de consumo de la figura 5.1, suponiendo consumos diferentes en función de si se esta en un modo activo o en un modo *idle*.

Se han simulado 6 casos específicos: uno sin gestión de energía con la CPU al 100 % el cual será utilizado como referencia comparativa; otro con la CPU al 100 % pero con gestión de instantes idle, lo que equivaldrá a utilizar el *algoritmo 2* propuesto en la figura 4.5; otro método utilizando un α constante mínimo calculado con el método propuesto *Reducido* $\mathcal{A}$; otro igual que el anterior pero utilizando el *algoritmo 2* propuesto; otro utilizando el algoritmo *lpps* comentado con anterioridad; y finalmente, el algoritmo propuesto *rmit*.

En la figura 5.11 se muestra un conjunto de barras que comparan el porcentaje de consumo energético entre los casos propuestos con respecto al máximo consumo obtenido con la CPU al 100 %, y esto en función de determinadas utilidades. Para estas comparaciones se cálculo el consumo energético hasta el hiper-período de cada conjunto de tareas. Sin embargo, en la figura por motivos de representación, hemos gráficado únicamente los datos para los casos en el que la diferencia entre el máximo consumo y el consumo a velocidad constante es menor.

En la figura 5.11 se puede observar, que como es lógico, a medida que aumenta la carga del sistema la diferencia entre el consumo de los diversos métodos y el consumo máximo se hace menor, haciéndose el ahorro de energía más perceptible en utilizaciones bajas y medias. En el *primer conjunto de barras* con una utilidad del 35 %, la diferencia de consumos es bastante grande. Con el método *sin DVS*, es decir a máxima frecuencia, pero utilizando simplemente el algoritmo propuesto en la figura

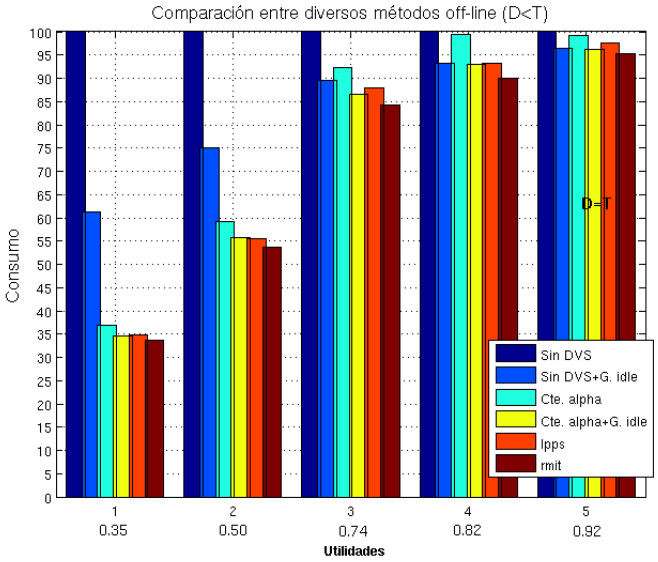

Figura 5.11: Comparación del ahorro de energía entre varios métodos on-line

4.5 para la gestión de tiempos *idle*, se puede llegar ahorrar hasta un 40 % de energía. Utilizando una frecuencia constante mínima se puede ahorrar hasta un 63 % de energía, pero si le aplicamos además el *algoritmo 2* el ahorro de energía se incrementa un 3 % hasta llegar a un 66 %, cuyo ahorro resulta interesante si se compara con el ahorro obtenido por el algoritmo *lpps* el cual es menor con un porcentaje del 65.5 %. Mientras que con el algoritmo propuesto *rmit* se obtiene el mayor ahorro de energía entre todos los métodos, siendo este del 67 %. En el *segundo conjunto de barras* con una utilidad del 50 %, el mayor ahorro de energía se consigue igualmente con el método propuesto *rmit* con un 53 % de ahorro, siendo este un 7 % menos que el ahorro conseguido utilizando únicamente la frecuencia mínima constante y un 2 % menos que el conseguido con el método *lpps*, cuyo ahorro en este caso es de 0.25 % superior al del "método con frecuencia mínima constante+*algoritmo 2*". En el *siguiente conjunto de barras* con utilidad del 74 %, la diferencia en el ahorro de energía entre el método *rmit* y el resto de métodos se hace mayor, ahorrando un 8 % más que el método con α constante mínimo y un 4 % menos que el método *lpps*. En esta prueba se pueden observar *dos resultados interesantes*, el primero, que el ahorro conseguido utilizando la máxima frecuencia pero aplicando el algoritmo propuesto para la gestión de instantes *idle* (*algoritmo 2*), es mayor que el conseguido por la utilización de un α constante mínimo, y el segundo, que el ahorro utilizando el "método con frecuencia mínima constante+*algoritmo 2*" es mayor que el conseguido con el método *lpps*. En el *cuarto conjunto de barras* con una utilidad del 82 %, la reducción de consumo obtenida con el método *rmit* en relación con los otros métodos es aún mayor que con las utilidades anteriormente mencionadas, llegando a un 9 % la diferencia entre el método *rmit* y el método con α constante mínimo, y un 4 % con los otros tres métodos. Para esta utilidad, el ahorro energético utilizando el método con frecuencia mínima constante+*algoritmo 2*, el método utilizando la máxima frecuencia pero aplicando el *algoritmo 2*, y el método *lpps*, ahorran práctiamente la misma cantidad de energía estando diferenciados por -0.11 % y -0.05 % respectivamente. En este caso, también resulta interesante que el consumo obtenido con el α constante mínimo es 6 % mayor que el obtenido utilizando la máxima frecuencia pero aplicando el algoritmo propuesto en la figura 4.5. Para este *último caso*, con una utilización del 92 %, se ha cambiado la suposición de $D < T$ por $D = T$, y se puede comprobar que el comportamiento relacionado con la diferencia entre los ahorros de energía de los algoritmos propuestos es el mismo que el expuesto con anterioridad. El método *rmit* sigue siendo el que más ahorro presenta, seguido por el método con α mínimo constante+*algoritmo 2*, después se encuentra el método que se ejecuta a la máxima frecuencia pero aplicando el algoritmo de gestión de tiempos *idle*, seguido por el método *lpps* y finalmente se encuentra el método que utiliza únicamente un α constante.

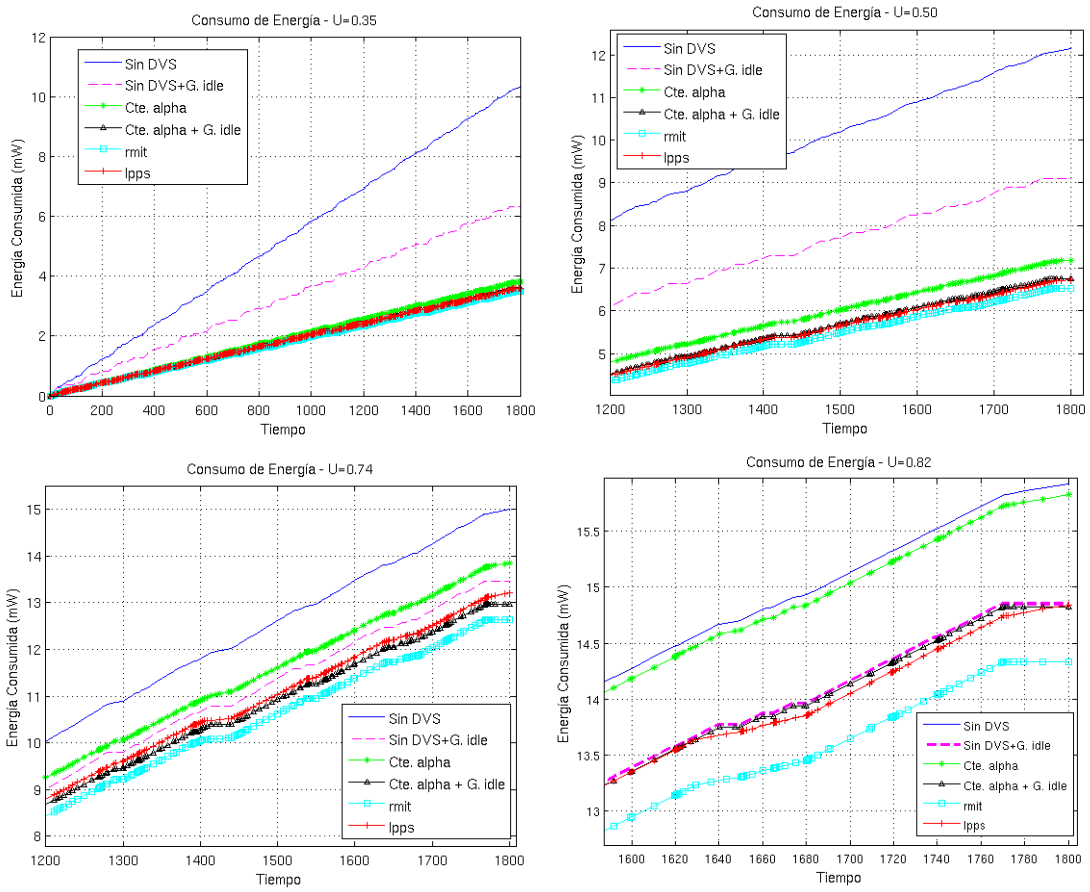

Figura 5.12: Curvas de consumo energético para diversos métodos *on-line* evaluados en varias utilidades para un conjunto de tareas con $D < T$

En la figura 5.12 se muestra el perfil de consumo energético al final del hiper-período (H) de un conjunto de tareas con $D < T$, evaluado en diferentes utilidades. Al realizarse a lo largo de H el valor final de potencia en la curva se corresponderá con el consumo resultante total de energía para el conjunto de tareas. En la figura se puede observar que para todos los casos, la curva de consumo del método propuesto *rmit* se encuentra siempre por debajo de la de los otros métodos. Además se puede observar que a medida que aumenta la utilidad en el conjunto de tareas las curvas tienden a juntarse en tres grupos destacados de consumo: el primero, en donde se presenta un mayor consumo, esta formado por las curvas resultantes de la ejecución a la máxima frecuencia y a una frecuencia constante mínima; el segundo grupo esta formado, en orden de mayor a menor consumo, por el método *lpps* y los métodos que utilizan el algoritmo de gestión idle propuesto; y el tercer grupo con menor consumo esta formado por el método *rmit*, cuya curva de consumo se va separando progresivamente del resto de algoritmos a medida que aumenta la utilización del sistema.

El siguiente paso es evaluar los diversos métodos *on-line* mediante la variación aleatoria de los tiempo de ejecución. De tal modo que podamos observar el ahorro de energía efectivo de estos métodos frente a dichas variaciones, ya que aunque el análisis se lleve a cabo utilizando los tiempos de ejecución de peor caso (WCET), en la práctica el radio entre BCET/WCET para diversos tipos de tareas puede variar ampliamente. Por tanto para simular estas variaciones del tiempo de ejecución de las tareas, las cuales *no son siempre iguales para todas las tareas*, proponemos utilizar intervalos en los cuales pueden escalarse los tiempos de ejecución por radios BCET/WCET aleatorios dentro de los límites del intervalo.

En la figura 5.13 se muestra una comparación entre diversos métodos *on-line* utilizando variaciones

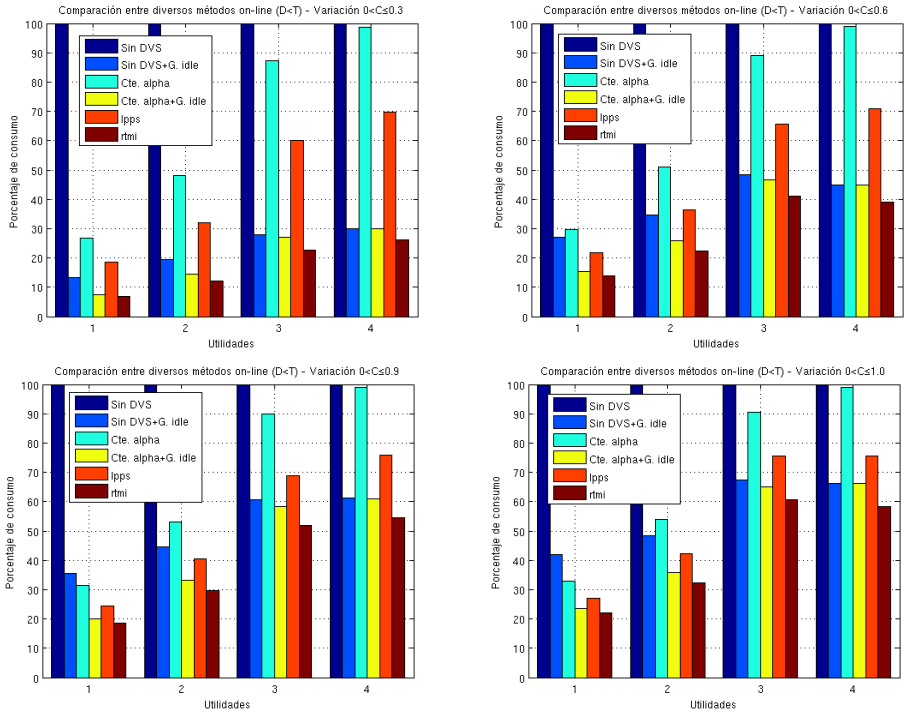

Figura 5.13: Comparación del ahorro de energía entre varios métodos on-line variando aleatoriamente el radio BCET/WCET

aleatorias de BCET/WCET en intervalos de $0 < BCET/WCET \leq 0{,}3$, $0 < BCET/WCET \leq 0{,}6$, $0 < BCET/WCET \leq 0{,}9$ y $0 < BCET/WCET \leq 1$. En las dos primeras graficas que utiliza los dos primeros intervalos propuestos, se puede observar que utilizando únicamente la frecuencia mínima constante sin ningún tipo de gestión ni mecánismo de recuperación de instantes ociosos (llevando el procesador úniamente a "*modo idle*"), el consumo de potencia es mayor que el resto de métodos para todos los casos. En este sentido le sigue el método *lpps*, que en todos los intervalos propuestos, a utilizaciones bajas presenta un menor consumo superado únicamente por los método "*alpha constante*" y "*Sin DVS + algoritmo 2*", cuya diferencia de consumos se hace mayor a medida que aumenta el radio BCET/WCET. Sin embargo, a medida que las utilidades se hacen mayores el ahorro conseguido con el método *lpps* comienza a mermarse, pasando el método "*Sin DVS + algoritmo 2*" a ahorrar mayor energía que este, y cuyas diferencias se hacen menores a medida que aumenta el intervalo del radio BCET/WCET. Por otro lado, el método "Alpha Constante + Gestión *idle*" y el método propuesto "*rmit*", presentan los menores consumos en todos los intervalos BCET/WCET y para todas las utilidades, presentando el método *rmit* el mayor ahorro de energía, cuya diferencia en ahorro aumenta con el método anteriormente mencionado a medida que aumenta el intervalo del radio BCET/WCET.

Conclusiones

Por los resultados obtenidos, podemos concluir que la aplicación por sí solo del cálculo de la frecuencia mínima constante sin algoritmos on-line para la gestión de instantes *idles*, utilizando únicamente el modo *idle* del procesador, no garantiza una reducción considerable del consumo especialmente cuando se trabaja con utilidades altas.

Por otro lado, los algoritmos de gestión de instantes ocioso propuestos, así como el método propuesto *rmit*, presenta mejoras considerables en el ahorro de energía sin perjudicar el sobre-coste computacional, si se compara con otros métodos para la reducción de energía.

Capítulo 6

TRABAJO EXPERIMIENTAL

6.1. Introducción

Esta sección recoge principalmente el trabajo relacionado con pruebas de implementación de los dos tipos de *Middleware de Kernel de control*, el TCKM y FCKM (capítulo 2). Para estas pruebas se propone un escenario con un sistema de control distribuido utilizando un modelo de control organizado en capas, con un conjunto de actividades que deben ser ejecutadas para asegurar la fiabilidad, seguridad y disponibilidad del sistema. Actividades tales como cambio y conmutación de controladores a través de la delegación de código nuevo en los nodos empotrados del sistema. Parte de este trabajo ha sido publicado por el autor en diversos artículos (Coronel et al. (2008), Simarro et al. (2008), Coronel (2005), Nicolau et al. (2008), Martínez et al. (2007)) y el código fuente generado en este trabajo forma parte de proyectos nacionales e internacionales.

6.2. Embedded Nodes

Los nodos en los que se implementarán el TCKM y FCKM son respectivamente un microcontrolador dsPIC y un microprocesador XScale ((Martínez et al., 2007)).

El dsPIC de Microchip ((Angulo et al., 2006)) combina la gran velocidad computacional de los Procesadores de Señal Digial (DSP) con un potente microcontrolador de 16bit. Este dsPIC logra velocidades superiores a 30MIPS, es eficiente para programación C y tiene memorias de programa Flash, de datos EEPROM y de datos SRAM, perifericos potentes y una variedad de librerías de software.

Por otro lado, una arquitectura empotrada XScale ha sido escogida como plataforma de desarrollo para la implementación del FCKM debido a su bajo consumo de energía, su alto rendimiento y su bajo coste. Todas las generaciones de XScale son procesadores de 32-bits ARM v5TE fabricados con una tecnología de 0.18 μm. Este procesador soporta cambios en la frecuencia del núcleo del procesador entre 100MHz y 400 MHz, conveniente para realizar pruebas de optimización de consumo de energía dentro del middleware.

El XScale utiliza el sistema operativo de tiempo real denominado ”RTLinuxGPL” ((Coronel et al., 2006d)), el cual ofrece características estricta de tiempo real en un entorno multi-tarea y el cual puede utilizarse con sistema operativo empotrado. Pero para esto fue necesario hacer una modificación del código fuente original de RTLinuxGPL para adaptarlo a la arquitectura XScale. Se ha desarrollado una versión RTLinux con soporte para la arquitectura XScale, disponible en http://rtportal.upv.es/.

Igualmente se han desarrollado herramientas para el análisis de ejecución de las tareas que corren sobre el XScale. Para esto se ha modificado el código fuente de RTLinuxGPL para ARM consiguiendo con ello un análisis en tiempo de ejecución. En la figura 6.1 se puede observar un ejemplo de la ejecución de tres tareas de tiempo real y Linux (color rosa, en la parte superior).

Igualmente se ha diseñado una herramienta para la gestión y monitorización de todo el sistema middleware. Los nodos de la red son representados por cajas contenedoras de tareas de aplicación tales como controladores (ver figura 6.2).

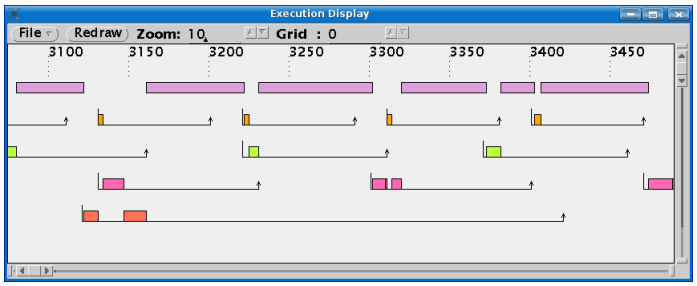

Figura 6.1: Cronograma de tareas en el XScale: Linux (color rosa, parte superior) y tres tareas de tiempo real.

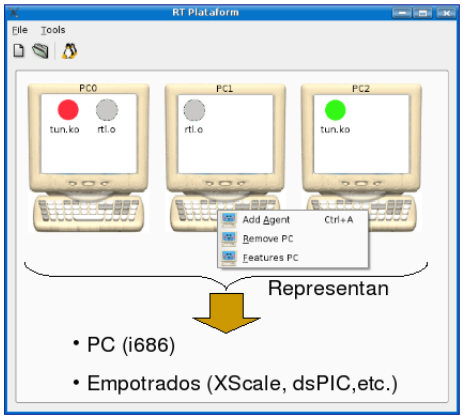

Figura 6.2: Aplicación para la monitorización de componentes dentro del sistema middleware.

6.3. Protocolo de comunicaciones

Aunque hay una gran variedad de buses de tiempo real, CAN (Controller Area Network) ((CiA, 1996)) es una de las soluciones preferidas para comunicar sistemas distribuidos de tiempo real ((Coronel et al., 2006a,b,c, 2005a,b)). Por consiguiente, en nuestro caso experimental, el CKM ha utilizado CAN como infraestructura para interconectar las dos unidades empotradas (XScale y dsPIC).

Para incorporar el bus de comunicaciones CAN en el nodo XSCale, se ha diseñado una tarjeta de expansión con un microcontrolador PIC el cual tiene empotrado un chip CAN (ver figura 6.3). Este PIC se comunica con el XScale a través de una memoria RAM de doble puerto. Los respectivos *drivers* software para el bus CAN han sido desarrollados para RTLinux. Por otro lado, el nodo DSPic ya tiene incorporado un chip CAN empotrado.

6.4. Experimentos

Con el fin de probar las características y capacidades del modelo de *Middleware de Kernel de control* propuesto, se propondrán dos casos de estudio: primero, evaluaremos la capacidad para conmutar controladores simples localizados sobre dos middleware diferentes interconectados mediante un canal de comunicaciones compartido, y segundo, aprovecharemos la posibilidad de cómputo distribuido para ejecutar un algoritmo de control predictivo para controlar un proceso real y proporcionar tolerancia a fallos ante errores esporádicos en la comunicación. La implementación de este trabajo fue presentado por el autor en Simarro et al. (2008).

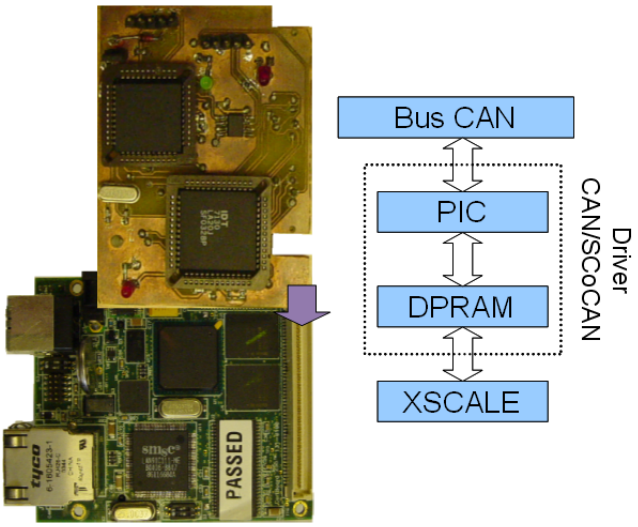

Figura 6.3: Placa de expansión CAN para el nodo XScale

6.4.1. Primer caso de estudio. Conmutación de controladores

La conmutación de controladores es una de las características clave en el modelo de CKM para ejecutar aplicaciones de control en modo seguro. Para este caso de estudio el nodo dsPIC que implementa la versión reducida del CKM (TCKM), esta conectado directamente a un proceso simulado y este le envía información acerca del estado del proceso al nodo XScale que incorpora la versión completa del CKM (FCKM), mediante el bus de comunicaciones CAN. En la figura 6.4 se puede observar el esquema general de la implementación.

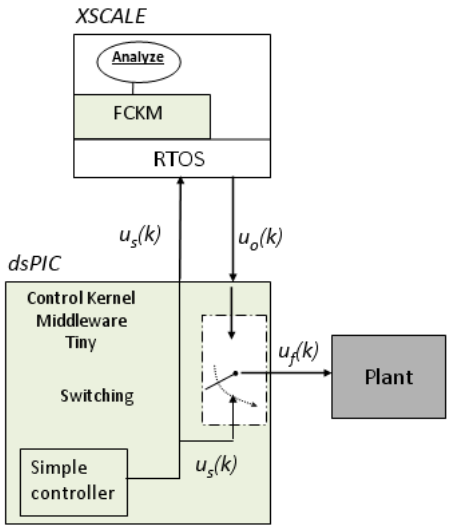

Figura 6.4: Esquema general del sistema distribuido a estudiar.

El dsPIC esta conectado a través de una tarjeta DAQ a un PC ejecutando un sistema simulado en Matlab con Simulink y la toolbox Real-Time WorkShop. El sistema es una simulación de Levitación Magnetica HUMUSOFT CE 152 a escala educacional. El modelo simulado transforma la señal de error en una señal real analógica a través de un puerto de salida analógico. La entrada analógica del DAQ es usada para obtener la señal de control.

Como se muestra en la figura 6.5, la señal de error ($e(k)$) es directamente muestreada por el dsPIC (TCKM) y su valor es transmitido al FCKM por medio del bus de comunicaciones. El TCKM aplica la señal de control final ($u_f(k)$) directamente sobre el proceso de control, cuya señal $u_f(k)$ puede ser $u_o(k)$ o $u_s(k)$.

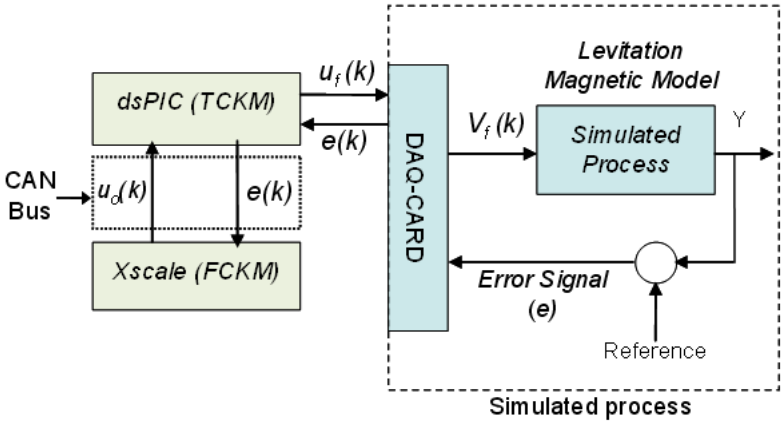

Figura 6.5: Esquema del sistema distribuido con el modelo simulado.

Diseño del Regulador

Por simplicidad y con el fin de ilustrar la conmutación de controladores, el TCKM obtiene la señal de error del sistema a través de su conversor A/D y computa un simple regulador proporcional derivativo PD. El algoritmo de control delegado al TCKM es:

$$u_s(k) = q_0.e(k) + q_1.e(k-1) \tag{6.1}$$

El FCKM recibe la señal de error y ejecuta un regulador discreto PID. El algoritmo de control sobre el XScale usa la siguiente ecuación:

$$u_0(k) = u_0(k-1) + q_0.e(k) + q_1.e(k-1) + q_2.e(k-2) \tag{6.2}$$

donde $e(k)$ es el error en el instante k, y $u_o(k)$ es la señal de control calculada para ser aplicada al sistema simulado.

Se han realizado varias pruebas sobre el entorno de simulación para obtener los valores de coeficientes óptimos para el proceso dado. Con esos resultados se ha determinado que la planta simulada podría ser controlada con un período de $5ms$. La señal de control es enviada del dsPIC a la planta por una salida PWM, a través de un filtro paso bajo RC para convertirla en una señal analógica continua.

Modos de ejecución

Dos tipos de situaciones son definidas para el modelo de control (ver figuras 6.4 y 6.5): primero, el FCKM produce una respuesta de control de alta calidad ($u_o(k)$) la cual es enviada al TCKM para su aplicación en la planta, y en este caso, el TCKM (dsPIC) actuará únicamente como interfaz hacia la planta; y segundo, cuando el control es llevado a cabo por el TCKM, y el FCKM únicamente monitoriza y analiza los datos de sensores y las acciones de control $u_s(k)$ recibidas en el bus CAN.

Para la primera situación, sí la señal $u_o(k)$ generada por el XScale (FCKM), no es recibida por el dsPIC o tiene algún tipo de retardo considerable, el TCKM aplicará la señal de control que él ha calculado ($u_s(k)$). Lo cual quiere decir, que una vez se supere un predeterminado umbral de espera, el TCKM pasará a ignorar la señal $u_o(k)$ y conmutará el control del sistema al regulador PD local ubicado en el dsPIC, ya que un retardo en la comunicación de la señal $u_o(k)$ puede significar que el

TCKM no se está ejecutando apropiadamente o que la red de comunicaciones está muy congestionado o fuera de servicio. Esta conmutación asegura que siempre exista una acción de control $u_f f(k)$ para ser aplicada a la planta evitando de esta forma la inestabilidad del sistema. En la figura 6.6 se muestra que la conmutación entre controladores no afecta la estabilidad del proceso.

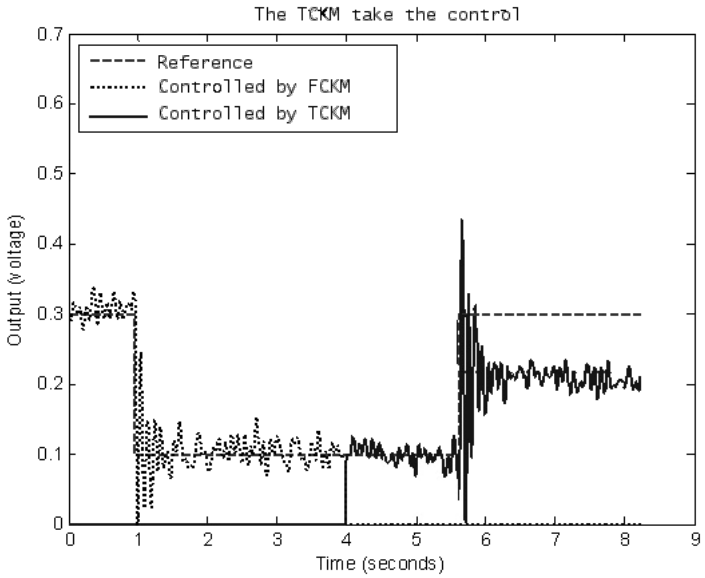

Figura 6.6: Evolución de las señales del proceso cuando algún mensaje CAN es perdido y el controlador del sistema es conmutado del FCKM a TCKM.

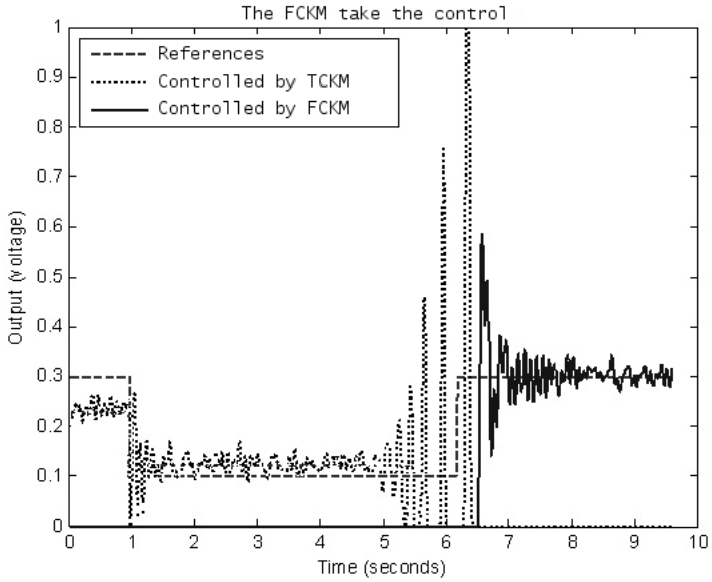

Figura 6.7: Evolución de las señales del proceso cuando se detecta un error de control y por consiguiente, el controlador es cambiado del TCKM al FCKM.

Para la segunda situación, la acción de control generada por el TCKM en el dsPIC es analizada por parte del FCKM para determinar si se está controlando apropiadamente el sistema. Cuando la

señal $u_s(k)$ es detectada como errónea, inmediatamente la señal $u_f(k)$ es conmutada a una señal segura $u_o(k)$ (ver figura 6.4). Esta conmutación es controlada mediante el análisis de la señal de error presente y acumulada por parte del dsPIC, sí el parámetro de error excede un valor predeterminado el regulador del proceso es cambiado al del supervisor, es decir, al del FCKM. En la figura 6.7 se observa que la regulación es realizada de forma correcta hasta que en el segundo 5 se simula una perdida excesiva de datos y entonces el proceso comienza a ser inestable, y una vez se detecta está situación el nodo con el FCKM toma el control del sistema.

La distribución y entrega de los datos de sensores y de las acciones de control dentro del sistema distribuido es llevada a cabo por el CKM (TCKM y FCKM).

6.4.2. Segundo caso de estudio. Supervisión de control con compensación local

En este caso, la idea es aprovechar el cómputo distribuido para implementar un algoritmo de control predictivo Camacho & Bordons (2007). Este tipo de algoritmos de control provee acciones de control futuras que pueden ser utilizadas por el procesador local (dsPIC) para ser aplicados a los controladores en caso de retardos de comunicaciones o perdida de datos inesperados. El algoritmo de control predictivo es un *Control Predictivo Generalizado* (GPC). A continuación se explicará la estrategia de desarrollo.

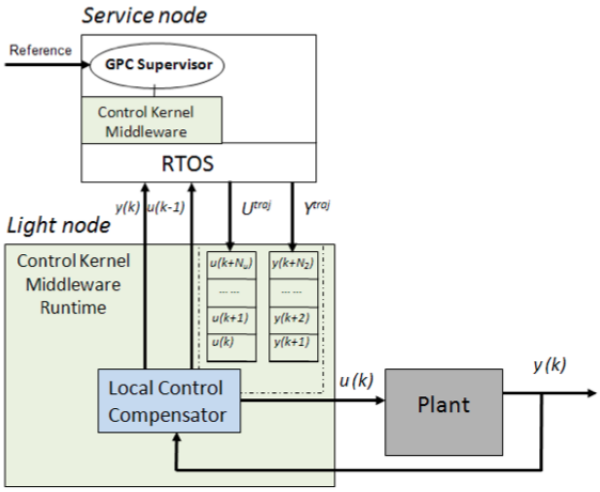

Figura 6.8: Estructura distribuida para el segundo caso

La estructura de desarrollo básica involucra un sistema de control distribuido formado por un nodo con FCKM en donde se ubicará el control GPC supervisor, y un nodo con TCKM. El nodo con el procesador XScale implementará el CKM completo y el nodo dsPIC implementará la versión reducida del CKM (TCKM).

En el FCKM se llevará a cabo el cálculo de las acciones de control del GPC que serán aplicadas en el TCKM. El XScale o FCKM enviará periódicamente las trayectorias de las acciones de control y las salidas predecidas para un horizonte de ciclos de control predeterminado. Si el nodo TCKM aplica la secuencia de acciones de control que han sido calculadas por el GPC desde la iteración "*k*", para ese caso la estrategia a ser aplicada sera en bucle abierto durante el horizonte de predicción $[N_1, N_2]$. Sin embargo, la primera acción de control $u(k)$ que ha sido enviada por el FCKM siempre será una acción calculada en una estrategia de bucle cerrado, y esta será la primera acción a ser aplicada por el TCKM. Este procedimiento es repetido en cada instante de muestreo. En la figura 6.8 se puede observar el esquema propuesto para este segundo caso.

En un sistema empotrado con recursos de computo limitados, la información de acciones de control futuras es muy valiosa, puesto que estas pueden ser usadas y aplicadas dentro el horizonte de predic-

ción, en situaciones adversas con pocas medidas disponibles o con tiempos de cálculos excesivos en comparación con el período de control o por perdida de datos en el canal de comunicación.

La principal ventaja de utilizar este método en sistema distribuidos de control es su tolerancia a fallos ante errores esporádicos en la comunicación. Tal como se muestra en la figura 6.9, cuando la acción de control no es recibida en el TCKM (debido por ejemplo a perdida de datos o errores en la comunicación), este tiene que aplicar las acciones predecidas en los próximos períodos de muestro. Si las acciones de control son mayor que el horizonte de control, el TCKM deberá aplicar la última acción de control de acuerdo a la trayectoria de datos predecida. Este escenario es mostrado en las figuras 6.9 y 6.10, donde los datos son perdidos durante 5 períodos de muestreo (50*ms*).

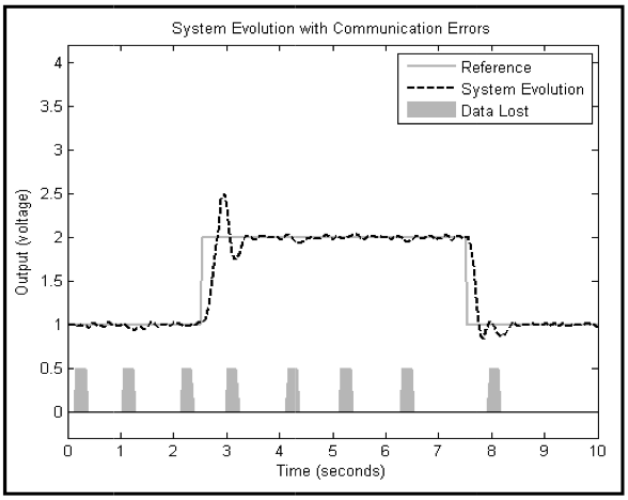

Figura 6.9: Evolución del control del proceso cuando hay perdida de datos

Las dos figuras son el resultado de las pruebas para el mismo escenario, figure 6.9 muestra la evolución del proceso (la señal del sistema trata a seguir el valor de referencia), mientras que la figura 6.10 muestra la señal de control aplicada, el valor de control calculado por el GPC y las diferencias entre estas.

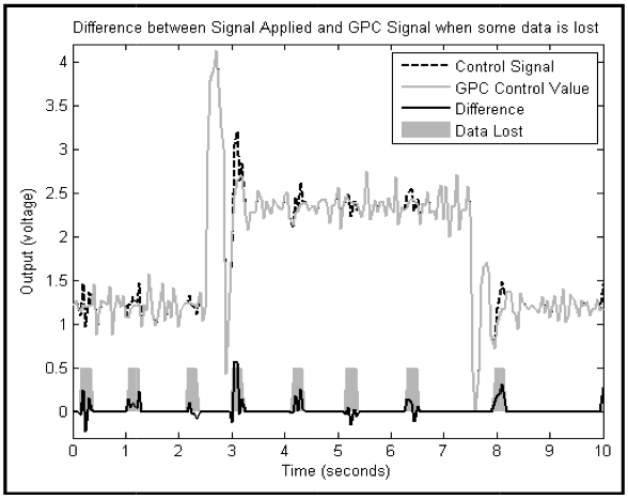

Figura 6.10: Señal de control aplicada, valores de control GPC calculados y su discrepancia cuando hay perdida de datos

La mayor diferencia entre la señal de control aplicada y el valor calculado por el GPC ocurre cuando

la comunicación es perdida justo dentro de la respuesta transitoria (segundo 3 en la figura 6.10). Aunque este es el peor caso por perdida de datos, el control en el TCKM se recupera con la evolución del proceso como es apreciado en la figura 6.9. El control en el TCKM corrige las acciones de control considerando la discrepancia entre la trayectoria de salida predecida y la salida real, minimizando el error en la salida.

6.5. Conclusiones

Desde el punto de vista de implementación el modelo de textitMiddleware de Kernel de control propuesto permite llevar a cabo un conjunto de actividades básicas que aseguran una operación fiable del sistema bajo control. La combinación de sistemas empotrados con diferentes características de computo permite abaratar costes en la implantación de sistemas de control sin afectar la operación fiable, segura y predecible del sistema.

Capítulo 7

CONCLUSIONES

El desarrollo de sistemas modernos requiere una integración fuerte a todos los niveles entre el entorno computacional y el entorno real. En este punto los sistemas ciber-físicos (CPS—Cyber-Physical Systems) (Khaitan & McCalley, 2015) se presentan como una solución a este tipo de desafíos, ofreciendo una nueva clase de sistema de ingeniería que ofrecen una interacción cercana entre componentes software, redes y procesos físicos. En sectores industriales tales como el ferroviario, aeroespacial y automóvil los sistemas CPS podrían tener un rol importante, donde la operatividad de estos sectores está basada en Sistemas Empotrados Críticos de Tiempo Real (también conocido por sus siglas en inglés CRTES — Critical Real-Time Embedded Systems), interconectados a través de redes de comunicaciones, y interactuando con el mundo físico a través de sensores, actuadores y consumiendo energía real. Dichos sistemas se enfrentan a nuevas demandas y exigencias relacionadas con el incremento de la fiabilidad, mayor inteligencia, conectividad, reducción del volumen, mejoras del rendimiento y eficiencia en el consumo energético (Perez et al., 2014a). Y es en estos aspectos donde las contribuciones de esta tesis pueden tener su aplicabilidad.

El consumo energético, es especialmente importante en sectores industriales basados en sistemas autónomos con energía limitada, donde la gestión del consumo energético es esencial. En general la gestión de este recurso es importante en los sectores donde los CRTES estén alimentados por baterías tales como aplicaciones de aviónica, control remoto de grúas, sensores inalámbricos de campo, robótica, automóviles (especialmente coches eléctricos y autónomos) y sectores emergentes tales como drones o vehículos aéreos no tripulados (VANT). Sin embargo, aunque los sistemas críticos no estén alimentados por baterías, el consumo energético es un recurso que debería ser considerado por varias razones tales como:

- Fiabilidad: el bajo consumo y la adecuada gestión de la temperatura de los sistemas es un factor importante para incrementar la disponibilidad y fiabilidad en muchos sistemas industriales. Si el consumo en el sistema se reduce, el calor también se reduce y el impacto en la fiabilidad puede llegar a ser el doble. La influencia negativa en el envejecimiento de los componentes hardware es reducida y además, esto puede evitar el uso de sistemas de refrigeración y partes móviles (ejemplo, ventiladores) en el diseño hardware lo cual puede inducir significantes probabilidades de fallos al sistema o reducir los intervalos de mantenimiento.

- Disponibilidad: un consumo energético bajo permite extender la operación de los sistemas en especiales situaciones tales como apagones y cortes energéticos.

- Ecológico: el bajo consumo también contribuye a la reducción de emisiones en sistemas con decenas y cientos de ECUs. Incluso pequeñas reducciones en el consumo por sistema embebido puede tener un gran impacto desde una perspectiva a escala global debido a el gran número de este tipo de sistemas en todo el planeta.

Pero el camino para mejorar el consumo eficiente de potencia pasa por la implementación de técnicas de gestión de energía a diferentes niveles del sistema: a nivel de chip hardware (por ejemplo, a nivel de núcleos de procesador, a NoC (network-on-chip)), a nivel de sistema software (ejemplo, hipervisor, sistema operativo) y a nivel de sistema distribuido (ejemplo nodos, redes, etc).

En este sentido, en este trabajo de tesis se han propuesto métodos y procedimientos enmarcados en la *aceptación de tareas, planificabilidad* y técnicas de *gestión de energía*, que combinados con otros criterios tales como retardos de comunicación, gestión térmica y estabilidad en aplicaciones de control pueden ser utilizados en la determinación del movimiento de tareas y el balance de cargas en un sistema interconectado.

El desarrollo de esta tesis se ha enmarcado en un entorno dinámico donde sistemas CRTES, basados en soportes *middleware* y conectados a una red de comunicaciones, permiten llevar a cabo migraciones de tareas y modificaciones de la frecuencia del procesador en tiempo de ejecución. Para ello se han propuesto un conjunto de requisitos mínimos, funcionalidades y componentes necesarios a ser considerados en el diseño de un sistema *middleware* basado en el concepto de núcleo de control. En este aspecto, las contribuciones de esta tesis se presentan especialmente como funcionalidades en la gestión de recursos dentro del núcleo de control.

En este trabajo se ha propuesto un algoritmo nuevo (método $\mathcal{A}$) para el *análisis de factibilidad* en políticas de planificación basadas en prioridades fijas con tareas con plazos de ejecución menor y/o igual que el periodo, y para el *cálculo de frecuencias estáticas* de procesador basado en técnicas de escalamiento de frecuencia y voltaje dinámico (también conocido como DVFS — Dynamic Voltage and Frequency Scaling). La frecuencia obtenida por este algoritmo es la frecuencia mínima que garantiza que si se usa de forma invariable en el procesador, se ahorrará la mayor energía posible y además se cumplirán todos los plazos de ejecución de las tareas del sistema. Uno de los propósitos de este algoritmo es su uso durante la ejecución del sistema, que permita gestionar adaptaciones de carga computacional y energética del procesador.

Además se han presentado los resultados de extensivas simulaciones donde se compara el comportamiento del método propuesto $\mathcal{A}$ con varios métodos de análisis de factibilidad para el cálculo del factor de escalamiento de frecuencia α cuando las tareas tienen $D = T$ y $D \leq T$. El comportamiento de los algoritmos ha sido analizado desde el punto de vista de consumo de energía, grado de rechazo y coste de computo real del algoritmo.

Los resultados de las simulaciones muestran que el algoritmo propuesto $\mathcal{A}$ aventaja a otros algoritmos en el cálculo de la constante de escalado de frecuencia desde el punto de vista de coste computacional, predecibilidad y grado de aceptación de tareas. El método propuesto permite no únicamente la verificación con una gran precisión de la factibilidad del sistema, sino que además permite el cálculo del factor α, por lo que el algoritmo puede utilizarse como método de prueba de aceptación de tareas en tiempo de ejecución en entornos con cargas de procesamiento dinámicas. El algoritmo también puede ser usado en sistema operativos de tiempo real porque la sobrecarga derivada de la ejecución del algoritmo es reducida y predecible, y se minimiza el consumo de energía a la vez que se garantiza la no perdida de plazos de ejecución de tareas.

El anterior algoritmo para el cálculo de frecuencias estáticas de procesador, ha sido complementado en esta tesis con la propuesta de procedimientos de gestión de energía y un método nuevo (algoritmo *rmit*) para la optimización dinámica del consumo energético, que aumentan el ahorro de energía basado en las condiciones de carga de computo reales en cada instante. Esta propuesta se utiliza en tiempo de ejecución de las tareas y está basada en la asignación dinámica de de frecuencias de operación del procesador. Estas nuevas frecuencias utilizan como referencia el cálculo previo de la frecuencia mínima constante. Los cambios de frecuencia dinámicos se suceden como respuesta a instantes ociosos de procesador debidos principalmente a terminaciones anticipadas en la ejecución de trabajos.

Esta optimización dinámica del consumo energético considera principalmente aspectos de implementación, es decir, la base de los procedimientos propuestos y el algoritmo *rmit* está condicionada por su viabilidad a la hora de ser desarrollados sobre un sistema de computo real. Lo cual implica que la información y los parámetros a ser utilizados en el análisis deberán estar disponibles a priori, tanto a nivel de hardware como a nivel de sistema operativo, teniendo en cuenta las especificaciones del planificador y la definición de los hilos de ejecución en un operativo común. Adicionalmente, hay que tener en cuenta que en optimizaciones dinámicas la reactividad del sistema debe ser muy alta, por lo que la simplicidad de los algoritmos es un factor clave para conseguir esto. Por tales motivos, en la evaluación del sistema propuesto se han descartado de la comparativa varios algoritmos presentes en la literatura. Algunos de los algoritmos descartados ofrecían una mejor optimización de consumo, pero estas propuestas se basaban en parámetros idealistas y difíciles de obtener o de planteamientos

complejos e inviables de ser llevados a la práctica con tiempos de respuesta aceptables. Algunos de estos tenían como base condiciones y estados futuros de tareas, parámetros no disponibles y atributos poco comunes en un planificador real.

De los resultados de aplicar el algoritmo *rmit* para la gestión dinámica de energía, podemos concluir que la utilización por sí sola del cálculo de la frecuencia mínima constante no garantiza una reducción óptima del consumo, especialmente en escenarios donde el tiempo de ejecución del peor caso es muy pesimista y poco repetido durante las activaciones de las tareas. Por tanto, para maximizar el ahorro energético también es necesario

Las principales aportaciones de esta tesis han sido presentados en diversos artículos de revistas y congresos a lo largo del desarrollo de esta tesis: (Cilku et al., 2015; Coronel & Simó, 2012; Coronel et al., 2009; Simarro et al., 2008; Coronel et al., 2008; Nicolau et al., 2008; Martínez et al., 2007; Coronel et al., 2006d)

7.1. Trabajo futuro

Una vez terminado el desarrollado de esta tesis y analizando las perspectivas y los nuevos retos que se prevén en sistemas CRTES con la incursión de sistemas con criticidad mixta, a continuación se proponen posible ampliaciones de los resultados y futuras líneas de investigación.

Aunque algunos de los resultados de evaluación presentados en esta tesis se basan en simulaciones, los algoritmos, metodología y pruebas están enfocados en aspectos que permiten una implementación directa en sistemas de cómputo reales. Estas consideraciones están fundamentadas por la amplia experiencia del autor en el diseño y construcción de sistemas operativos de tiempo real (Coronel, 2005; Carrascosa et al., 2014; Crespo et al., 2014; Coronel et al., 2013; Masmano et al., 2013; Galizzi et al., 2012; Coronel et al., 2005a). Por tal motivo, se propone la ampliación de los experimentos presentados en esta tesis en sistemas empotrados reales que incluyan la aplicación conjunta de todas las técnicas y algoritmos presentados.

En esta tesis y la mayoría de la literatura, el consumo energético y la factibilidad en la planificación del sistema ha estado supeditado por el tiempo, por el cumplimiento del plazo de todas las tareas, donde el factor más importante es la variable temporal. Sin embargo, en algunos sistemas (ejemplo, sistemas de criticidad mixta) con trabajos de diferente índole y importancia, donde la ejecución y cumplimiento de algunas tareas podría ser de mayor importancia que otras, incluso a expensas de que hayan tareas que no se lleguen a ejecutar. En este escenario, y suponiendo que se está en presencia de recursos limitados, las motivaciones para la asignación de energía podría cambiar. En determinadas circunstancias, el control del consumo del sistema basado únicamente en el tiempo puede llevar a que los objetivos generales de este tipo de sistema no se lleguen a conseguir, teniendo en cuenta que el consumo de potencia de una aplicación reduce la energía disponible para otra aplicación o la fiabilidad y el tiempo de vida de todo el sistema. En tales casos, es necesario cambiar la perspectiva del análisis de factibilidad donde las tareas deberían planificarse también en función de una componente energética y no únicamente en función de la temporalidad. Esta nueva perspectiva lleva a considerar una nueva dimensión en el análisis de factibilidad y cálculo de la frecuencia del procesador, donde los recursos de tiempo y energía cobren la misma relevancia. En este sentido, el trabajo presentado en esta tesis podría extenderse para incluir estos nuevos requerimientos. Este nuevo enfoque es especialmente importante en sectores industriales basados en sistemas autónomos o con energía limitada tales como el espacial, donde la gestión del consumo energético es fundamental.

En sistemas con criticidad mixta, los sistemas particionados basados en hipervisores (Crespo et al., 2014; Coronel et al., 2013; Windsor & Hjortnaes, 2009) cobran una gran relevancia especialmente a la hora de hacer frente a esas nuevas demandas de escalabilidad, seguridad (Perez et al., 2014b) y rendimiento. Sin embargo, algunos aspectos tales como la gestión y optimización del consumo energético aún no han sido completamente abordados en este tipo de sistemas particionados. En este sentido, se propone aplicar algunas de las contribuciones de esta tesis en la caracterización de particiones, donde el factor energético se convierte en un atributo adicional en la especificación del sistema de computo. Para ello, cada partición se convierte en un entorno de computo limitado por la disponibilidad de energía asignada. De esta forma, la asignación de tareas en sistemas con criticidad mixta teniendo en cuenta que la energía es un recurso compartido podría simplificarse.

Apéndice A

Interfaces y interacciones de componentes Middleware de control

Este anexo recoge las interfaces, interacciones y comportamientos referenciados en el capítulo 2 y relacionado con la definición del middleware de control.

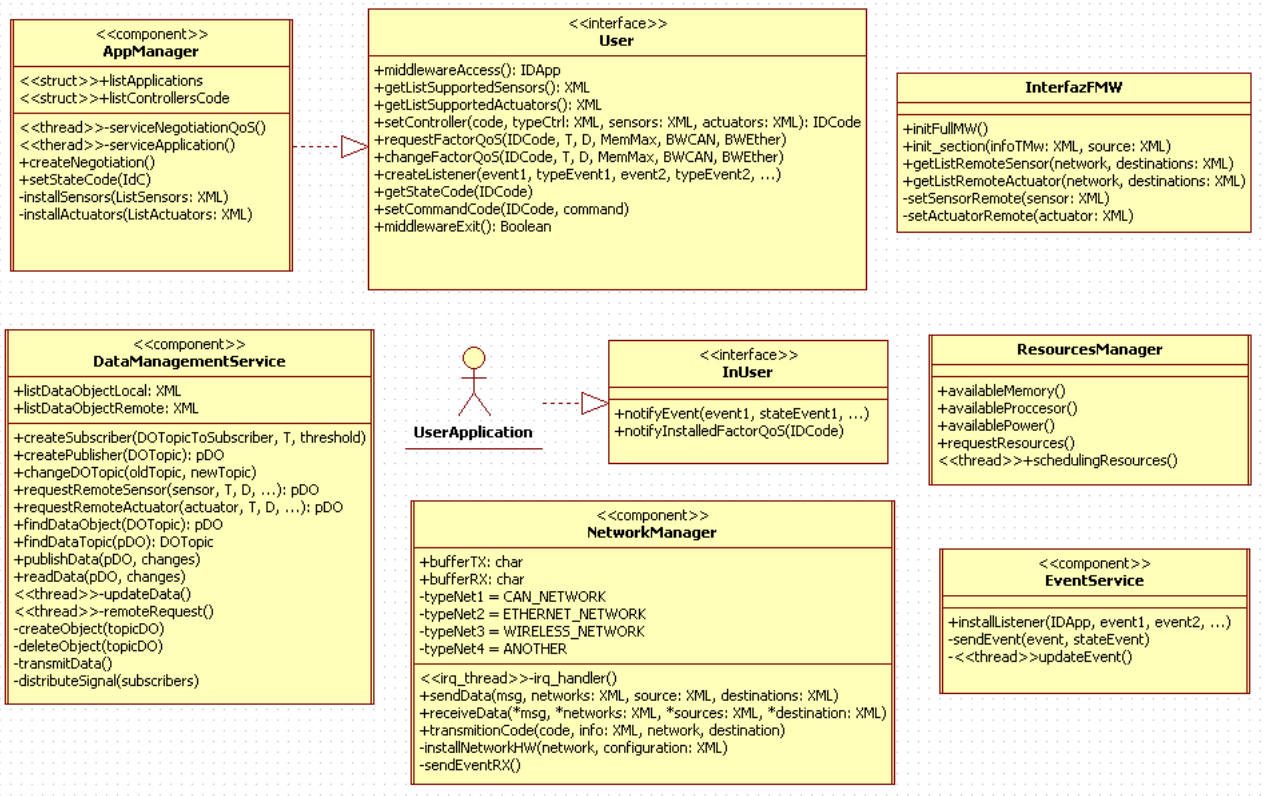

Figura A.1: Parte 1: Interfaces de los componentes del full-middleware

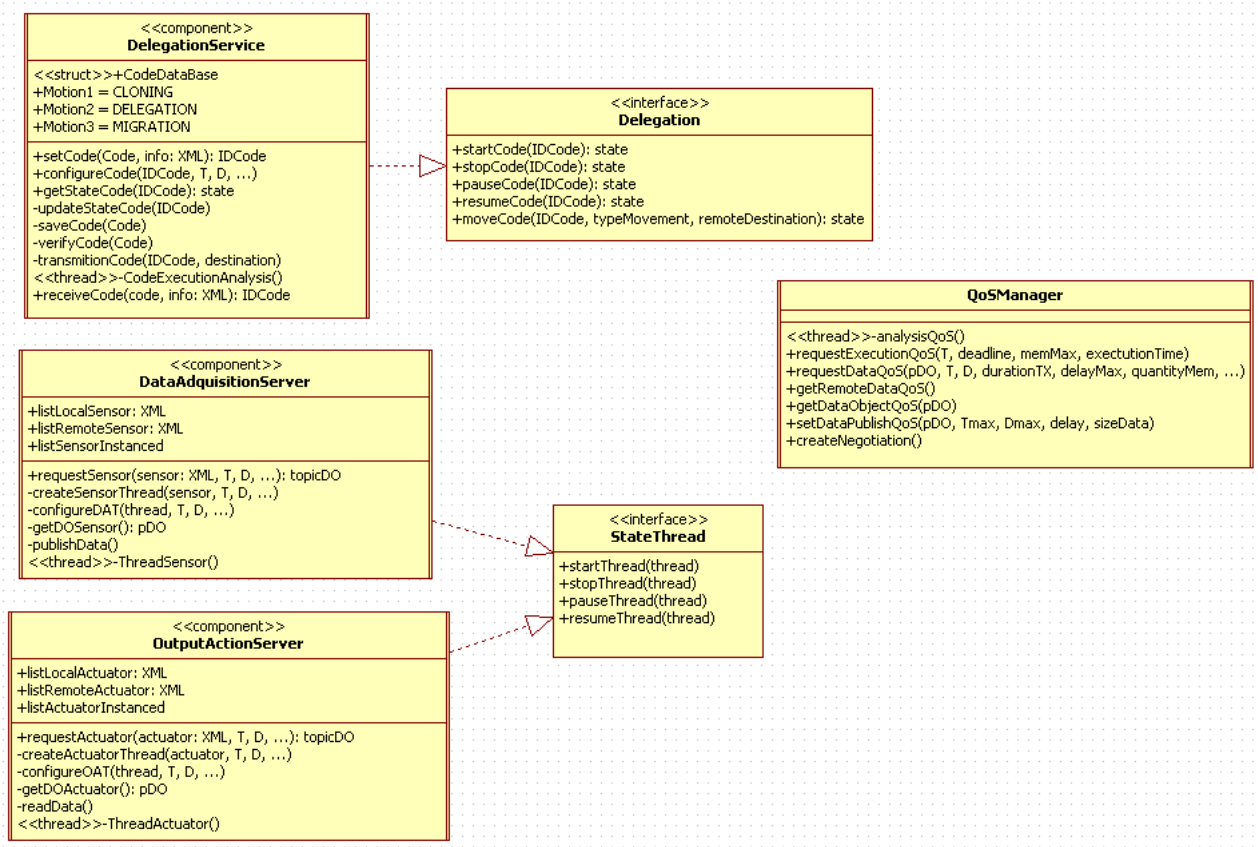

Figura A.2: Parte 2: Interfaces de los componetes del full-middleware

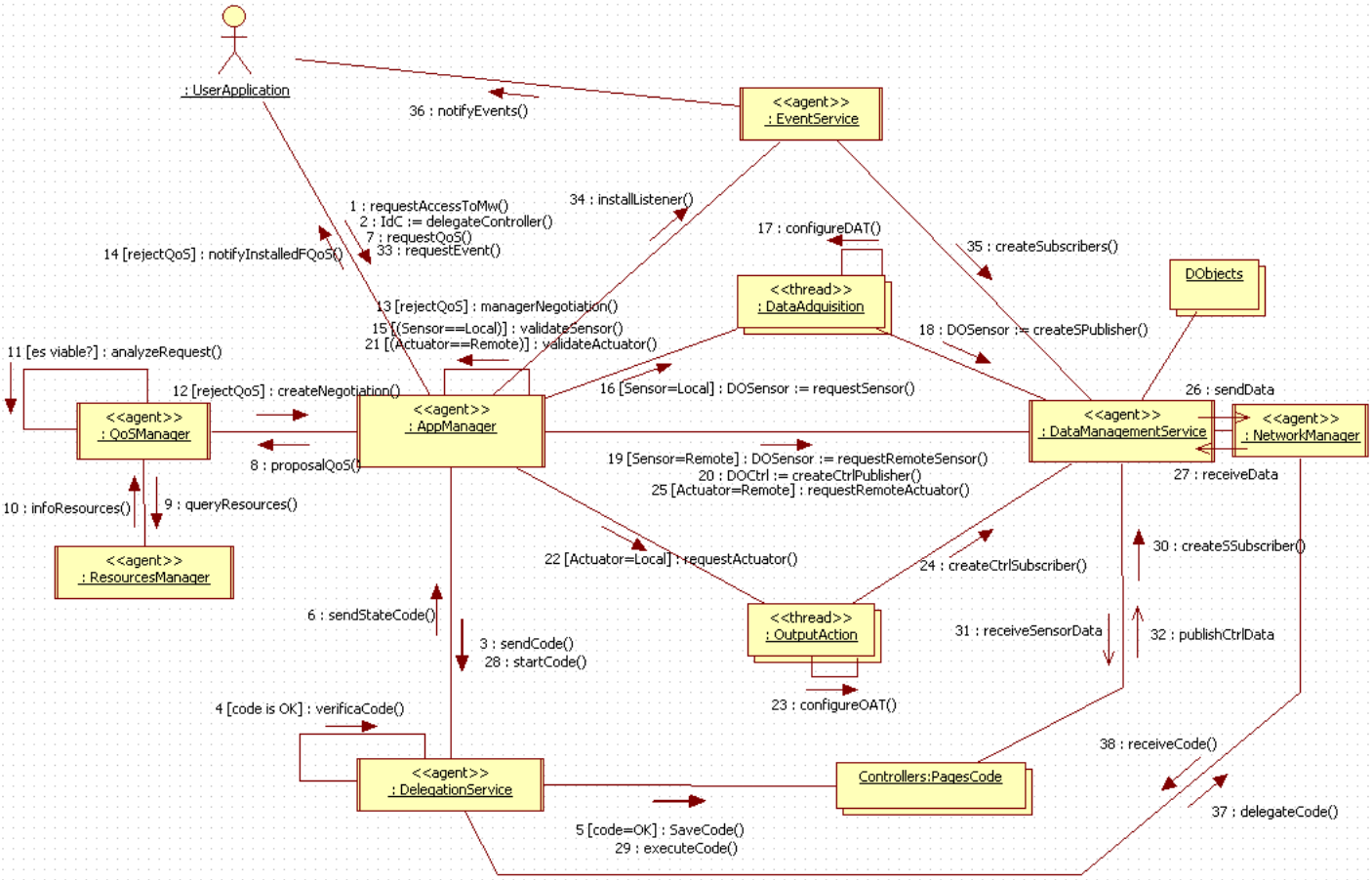

Figura A.3: Relación de componentes del Full-Middleware

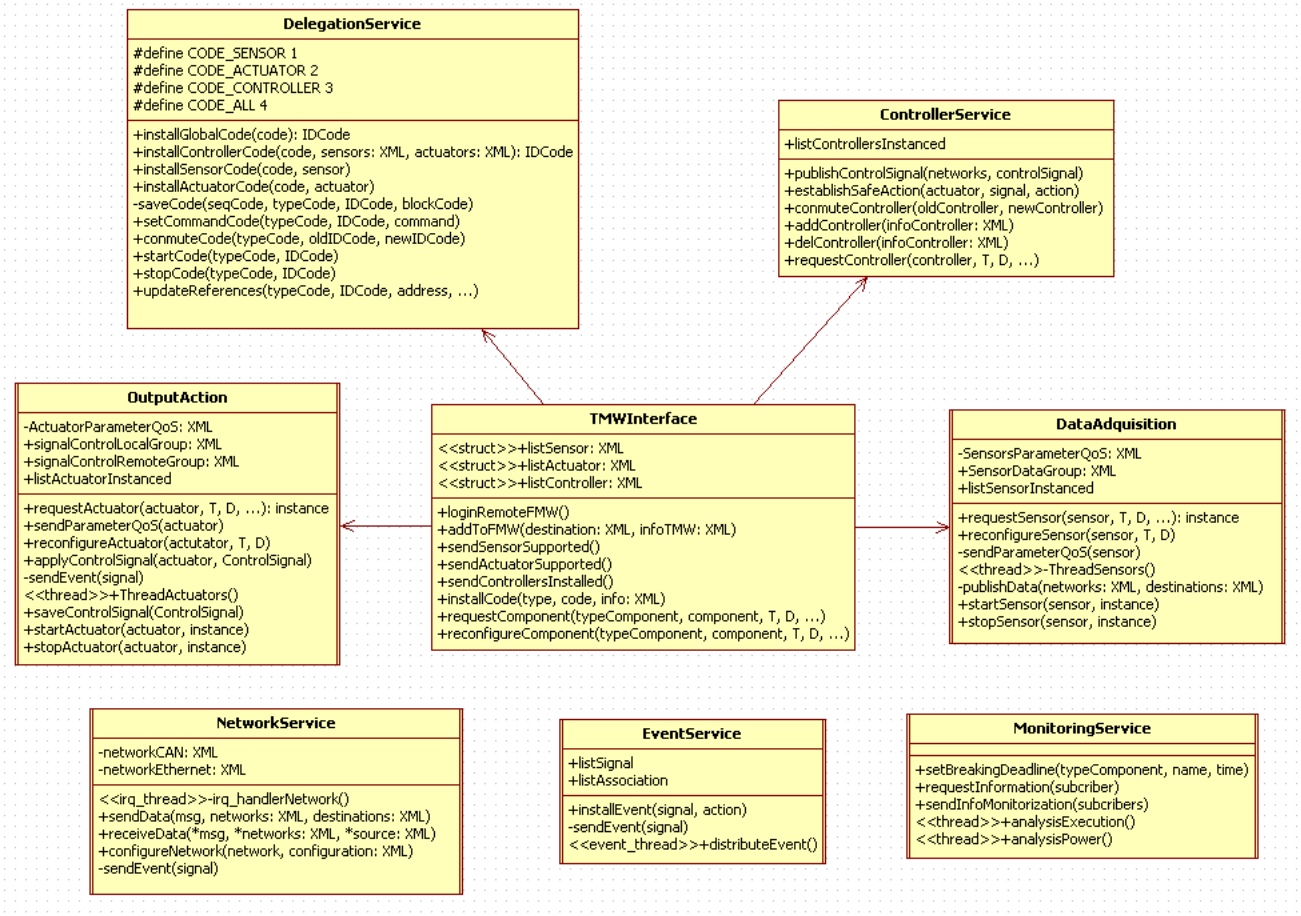

Figura A.4: Interfaces de los componetes del Tiny-Middleware

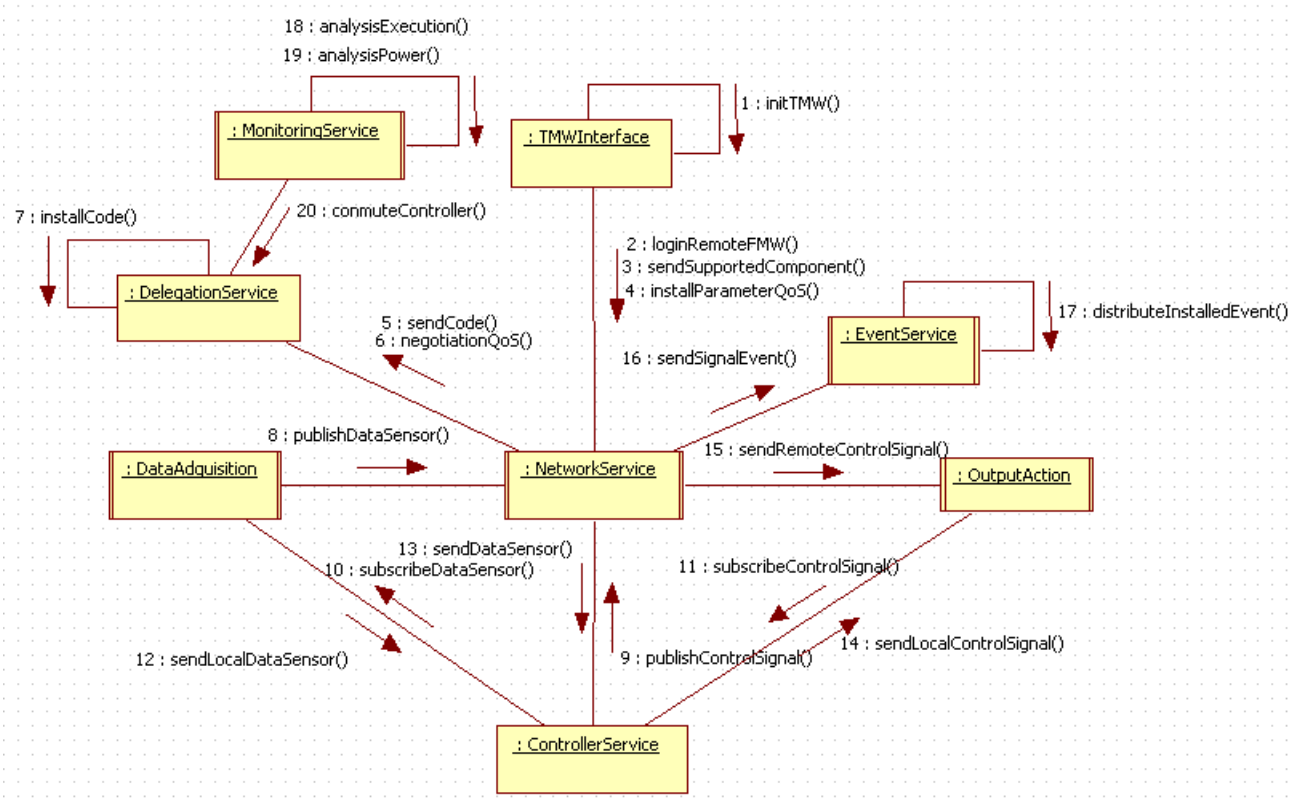

Figura A.5: Relación de los componentes de la versión Tiny-Middleware

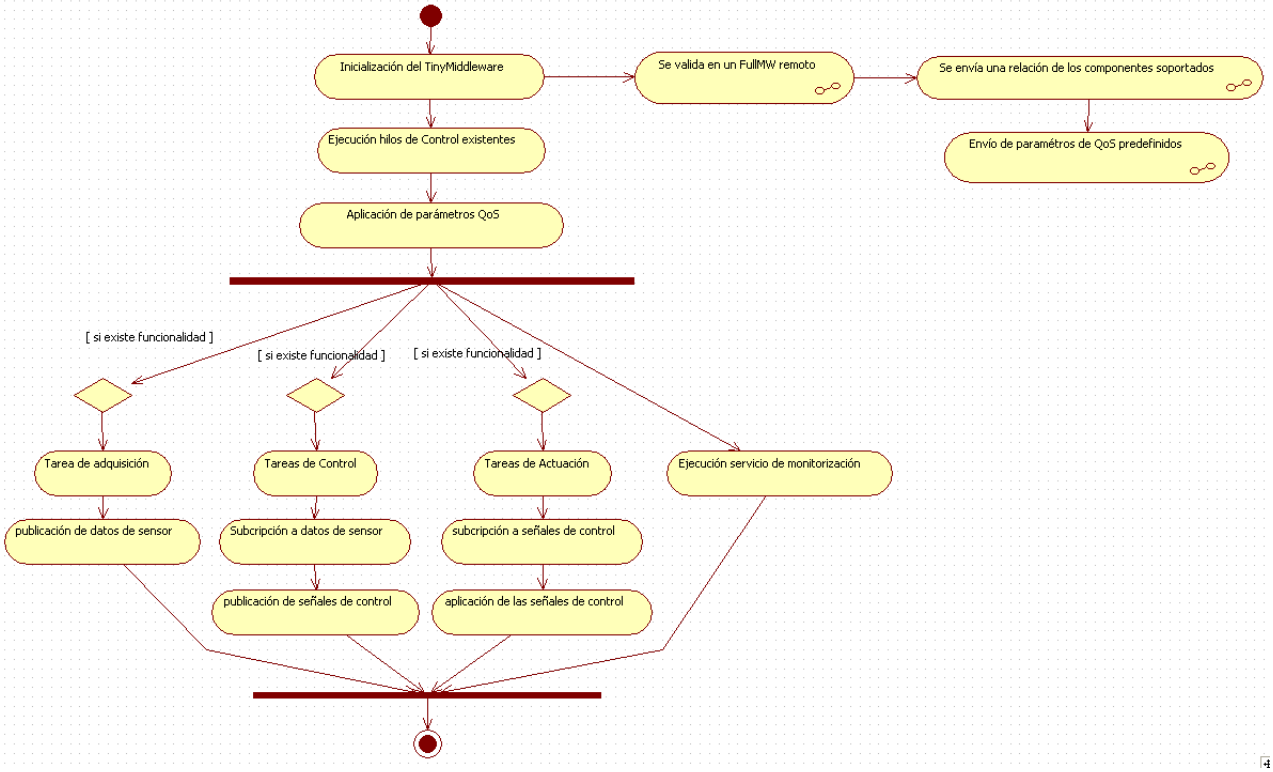

Figura A.6: Inicialización del sistema Tiny-Middleware

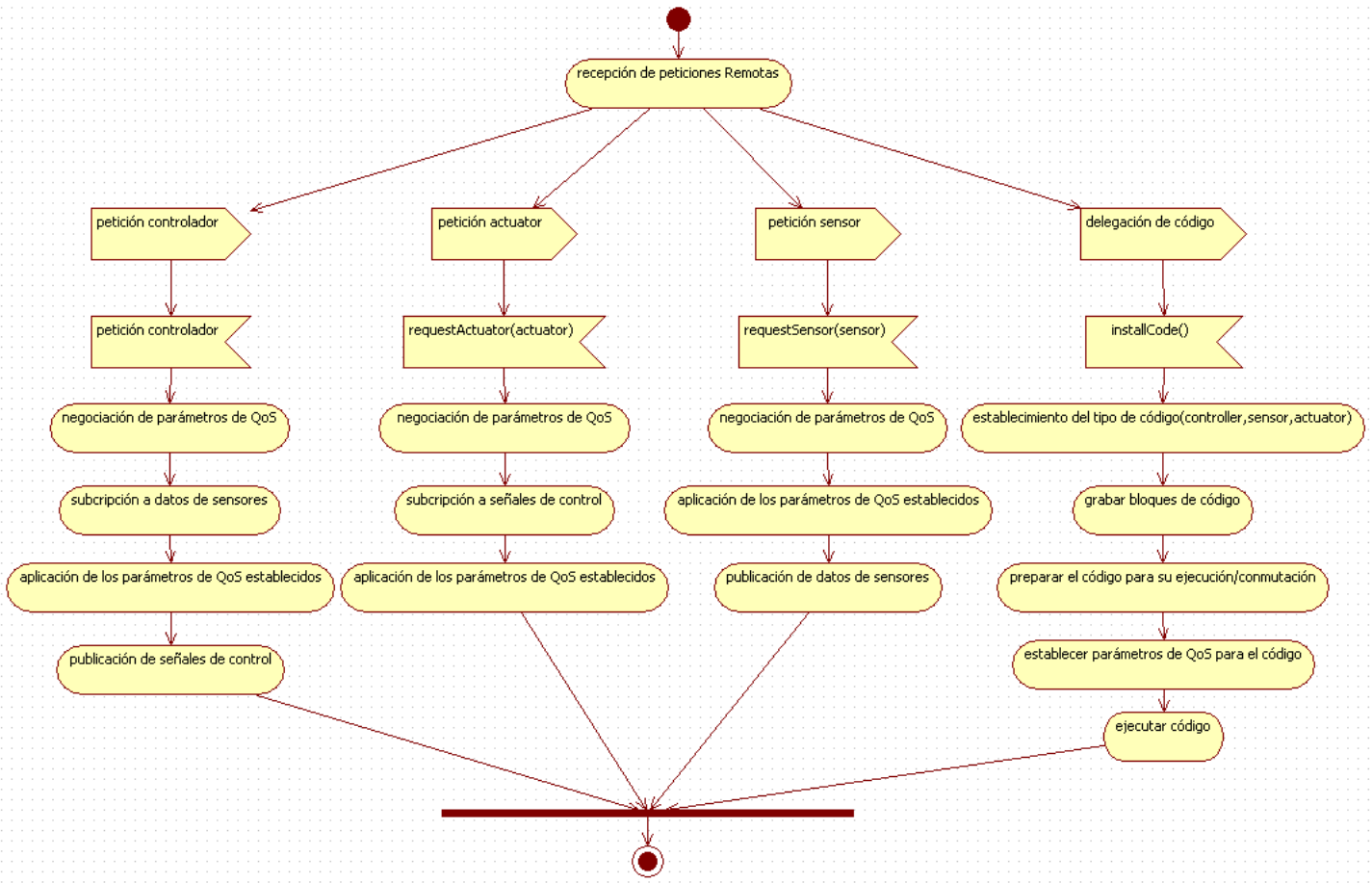

Figura A.7: Enlace con componentes del sistema Tiny-Middleware

Bibliografía

ADS (). Ads bitsyxb platform. In *www.applieddata.net*.

Al-Areqi, S., Gorges, D., & Liu, S. (2015). Event-based control and scheduling codesign: Stochastic and robust approaches. *Automatic Control, IEEE Transactions on*, *60*, 1291–1303.

Albertos, P., Crespo, A., & Simó, J. (2006). Control kernel: A key concept in embedded control systems. *4th IFAC Symposium on Mechatronic Systems*, .

AlEnawy, T., & Aydin, H. (2005). Energy-aware task allocation for rate monotonic scheduling. In *Real Time and Embedded Technology and Applications Symposium, 2005. RTAS 2005. 11th IEEE* (pp. 213–223).

Anderson, J. H., Bud, V., & Devi, U. C. (2005). An edf-based scheduling algorithm for multiprocessor soft real-time systems. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems* (pp. 199–208). Washington, DC, USA: IEEE Computer Society.

Angulo, J., Etxebarría, A., Angulo, I., & Trueba, I. (2006). *dsPIC Diseño práctico de aplicaciones*. McGraw Hill (primera ed.).

Aydin, H., Devadas, V., & Zhu, D. (2006). System-level energy management for periodic real-time tasks. In *RTSS '06: Proceedings of the 27th IEEE International Real-Time Systems Symposium* (pp. 313–322). Washington, DC, USA: IEEE Computer Society.

Aydin, H., Melhem, R., Mossé, D., & Mejía-Alvarez, P. (2004). Power-aware scheduling for periodic real-time tasks. *IEEE Trans. Comput.*, *53*, 584–600.

Balbastre, P., Balbastre, P., Ripoll, I., & Crespo, A. (2006). Optimal deadline assignment for periodic real-time tasks in dynamic priority systems. In I. Ripoll (Ed.), *Proc. 18th Euromicro Conference on Real-Time Systems* (pp. 10 pp.–).

Baliga, G., Graham, S., Sha, L., & Kumar, P. R. (2004). Service continuity in networked control using etherware. In *Proceedings of Middleware 2004*. Toronto, Canada.

Bini, E., & Buttazzo, G. (2004). Schedulability analysis of periodic fixed priority systems. *IEEE Transactions on Computers*, *53*, 1462–1473.

Bini, E., Buttazzo, G., & Lipari, G. (2005). Speed modulation in energy-aware real-time systems. In *ECRTS '05: Proceedings of the 17th Euromicro Conference on Real-Time Systems* (pp. 3–10). Washington, DC, USA: IEEE Computer Society.

Bini, E., Buttazzo, G. C., & Buttazzo, G. M. (2003). Rate monotonic analysis: the hyperbolic bound. *IEEE Transactions on Computers*, *52*, 933–942.

Bini, E., Natale, M. D., & Buttazzo, G. C. (2006). Sensitivity analysis for fixed-priority real-time systems. In *Proceedings of the 18th Euromicro Conference on Real-Time Systems*. Dresden, Germany.

Biondi, A., & Buttazzo, G. (2015). Engine control: Task modeling and analysis. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition* DATE '15 (pp. 525–530). San Jose, CA, USA: EDA Consortium.

Briao, E. W., Barcelos, D., Wronski, F., & Wagner, F. R. (2007). Impact of task migration in noc-based mpsocs for soft real-time applications. In *Very Large Scale Integration, 2007. VLSI - SoC 2007. IFIP International Conference on* (pp. 296–299).

Buttle, D. (2012). Real-time in the prime-time - ecrts (keynote talk). *ECRTS*, .

Camacho, E., & Bordons, C. (2007). *Model Predictive Control*. Springer-Verlag London.

Carrascosa, E., Coronel, J., Masmano, M., Balbastre, P., & Crespo, A. (2014). Xtratum hypervisor redesign for leon4 multicore processor. *SIGBED Rev.*, *11*, 27–31.

Cervin, A. (2003). *Integrated Control and Real-Time Scheduling*. Ph.D. thesis Department of Automatic Control, Lund University ISRN LUTFD2/TFRT–1065–SE.

Cervin, A., Ohlin, M., & Henriksson, D. (2007). Simulation of networked control systems using TrueTime. In *Proc. 3rd International Workshop on Networked Control Systems: Tolerant to Faults*. Nancy, France. Invited talk.

Chen (2007). Energy-efficient real-time task scheduling with task rejection, .

Chen, J.-J. (2005). Multiprocessor energy-efficient scheduling for real-time tasks with different power characteristics. In *ICPP '05: Proceedings of the 2005 International Conference on Parallel Processing* (pp. 13–20). Washington, DC, USA: IEEE Computer Society.

Cho, Y., & Chang, N. (2006). Energy-aware clock-frequency assignment in microprocessors and memory devices for dynamic voltage scaling. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *26*, 1030–1040.

CiA (1996). *CAN Appplication Layer for Industrial Applications*. Technical Report DS-201...207, v 1.1 CAN in Automation.

Cilku, B., Crespo, A., Coronel, J., & Peiro, S. (2015). A tdma-based arbitration scheme for mixed-criticality multicore platforms. *First International Conference on Event-Based Control, Communication, and Signal Processing*, .

Coronel, J. (2005). Porting of rtlinux 3.2rc1 to xscale processors. In *http://rtportal.upv.es/apps/xscale/*.

Coronel, J., Blanes, F., Benet, G., Pérez, P., & Simó, J. (2005a). Can-based distributed control architecture using the scocan communication protocol. In *Proc. IEEE Int'l Conf. on Emerging Tchnologies and Factory Automation*. ETFA'2005 volume 1.

Coronel, J., Blanes, F., Pérez, P., Benet, G., & Simó, J. (2004). Arquitectura de control distribuida usando nodos empotrados con rt-linux sobre el protocolo de comunicaciones scocan. In *XXV Jornadas de Automática*. Ciudad Real, España: Universidad de Castilla la Mancha.

Coronel, J., Blanes, F., Pérez, P., Benet, G., & Simó, J. (2006a). Scocan: Un protocolo de comunicaciones de tiempo real para sistemas empotrados distribuidos. aplicación al control de robots. *Revista Iberoamericana de Automática e Informática Industrial (RIAI)*, *3*, 71–78.

Coronel, J., Blanes, F., Pérez, P., & Simó, G. B. J. (2006b). Scocan, un protocolo flexible basado en can. *Revista Energía y Computación. Edición Especial*, *14*, 55–60.

Coronel, J., Pérez, P., Benet, G., Blanes, F., & Simó, J. (2005b). Un protocolo flexible sobre el bus can para sistemas distribuidos de tiempo real. In *VII Jornadas internacionales de las ciencias computacionales*. Colima, México: Universidad de Colima.

Coronel, J., Pérez, P., Benet, G., Blanes, F., Simó, J., & Crespo, A. (2006c). Scocan: A communication protocol for distributed real time systems. In *14th International conference on real-time and network systems (RTNS)*. Poitiers, France.

Coronel, J., Simó, J., Posadas, J., & Blanes, F. (2006d). Sistemas distribuidos de tiempo real desde una perspectiva basada en sistemas multiagentes con movilidad. In *XXVII Jornadas de Automática*. Almería: Universidad de Almería.

Coronel, J., Tsagkaropoulos, M., & Mylonas, D. (2013). Validation of securely partitioned systems over multicore architectures based on xtratum. In *Proceedings of DASIA 2013. Data Systems In Aerospace* 18. Porto, Portugal: ESA. European Space Agency.

Coronel, J. O. (2008). *Arquitectura de Control para el desarrollo de aplicaciones bajo restricciones de tiempo real*. Technical Report STR-2008-35 DISCA - Universidad Politécnica de Valencia Valencia (España).

Coronel, J. O., Blanes, F., Simó, J., & Nicolau, V. (2008). Middleware de kernel de control para el desarrollo de aplicaciones en sistemas empotrados de tiempo real. In *XXIX Jornadas de Automática*. Tarragona: Universitat Rovira i Virgili.

Coronel, J. O., & Simó, J. E. (2012). High performance dynamic voltage/frequency scaling algorithm for real-time dynamic load management. *J. Syst. Softw.*, *85*, 906–919.

Coronel, J. O., Ten, J. S., & Blanes, F. (2009). Modelos de planificabilidad para la reducción del consumo energético de cpu aplicado al movimiento de código en sistemas de tiempo real. In *XII Jornadas de Tiempo Real (JTR)*. Madrid, España: Universidad Carlos III de Madrid.

Corporation, I. (2003). Developer's manual. In *Intel PXA255 Processor*.

Corporation, I. (2006). Developer's manual. In *Intel PXA27x Processor Family*.

Corporation, T. (2000). Tm5400 processor spceifications.

Corporation, T. (2008). www.transmeta.com.

Crespo, A., Albertos, P., Balbestre, P., Vallés, M., Lluesma, M., & Simó, J. (2006). Schedulability issues in complex embedded control systems. *IEEE International Conference on Control Applications*, .

Crespo, A., Masmano, M., Coronel, J., Peiro, S., Balbestre, P., & Simó, J. (2014). Multicore partitioned systems based on hypervisor. In *IFAC World Congress 2014*. Cape Town, South Africa.

Culver, V., & Khatri, S. (2005). A dynamic voltage scaling algorithm for energy reduction in hard real-time systems. In *Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific* (pp. 842–845 Vol. 2). volume 2.

Dharwar, D., Bhat, S., Srinivasan, V., Sarma, D., & Banerjee, P. (2012). Approaches towards energy-efficiency in the cloud for emerging markets. In *Cloud Computing in Emerging Markets (CCEM), 2012 IEEE International Conference on* (pp. 1–6).

Duan, C., & Khatri, S. P. (2006). Computing during supply voltage switching in dvs enabled real-time processors. In *IEEE ISCAS Conference* (pp. 2254–2259).

Emberson, P., & Bate, I. (2007). Minimising task migration and priority changes in mode transitions. In *RTAS '07: Proceedings of the 13th IEEE Real Time and Embedded Technology and Applications Symposium* (pp. 158–167). Washington, DC, USA: IEEE Computer Society.

Farias, G., Årzén, K.-E., & Cervin, A. (2007). Interactive real-time control labs with TrueTime and Easy Java simulations. In *Proc. International Multiconference on Computer Science and Information Technology, International Workshop on Real Time Software*. Wisla, Poland.

Fateh, B., & Govindarasu, M. (2015). Joint scheduling of tasks and messages for energy minimization in interference-aware real-time sensor networks. *Mobile Computing, IEEE Transactions on*, *14*, 86–98.

Galizzi, J., Metge, J.-J., Arberet, P., Morand, E., (CNES), F. V., Crespo, A., Masmano, M., Coronel, J., Ripoll, I., de Valencia UPV), V. B. U. P., Roubert, F., Scuri, C., (CS-SI), V. T., & (SOGETI), T. N. (2011). Porting of lvcugen (tsp-based solution) on cpu itar-free board. *DASIA*, .

Galizzi, J., Metge, J.-J., Arberet, P., Morand, E., (CNES), F. V., Crespo, A., Masmano, M., Coronel, J., Ripoll, I., de Valencia UPV), V. B. U. P., Roubert, F., Scuri, C., (CS-SI), V. T., & (SOGETI), T. N. (2012). Lvcugen (tsp-based solution) and first porting feedback. *Embedded Real Time Software and Systems(ERTS)*, .

Gaujal, B., & Navet, N. (2007). Dynamic voltage scaling under edf revisited, .

Group, O. M. (Dec. 2001). The common object request broker: Architecture and specification.

Guo, H., Hu, K., & Xia, T. (2014). Energy-efficient co-scheduling of receiving packets and decoding tasks on mobile video streaming terminals. In *Multimedia and Expo Workshops (ICMEW), 2014 IEEE International Conference on* (pp. 1–6).

Hang, Y., & Hansson, H. (2011). Timing analysis for a composable mode switch. *SIGBED Rev.*, *8*, 15–18.

Intel (2004). Enhanced intel speedstep technology for the intel pentium m processor. In *White Paper*.

Interface, A. A. S. S. (2003). *ARINC653, Arinc Specification 653-1*. RTCA.

Jejurikar, R., & gupta, R. (2002). Energy aware task scheduling with task synchronization for embedded real time systems. In *Internacional Conference Compilers, Architecture and Synthesis for Embedded Systems*.

Jejurikar, R., & Gupta, R. (2004). Dynamic voltage scaling for systemwide energy minimization in real-time embedded systems. In *ISLPED '04: Proceedings of the 2004 international symposium on Low power electronics and design* (pp. 78–81). New York, NY, USA: ACM.

J.L.Posadas, J.L.Poza, J.E.Simó, G.Benet, & F.Blanes (2008). Agent based distributed architecture for mobile robot control. *Engineering Applications of Artificial Intelligence*, *21*, 805–823.

Kadayif, I., Kandemir, M., Narayanan, V., Irwin, M., & Kolcu, I. (2004). Exploiting processor workload heterogeneity for reducing energy. In *Design, Automation and Test in Europe Conference and Exhibition'04*.

Khaitan, S., & McCalley, J. (2015). Design techniques and applications of cyberphysical systems: A survey. *Systems Journal, IEEE*, *9*, 350–365.

Kim, J., Yoo, S., & Kyung, C.-M. (2011). Program phase-aware dynamic voltage scaling under variable computational workload and memory stall environment. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, *30*, 110–123.

Kim, W., Shin, D., Yun, H.-S., Kim, J., & Min, S. L. (2002). Performance comparison of dynamic voltage scaling algorithms for hard real-time systems. In *RTAS '02: Proceedings of the Eighth IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'02)* (p. 219). Washington, DC, USA: IEEE Computer Society.

Kumar, G. S. A., Manimaran, G., & Wang, Z. (2008). End-to-end energy management in networked real-time embedded systems. *IEEE Trans. Parallel Distrib. Syst.*, *19*, 1498–1510.

Lee, E. A. (2006). Cyber-physical systems – are computing foundations adequate? *NFS Workshop On Cyber-Physical Systems*, .

Lee, H.-W., Kim, K.-H., Choi, Y.-K., Sohn, J.-H., Park, N.-K., Kim, K.-W., Kim, C., Choi, Y.-J., & Chung, B.-T. (2012). A 1.6 v 1.4 gbp/s/pin consumer dram with self-dynamic voltage scaling technique in 44 nm cmos technology. *Solid-State Circuits, IEEE Journal of*, *47*, 131–140.

Lehoczky, J., Sha, L., & Ding, Y. (1989). The rate-monotonic scheduling algorithm: Exact characterization and average case behavior. In *10th IEEE Real-Time Systems Symposium* (pp. 166–172).

Leung, L.-F. (2005). Exploiting dynamic workload variation in low energy, .

Li, T., & Ding, C. (2001). Instruction balance and its relation to program energy consumption. In *Internaional Workshop Languages and Compilers for Parallel computing*.

Liang, W.-Y., Chen, S.-C., Chang, Y.-L., & Fang, J.-P. (2008). Memory-aware dynamic voltage and frequency prediction for portable devices. In *Embedded and Real-Time Computing Systems and Applications, 2008. RTCSA '08. 14th IEEE International Conference on* (pp. 229–236).

Liu, Y., & Mok, A. (2003). An integrated approach for applying dynamic voltage scaling to hard real-time systems. In *9th IEEE Real-Time and Embedded Technology and Applications Symposium*. Toronto, Canada.

Lluesma, M., Cervin, A., Balbastre, P., Ripoll, I., & Crespo, A. (2006). Jitter evaluation of real-time control systems. *rtcsa*, *00*, 257–260.

Marinoni, M., & Buttazzo, G. (2007). Elastic dvs management in processors with discrete voltage/frequency modes. *IEEE Transactions on Industrial Informatics*, *3*.

Martínez, P., Coronel, J., Blanes, J., Simó, J., Albero, M., & Benet, G. (2007). Low-cost distributed embedded control systems. In *8 IFAC Symposium on Cost Oriented Automation*. Cuba.

Marvell (2008). System and timer configuration developers manual. In *Marvell PXA3xx Processor Family*. volume I.

Masmano, M., Coronel, J., Balbastre, P., Crespo, A., Simo, J., & Peiro, S. (2013). Xtratum hypervisor for mixed-criticality systems: Configuration and deployment. *RTSS 2013: IEEE Real-Time Systems Symposium*, .

Mehariya, S., Songara, D., Mitra, V., & Govil, M. (2014). An approach for a reduced response time and energy consumption in mixed task set using a priority exchange server. In *Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on* (pp. 62–67).

Mejia-Alvarez, P., Levner, E., & Mosse, D. (2002). Power-optimized scheduling server for real-time tasks. In *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE* (pp. 239–250).

Mejia-Alvarez, P., Levner, E., & Mossé, D. (2004). Adaptive scheduling server for power-aware real-time tasks. *ACM Trans. Embed. Comput. Syst.*, *3*, 284–306.

Nicolau, V., Coronel, J. O., Blanes, F., & Simó, J. (2008). Implementación de un middleware de control para sistemas con recursos límitados. In *XXIX Jornadas de Automática*. Tarragona: Universitat Rovira i Virgili.

Ohlin, M., Henriksson, D., & Cervin, A. (2007). *TrueTime 1.5—Reference Manual*.

Perez, J., Gonzalez, D., Nicolas, C., Trapman, T., & Garate, J. (2014a). A safety certification strategy for iec-61508 compliant industrial mixed-criticality systems based on multicore partitioning. In *Digital System Design (DSD), 2014 17th Euromicro Conference on* (pp. 394–400).

Perez, J., Gonzalez, D., Trujillo, S., Trapman, A., & Garate, J. M. (2014b). Safety concept for a wind power mixed-critically embedded system based on multicore partitioning. *11th International Symposium. Functional Safety in Industrial Applications (TUV Rheinland)*, .

Piao, X., Kim, H., Cho, Y., Park, M., Han, S., Park, M., & Cho, S. (2009). Energy consumption optimization of real-time embedded systems. In *ICESS '09: Proceedings of the 2009 International Conference on Embedded Software and Systems* (pp. 281–287). Washington, DC, USA: IEEE Computer Society.

Pillai, P., & Shin, K. (2001). Real-time dynamic voltage scaling for low-power embedded operating systems. In *ACM symposium on operating systems principles*. Banff, Canada.

Pouwelse, J., Langendoen, K., & Sips, H. (2001). Dynamic voltage scaling on a low-power microprocessor. In *Seventh Internacional Conference Mobile Computing and Networking (MOBICOM)*.

Quan (2004). Fixed priority scheduling for reducing overall energy on variable voltage processors, .

Racu, R., Jersak, M., & Ernst, R. (2005). Applying sensitivity analysis in real-time distributed systems. In *In 11th IEEE Real-Time Technology and Applications Symposium (RTAS?05)* (pp. 160–169). IEEE Computer Society.

Real, J., & Crespo, A. (2004). Mode change protocols for real-time systems: A survey and a new proposal. *Real-Time Syst.*, *26*, 161–197.

Saewong, S., & Rajkumar, R. R. (2003). Practical voltage-scaling for fixed-priority rt-systems. In *RTAS '03: Proceedings of the The 9th IEEE Real-Time and Embedded Technology and Applications Symposium* (p. 106). Washington, DC, USA: IEEE Computer Society.

Sánchez, E., Munoz, M., Posada, J. L., Poza, J. L., & Blanes, J. (2014). Integration of mobile robot navigation on a control kernel middleware based system. In S. Omatu, H. Bersini, J. M. Corchado, S. Rodríguez, P. Pawlewski, & E. Bucciarelli (Eds.), *Distributed Computing and Artificial Intelligence, 11th International Conference* (pp. 477–484). Springer International Publishing volume 290 of *Advances in Intelligent Systems and Computing*.

Santos, J., Lima, G., Bletsas, K., & Kato, S. (2013). Multiprocessor real-time scheduling with a few migrating tasks. In *Real-Time Systems Symposium (RTSS), 2013 IEEE 34th* (pp. 170–181).

Schantz, R. E., Schmidt, D. C., Loyall, J. P., & Rodrigues, C. (2006). Controlling qos in distributed real-time and embedded systems via adaptive middleware. In *Wiley Software Practice and Experience journal special issue on Experiences with Auto-adaptive and Reconfigurable Systems* 36:11-12 (pp. 1189–1208). Toronto, Canada.

Schmidt, D. (Dec. 1993 and June 1994). The adaptive communication environment: Object-oriented network programming components for developing client/server applications.

Schmidt, D., & Kuhns, F. (June 2000). An overview of the real-time corba specification. *Special Issue on Object-oriented Real-time Computing*, *33*.

Schmidt, D., Stal, M., Rohnert, H., & Buschmann, F. (2000). Pattern-oriented software architecture: Patterns for concurrent and networked objects.

Scordino, C., & Lipari, G. (2006). A resource reservation algorithm for power-aware scheduling of periodic and aperiodic real-time tasks. *IEEE Trans. Comput.*, *55*, 1509–1522.

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

Shin, Y., Choi, K., & Sakurai, T. (2000). Power optimization of real-time embedded systems on variable speed processors. In *ICCAD '00: Proceedings of the 2000 IEEE/ACM international conference on Computer-aided design* (pp. 365–368). Piscataway, NJ, USA: IEEE Press.

Simarro, R., Coronel, J., Simó, J., & Blanes, J. (2008). Hierarchical and distributed embedded control kernel. In *17th IFAC World Congress*. Seoul, Korea.

Sloss, A. N., Symes, D., & Wright, C. (2004). *Arm System Developer's Guide (Designing and Optimizing System Software)*. (Primera ed.). Elsevier.

Stangaciu, C., Horvath, A., Micea, M., Cretu, V., & Groza, V. (2015). Analysis and improvements in energy consumption models for rts. In *Applied Computational Intelligence and Informatics (SACI), 2015 IEEE 10th Jubilee International Symposium on* (pp. 277–282).

Stangaciu, C., Micea, M., & Cretu, V. (2013a). Energy efficiency in real-time systems: A brief overview. In *Applied Computational Intelligence and Informatics (SACI), 2013 IEEE 8th International Symposium on* (pp. 275–280).

Stangaciu, V., Pescaru, D., Micea, M., & Cretu, V. (2013b). Practical aspects regarding implementation of real time wsn applications based on ieee 802.15.4. In *Telecommunications Forum (TELFOR), 2013 21st* (pp. 295–298).

Troger, P., Werner, M., & Richling, J. (2015). Cyber-physical operating systems - what are the right abstractions? In *Embedded Computing (MECO), 2015 4th Mediterranean Conference on* (pp. 13–16).

Tsai, Y.-H., Wang, K., & Chen, J.-M. (2007). A deferred-workload-based inter-task dynamic voltage scaling algorithm for portable multimedia devices. In *Proceedings of the 2007 International Conference on Wireless Communications and Mobile Computing* IWCMC '07 (pp. 677–682). New York, NY, USA: ACM.

Wang, X., Huang, H., Subramonian, V., Lu, C., & Gill, C. (2004). Camrit:control-based adaptive middleware for real-time image transmission. In *In Proceedings of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. Toronto, Canada.

Windsor, J., & Hjortnaes, K. (2009). Time and space partitioning in spacecraft avionics. *Space Mission Challenges for Information Technology, IEEE International Conference on*, *0*, 13–20.

Wolf, W. (2009). Cyber-physical system. *IEEE Computer. Innovative Techonolgy for Computer Professionals.*, *42*, 88–89.

Xia (2008). Control-theoretic dynamic voltage scaling for embedded controllers, .

Xia, F., & xian Sun, Y. (2008). *Control and Scheduling Codesign* volume XVI. Springer.

Xia, F., & Sun, Y. (2008). Control-scheduling codesign: A perspective on integrating control and computing. *arXiv preprint arXiv:0806.1385*, .

Xu, Y., Arzen, K.-E., Cervin, A., Bini, E., & Tanasa, B. (2015). Exploiting job response-time information in the co-design of real-time control systems. In *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2015 IEEE 21st International Conference on* (pp. 247–256).

Yazdi, H., Salmani, H., Khatib-Astaneh, N., Salmani, M., & Fard, A. (2008). Exploiting laxity for heterogeneous multiprocessor real-time scheduling. In *Information and Communication Technologies: From Theory to Applications, 2008. ICTTA 2008. 3rd International Conference on* (pp. 1–6).

Yi, J., Poellabauer, C., Hu, X. S., Simmer, J., & Zhang, L. (2009). Energy-conscious co-scheduling of tasks and packets in wireless real-time environments. In *RTAS '09: Proceedings of the 2009 15th IEEE Real-Time and Embedded Technology and Applications Symposium* (pp. 265–274). Washington, DC, USA: IEEE Computer Society.

Yun, H.-S., & Kim, J. (2003). On energy-optimal voltage scheduling for fixed-priority hard real-time systems. *ACM Trans. Embed. Comput. Syst.*, *2*, 393–430.

Zhang, F., Szwaykowska, K., Wolf, W., & Mooney, V. (2008). Task scheduling for control oriented requirements for cyber-physical systems. In *Real-Time Systems Symposium, 2008* (pp. 47–56).

Zhang, R., Lu, C., Abdelzaher, T., & Stankovic, J. (2002). Controlware: A middleware architecture for feedback control of software performance. In *Proceedings of the 2002 International Conference on Distributed Computing Systems*. Vienna, Austria.

Zhong, Xiliang, Xu, & Cheng-Zhong (2007). Frequency-aware energy optimization for real-time periodic and aperiodic tasks. *SIGPLAN Not.*, *42*, 21–30.

Zhou, J., & Wei, T. (2015). Stochastic thermal-aware real-time task scheduling with considerations of soft errors. *J. Syst. Softw.*, *102*, 123–133.

Zhu, D., Melhem, R., & Mossé, D. (2004). The effects of energy management on reliability in real-time embedded systems. In *International Conference on Computer-Aided Design*.