53/2002-03

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
UNIDAD DE TERCER CICLO Y POSTGRADO

Reunido el día de la fecha, el Tribunal nombrado por el Excmo. Sr. Rector Magfco. de esta Universidad, el/a aspirante expuso esta TESIS DOCTORAL.

Terminada la lectura y contestadas por el/a Doctorando/a las objeciones formuladas por los señores miembros del Tribunal, éste calificó dicho trabajo con la nota de SOBRESALIENTE

CUM LAUDE POR UNANIMIDAD

Las Palmas de Gran Canaria, a 23 de mayo de de 2003.

El/a Presidente/a: Dr.D. Juan Ángel Méndez Rodríguez,

El/a Secretario/a: ~~Dr.~~D. Francisco Mario Hernández Tejera,

El/a Vocal: Dr.D. Blas Galván González,

El/a Vocal: Dr.D. Vicente Matellán Olivera,

El/a Vocal: Dr.D. Humberto Martínez Barberá,

El Doctorando: D. José Daniel Hernández Sosa,

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Departamento de Informática y Sistemas

TESIS DOCTORAL

ADAPTACIÓN COMPUTACIONAL EN SISTEMAS PERCEPTO-EFECTORES. PROPUESTA DE ARQUITECTURA Y POLÍTICAS DE CONTROL

José Daniel Hernández Sosa

Las Palmas de Gran Canaria

Marzo de 2003

AGRADECIMIENTOS

Quiero agradecer a los directores de esta tesis, Jorge Cabrera Gámez y Antonio Falcón Martel, el apoyo y los consejos recibidos en la realización de la misma. A Jorge en particular, quiero agradecerle sinceramente su dedicación y entusiasmo, fundamentales a lo largo de todo este tiempo.

También quiero agradecer a todos los miembros del antiguo grupo de investigación "Grupo de Inteligencia Artificial y Sistemas", por su ayuda.

Esta memoria quiero dedicarla a mi familia, en especial a Carmen por su paciencia y comprensión.

Gracias a todos.

A Carmen

y

a mis padres

Índice general

Índice de figuras

Índice de Tablas

Índice de Algoritmos

Resumen

La tarea de diseñar e implementar sistemas percepto-efectores robustos requiere contribuciones procedentes de múltiples disciplinas. Aunque a corto plazo puedan alcanzarse rendimientos significativos con aplicaciones ad hoc, el éxito en el largo plazo pasa por el uso de metodologías de desarrollo que permitan la construcción de sistemas robustos, capaces y adaptables. En este sentido son deseables aspectos como la promoción del diseño modular y la reutilización del código en aras de acortar los tiempos de puesta en funcionamiento de nuevas aplicaciones y facilitar el mantenimiento y modificación de las existentes. Por otra parte, y como característica distintiva, un sistema percepto-efector está sometido generalmente a fuertes restricciones temporales y de recursos, las cuales deben ser controladas adecuadamente para permitir que el sistema se adapte dinámicamente a los cambios del entorno. En caso contrario, la ausencia de control puede llegar a provocar una degradación catastrófica que limite seriamente la funcionalidad del sistema. Es éste un aspecto frecuentemente ignorado en el diseño de arquitecturas de control de este tipo de sistemas.

En esta tesis se propone una organización modular para el diseño de sistemas percepto-efectores. Esta estructura permite definir diferentes arquitecturas multinivel distribuidas basadas en un conjunto reducido de objetos: sensores para la captación de los datos, diagnósticos para su procesamiento, acciones para evaluar los resultados del cómputo y generar comandos como respuesta y actuadores para hacer operativos los comandos. El conjunto se completa con módulos supervisores, que realizan tareas de coordinación y control. La configuración interna es común a todos los módulos, y está constituida por la interconexión de tres unidades funcionales: la unidad BU encargada del procesamiento, la unidad TD para el control y la unidad COM que gestiona las comunicaciones. Se dispone de un lenguaje de programación simple, basado en un conjunto reducido de instrucciones, para la implementación de cada tipo de módulo.

Se presenta una organización del cómputo basada en estados y tareas para el control de la misión a más alto nivel. Las tareas se componen mediante una combinación

de acciones y diagnósticos, siendo posible la introducción de valores de frecuencia de funcionamiento y prioridad a nivel de tarea.

En esta tesis se propone, de forma integrada con la arquitectura, un esquema de adaptación de bajo nivel que permite ajustar los recursos demandados a las capacidades computacionales disponibles en cada momento. Se analizan los diferentes problemas de control planteables, las acciones de control disponibles y su integración en políticas de control que las coordinen. De esta forma, se consigue que el sistema degrade y promocione en función de los recursos existentes de manera controlada. Se contempla tanto el caso de los sistemas calibrados como los no calibrados, y se realiza un análisis particular para contextos con multiprocesamiento.

Se incluyen modelos para la estimación de la carga del sistema, a nivel local y global. En el caso de distribución del cómputo se definen modelos para estimar el tiempo de procesamiento en los diferentes esquemas de balanceo de carga. Se proponen además medidas para la caracterización de aspectos del sistema como su nivel de calibración o las capacidades de adaptación del mismo.

Dentro de la estrategia de control, se arbitran políticas tanto locales como globales, las cuales se ofrecen como una utilidad al diseñador a través de los diferentes parámetros de configuración. En las definiciones de las tareas, pueden añadirse valores de tolerancia a las violaciones temporales, con el propósito de que sean utilizados en las políticas de control. A nivel de módulos es posible incluir información sobre los recursos de degradación disponibles, en caso de existir.

Finalmente, se propone la inclusión de mecanismos de aprendizaje basado en casos y se presenta una configuración de control integrado que combina el aprendizaje con los procesos de autocalibración en el esquema de control global del sistema.

Las diferentes propuestas realizadas a lo largo de la tesis se ilustran en un conjunto de experimentos para su discusión y validación.

Capítulo 1

Introducción

Una de las principales características de un ser vivo es la de reaccionar, bien de forma refleja bien de forma deliberada, ante cambios detectados internamente y en su entorno. Los seres artificiales, en su afán por emular las capacidades biológicas, tratan de incorporar esta funcionalidad con un nivel de competencia comparable. La tarea no es en modo alguno sencilla. Un sistema biológico es capaz de reaccionar de forma adecuada aún en casos de falta de información o en presencia de datos imprecisos, procesando y registrando de forma automática las experiencias pasadas para tratar de mejorar constantemente su rendimiento. La robustez y el aprendizaje son, por lo tanto, propiedades igualmente deseables y perseguidas por los sistemas artificiales. Los sistemas percepto-efectores son claros exponentes de estos intentos de emulación.

Se han propuesto múltiples definiciones de sistemas percepto-efectores, también denominados agentes inteligentes autónomos o simplemente agentes autónomos. A continuación se incluyen algunas de ellas:

"*Un agente es cualquier cosa que perciba su entorno a través de sensores y actúe sobre el mismo a través de efectores*" [Russell y Norvig, 1995]

"*Los agentes autónomos son sistemas computacionales que habitan complejos entornos dinámicos, perciben y actúan autónomamente en los mismos y, de esta forma, alcanzan un conjunto de metas o tareas para las que han sido diseñados*" [Maes, 1995]

"*Los agentes inteligentes realizan continuamente tres funciones: percepción de condiciones dinámicas en el entorno; acción para modificar las condiciones en ese entorno; y razonamiento para interpretar percepciones, resolver problemas, plantear inferencias y determinar acciones*" [Hayes-Roth, 1995]

"*Los agentes autónomos son sistemas capaces de realizar acciones autónomamente y*

con propósito en el mundo real" [Franklin, 1995]

Los sistemas percepto-efectores constituyen un área de investigación en la que confluyen técnicas de diferentes disciplinas como son la Inteligencia Artificial, los Sistemas Operativos y el Control Automático. Las limitaciones de recursos a las que se ven sometidos normalmente este tipo de sistemas hacen que el cumplimiento de misiones de forma robusta requiera de una cuidada armonización de las técnicas disponibles en cada una de las áreas y niveles de actuación posibles.

A raíz de la proliferación de sistemas y aplicaciones cada vez más complejas se ve clara la necesidad de imponer organizaciones modulares en los desarrollos. Una vez alcanzados los objetivos de eficacia y competencia a nivel técnico y táctico, se plantean objetivos estratégicos a más largo plazo. El propósito es ahora establecer bases metodológicas desde las que, por un lado, se simplifique la generación de las aplicaciones actuales, y por otro, puedan abordarse metas más ambiciosas. Como vía para trabajar eficientemente con la complejidad se proponen estructuras modulares que se combinan en arquitecturas sobre las que implementar aplicaciones de una manera sistemática. De esta forma se facilita su depuración y modificación, además de favorecer una mayor reutilización del código.

Paralelamente se han dedicado esfuerzos de investigación a mejorar la capacidad de adaptación de los sistemas percepto-efectores con vistas a conseguir un comportamiento más robusto. De los sistemas caracterizados por un funcionamiento que se prolonga durante periodos extensos, si no indefinidos, más que el alcanzar un pico de rendimiento elevado en un instante determinado interesa la evolución media. En esta línea, se persigue el diseño de mecanismos de control y regulación que proporcionen un comportamiento promedio aceptable. Como consecuencia, el control adaptativo puede sacrificar el aprovechar al 100 % las situaciones de recursos sobrantes, en favor de un mejor rendimiento cuando éstos escasean.

Lo que sí es exigible en cualquier caso es un rendimiento mínimo, a fin de preservar aspectos como la seguridad y la integridad del sistema. No se aspira, lógicamente a que el sistema sea capaz de reponerse ante cualquier circunstancia adversa, pero al menos debe poder identificar correctamente su estado. En caso extremo se llegaría a una situación de "fallo consciente", de la que el sistema no puede salir sin ayuda exterior, pero sí aportar información suficiente para determinar cómo se llegó a ese estado.

Considerando el comportamiento dinámico del sistema en situaciones de alteración de los recursos disponibles, los mecanismos de adaptación deben imponer una evolución suave y homogénea. Es deseable, por ejemplo, que ante un recorte en la potencia

computacional utilizable, el sistema degrade de forma paulatina, reduciendo progresivamente su rendimiento. El sistema debe mantenerse operativo, siendo la calidad de los resultados producidos la que se ve afectada negativamente por las circunstancias. La evolución suave del sistema lo hace más predecible, lo que reduce la incertidumbre e incrementa la seguridad.

Normalmente, los aspectos ligados a la adaptación, especialmente a bajo nivel, se abordan de forma independiente a la estructuración y organización de los sistemas. De forma que son pocas las aproximaciones que tratan de incluir ambas vertientes de forma integrada.

En este ámbito de problemas se sitúa la presente tesis. Se trata de la propuesta, implementación y prueba de una arquitectura para sistemas percepto-efectores que cumpla con los objetivos que se explicitan a continuación:

- Estructurales:
  - Arquitectura modular flexible.
  - Distribución funcional claramente identificable.
  - Integración de diferentes fuentes de información.
  - Reactividad.
- Adaptación / Robustez:
  - Ajuste de la precisión de los resultados a la capacidad de cómputo disponible.
  - Estabilidad del sistema.
  - Mecanismos alternativos de procesamiento.
- Observabilidad:
  - Monitorización del funcionamiento del sistema.
  - Informe de la calidad del resultado alcanzado.
  - Predecibilidad de los tiempos de respuesta.
  - Notificación de errores.
- Validez / Aplicabilidad:
  - Combinación de ciclos de procesamiento con tiempos de respuesta variable.
  - Manejo de prioridades.

- Reutilización del código.

De todos ellos, son los objetivos relacionados con la modularidad y la adaptabilidad los que serán tratados con mayor profundidad en este trabajo.

El presente documento se estructura en seis capítulos. En el capítulo 2 se describe el contexto de la tesis con una revisión de diferentes tendencias en arquitecturas robóticas y estrategias de comportamiento adaptativo. Se incluyen ejemplos representativos de diversos trabajos relacionados.

El capítulo 3 presenta el sistema propuesto en esta tesis. En primer lugar se realiza una descripción desde el punto de vista de la organización estructural, definiendo los elementos modulares empleados. A continuación se presenta la propuesta desde la perspectiva funcional.

En el capítulo 4 se profundiza en los aspectos ligados a la adaptación. Se describen las distintas técnicas que van a emplearse y su coordinación dentro de las diferentes políticas de control implementadas. También se presentan aspectos ligados al control distribuido y el aprendizaje.

El capítulo 5 está dedicado a los experimentos, donde se analiza la influencia de diferentes factores sobre el comportamiento de los mecanismos de adaptación. Se incluye una aplicación del sistema propuesto en un entorno real.

Finalmente, en el capítulo 6 se presentan los resultados y las conclusiones del trabajo, así como las futuras líneas de desarrollo.

Capítulo 2

Elementos de la Adaptación Computacional: Arquitecturas y Lenguajes

En este capítulo se sitúa el contexto de la tesis. Se presenta una revisión de las principales propuestas de arquitecturas para sistemas percepto-efectores. Paralelamente se introducen los conceptos relativos a la adaptación computacional y el comportamiento robusto en este tipo de sistemas.

2.1. Introducción

Un sistema percepto-efector es aquél que captura información, tanto de su entorno como interna al propio sistema, la procesa y analiza, y modifica su comportamiento en base a los resultados obtenidos. Dentro de esta definición general se incluyen multitud de sistemas como son:

- Sistemas biológicos.
- Sistemas robóticos.
  - Sistemas robóticos móviles, sistemas de visión activa, manipuladores, etc.
- Sistemas de control de procesos.

Se trata de sistemas que evolucionan en un entorno cambiante, afectados por una serie de condicionantes adversos como son un elevado grado de incertidumbre tanto en la

percepción del entorno como del estado interno, la disponibilidad de recursos limitados, o las fuertes restricciones exigidas a los tiempos de respuesta. La eficacia del cumplimiento de su misión depende, pues, de mantener un constante y delicado equilibrio: tomar una decisión de control adecuada a partir de unos resultados de calidad aceptable en un tiempo suficientemente corto.

A pesar de lo indicado anteriormente, los sistemas robóticos han sido objeto de notables avances en los últimos años, lo que ha conducido a una mejora significativa en sus capacidades y ámbitos de aplicación. Los resultados obtenidos alcanzan niveles elevados de eficacia y eficiencia en aplicaciones cada vez más exigentes [Kortenkamp et al., 1998]. Algunos ejemplos de aplicaciones exitosas incluyen los robots para limpieza [Prassler et al., 2000] [Simoncelli et al., 2000], exploración de áreas siniestradas o de difícil acceso [Kato y Hirose, 2001] [Iagnemma et al., 2001], robots autónomos para museos [Thrun et al., 1999] [Domínguez-Brito et al., 2001], seguimiento visual en tiempo real [Paulus et al., 2000], etc.

Como consecuencia de esta proliferación de sistemas surgen nuevas demandas desde las que un primer análisis pone en evidencia la necesidad de disponer de herramientas que faciliten tanto el diseño y la implementación como la monitorización del rendimiento en este tipo de aplicaciones [Kortenkamp et al., 2001] [Kortenkamp et al., 2002]. A largo plazo, los conceptos de facilidad de mantenimiento y modificación de los sistemas, así como la reutilización de desarrollos previos cobran una relevancia cada vez mayor. En términos de objetivos a alcanzar podemos hablar de objetivos tácticos y estratégicos. Dentro de las metas en el plano táctico se encontrarían las siguientes:

- Robustez.
- Alto grado de autonomía.
- Capacidad para alcanzar metas propuestas.
- Respuesta en tiempo real a los cambios del entorno.
- Capacidad para seguir simultáneamente múltiples objetivos.

En el horizonte estratégico cabe resaltar los siguientes puntos:

- Facilidad de mantenimiento.
- Reutilización del código.

- Facilitar futuras extensiones.
- Alcanzar un uso óptimo de los recursos computacionales disponibles.

Un requisito fundamental para alcanzar los objetivos planteados es la utilización de una arquitectura de referencia en el diseño e implementación del sistema [Coste-Manière y Simmons, 2000]. Una arquitectura claramente definida impone restricciones en el diseño que reportan beneficios en las etapas previas a la puesta en marcha del sistema, durante su ejecución, así como posteriormente en la validación del mismo. Lógicamente, las mencionadas restricciones deben ser entendidas como guías o recomendaciones de programación disciplinada, sin llegar al extremo de condicionar tanto el diseño del sistema que impida su utilización práctica.

Dentro de un contexto como en el que nos encontramos de aplicaciones enormemente variables y complejas, pensamos que no tiene sentido buscar una solución que se adapte a todas ellas. Siempre existirán casos particulares en los que una aplicación se adapte mejor a una determinada arquitectura que a otra. Sin embargo, resultará beneficioso disponer al menos de un conjunto de alternativas entre las que poder seleccionar la más adecuada. También existen esquemas de organización modulares flexibles que no imponen ninguna configuración previa, permitiendo adaptarse a diferentes aplicaciones.

La investigación en este área ha conducido a la presentación de múltiples propuestas. Con una visión general, podemos clasificar las arquitecturas robóticas en tres grandes categorías:

- Arquitecturas deliberativas o jerárquicas.
- Arquitecturas reactivas o basadas en comportamientos.
- Arquitecturas híbridas.

Las arquitecturas deliberativas se fundamentan en la utilización de un esquema ordenado de percepción, planificación y acción. Constituyen una configuración de tipo vertical (*bottom-up*) en la que la toma de decisiones se basa en la confrontación de los datos de los sensores con un modelo del entorno en constante actualización.

Las arquitecturas reactivas definen una serie de comportamientos que ligan estrechamente la percepción con la acción. Pueden verse como estructuras de tipo horizontal en las que las decisiones pueden tomarse a distintos niveles, sin necesidad de mantener una representación común del entorno.

Las arquitecturas híbridas son un intento de combinar la deliberación con la reacción de forma que se aproveche lo mejor de ambos planteamientos. Se trata de la línea que aglutina un mayor número de propuestas, impulsadas por diversos factores entre los que destacan la plausibilidad biológica, el rango de aplicación o el éxito alcanzado por muchas de sus implementaciones.

2.2. Arquitecturas Robóticas: Deliberación o Reacción

En esta sección analizaremos brevemente los antecedentes de las arquitecturas híbridas, representados por las tendencias orientadas a la reacción frente a las puramente deliberativas.

2.2.1. Arquitecturas deliberativas

Los primeros intentos de integrar las técnicas de inteligencia artificial en sistemas robóticos condujeron a la utilización generalizada de arquitecturas jerárquicas basadas en el paradigma de los ciclos de percepción, planificación y acción (SPA, *Sense Plan Act*). En éstos se defiende la necesidad de mantener una representación o modelo del mundo sobre el que mapear los resultados de los sensores. Este modelo permite la reducción del volumen de información a analizar en la etapa de razonamiento. Algunos ejemplos representativos de esta tendencia son los trabajos de Nilsson [Nilsson, 1980] y Moravec [Moravec, 1990].

La dificultad principal a la que tienen que hacer frente las arquitecturas deliberativas es la necesidad de garantizar una correspondencia directa entre el mundo externo y la representación del mismo manejada por el sistema robótico. La consecuencia es que gran parte de los esfuerzos se concentran en conseguir una elevada reactividad del modelo a los cambios del entorno, para que éstos se reflejen adecuadamente; y de forma simultánea se intenta reducir el tiempo de respuesta del sistema ante los cambios en el modelo. En situaciones donde la rapidez con que se producen cambios en el entorno es elevada, estos métodos tienden a producir resultados desfasados en el tiempo.

2.2.2. Arquitecturas reactivas

Ante los problemas mostrados por las propuestas SPA, se lanzan alternativas centradas en tratar de acortar el tiempo de reacción del sistema ante estímulos externos. Las arquitecturas reactivas proponen la construcción de sistemas robóticos a partir de la combinación de comportamientos de percepción/reacción simples. Las características principales que presentan son las siguientes:

- Uso de los comportamientos como bloques básicos de construcción.
- Evitar en lo posible el uso de modelos del mundo, los cuales - de existir - serán siempre locales.
- Inspiración biológica.

Algunos ejemplos representativos de esta propuesta incluyen los vehículos de Braitenberg [Braitenberg, 1984], la arquitectura de supresión ("subsumption architecture") de Brooks [Brooks, 1986] o los esquemas motores ("motor schema") de Arkin [Arkin, 1989].

Un aspecto importante de los sistemas basados en comportamientos es la coordinación de los mismos, puesto que las reacciones a los estímulos se traducen al final en acciones que compiten por un conjunto reducido de actuadores. Las clases principales de coordinación son los mecanismos selectivos por un lado y los de combinación por otro. Dentro de las estrategias de selección están las conexiones de supresión/inhibición en las que se basa la arquitectura de supresión o las de selección por votos empleadas en la arquitectura distribuida de Rosemblatt DAMN [Rosenblatt, 1995].

La combinación de comportamientos pretende que todas las salidas sean tenidas en cuenta para generar una señal de control por cooperación. Algunas posibilidades incluyen los mecanismos de combinación basados en lógica difusa [Saffiotti et al., 1997], los campos potenciales [Khatib, 1986], o los de composición vectorial, como en el caso de los esquemas motores de Arkin. Otra propuesta interesante es la debida a Schoner, Dose y Engels [Schoner et al., 1996], en la que se incluyen ecuaciones diferenciales para modelar la dinámica de los comportamientos. El trabajo de Pirjanian [Pirjanian, 1998] incluye un análisis en profundidad de las diferentes alternativas en fusión de comportamientos y selección de acciones.

Los sistemas puramente reactivos han demostrado su eficacia en multitud de aplicaciones. Sin embargo, estas arquitecturas presentan también características negativas

[Tsotsos, 1995]. De entre las principales críticas que pueden lanzarse sobre los modelos reactivos se encuentran:

Sensibilidad: La búsqueda de unos tiempos de respuesta tan ajustados puede volverse en contra de la estabilidad del propio sistema si los sensores devuelven lecturas erróneas, aunque sea puntualmente.

Escalado: La falta de modularidad y de mecanismos para manejar la complejidad hacen que se presenten dificultades al afrontar problemas de magnitud creciente.

2.3. Arquitecturas Robóticas Híbridas

Ante los problemas presentados por las tendencias deliberativas ("reaccionar demasiado tarde") y las reactivas ("reaccionar demasiado pronto"), surgen como solución de compromiso las arquitecturas robóticas híbridas. El objetivo es combinar las ventajas de las tendencias anteriores para conseguir sistemas más capaces. Esta combinación no es en modo alguno simple, puesto que hay que poner en correspondencia dispositivos robóticos eminentemente continuos y numéricos con técnicas de inteligencia artificial de naturaleza discreta y simbólica [Bajscy y Large, 1999]. El problema entonces no es si dar más peso a la deliberación o a la reacción, sino cuándo y cuánto (o durante cuánto tiempo) deliberar y cuándo reaccionar [Hayes-Roth, 1993].

Numerosas propuestas se han desarrollado dentro de la corriente híbrida. Sin embargo, es posible encontrar similitudes entre ellas que van más allá de compartir un conjunto de ideas de base [Gat, 1997*b*]. De esta forma, es posible identificar un patrón estructural común en muchos planteamientos que consta de tres componentes principales:

- Nivel reactivo o de comportamientos.
- Nivel ejecutivo o de secuenciación.
- Nivel deliberativo o de planificación.

El nivel reactivo está integrado por bucles de control realimentados que tratan de responder a los estímulos con rapidez. Se constituyen como lazos de control con periodos de funcionamiento cortos que relacionan sensores con actuadores físicos del sistema. El nivel deliberativo incorpora mecanismos de razonamiento que permiten la generación de

planes para resolver las misiones que se le encomienden al sistema. Los periodos típicos de operación son mucho más largos que en el caso del nivel reactivo. El nivel ejecutivo se encarga de enlazar los dos niveles anteriores de forma que los planes del nivel deliberativo se hagan operativos a través de comandos dirigidos al nivel reactivo.

Algunos autores dan un enfoque ligeramente distinto, estableciendo niveles en los comportamientos [Hexmoor et al., 1992], desde capas subconscientes a niveles conscientes.

2.3.1. Nivel reactivo

El nivel reactivo constituye la base funcional de un sistema híbrido. En él se integran las habilidades básicas o comportamientos primitivos de cuya combinación resultan las capacidades del sistema para resolver tareas. Este nivel recibe multitud de denominaciones alternativas tales como conjunto de habilidades básicas, nivel funcional, nivel basado en comportamientos, o controlador.

Algunos ejemplos clásicos de los comportamientos elementales que suelen situarse en este nivel son el seguimiento visual de objetivos, el desplazamiento con evitación de obstáculos, capacidades de navegación reactiva como el seguimiento de paredes o de líneas, etc. Una lista extraída de [Arkin, 1998], sirve de ilustración para la variedad de recursos existentes:

Comportamientos de exploración/direccionales: Movimiento en una dirección general, es el caso de movimientos de cambios de orientación (*head based*) o exploración/vagabundeo (*wandering*).

Comportamientos orientados hacia un objetivo: Movimiento dirigido hacia un cierto atractor. Se incluyen aquí los de atracción hacia un objeto discreto o los movimientos hacia una cierta área atractiva.

Comportamientos aversivos/protectores: Destinados a prevenir colisiones. Entre éstos están los que evitan colisiones con objetos estacionarios, los que esquivan objetos en movimiento (*dodge, escape*) o los que agreden o persiguen al objeto que tienen enfrente (*aggression*).

Comportamientos de seguimiento de caminos: Movimiento a lo largo de un camino diseñado. Seguimiento de caminos, navegación en vestíbulos o seguimiento de líneas (*road following, hallway navigation, stripe following*).

Comportamientos posturales: Relacionados con la estabilidad o el balanceo.

Comportamientos sociales/cooperativos: Tienen que ver con reuniones, incursiones para la búsqueda de alimentos (*foraging*), o conductas de rebaño o manadas (*flocking/herding*).

Comportamientos teleautónomos: Comportamientos coordinados por un operador humano. En éstos se incluyen las conductas de influencia o la modificación de conductas.

Comportamientos perceptuales: Relacionados con ciertos comportamientos de dispositivos, tales como movimientos sacádicos, búsqueda visual o reflejos oculares.

Comportamientos al andar: Se asocian a robots con patas, como es el caso del control del desplazamiento.

Comportamientos de manipulación: Comportamientos asociados a manipuladores o dispositivos de captura de objetos (*grasping, reaching, enveloping*).

Los problemas fundamentales se centran en determinar qué es una conducta o comportamiento básico y qué puede obtenerse como combinación de habilidades más elementales. La solución no es precisamente trivial, y para tratar de alcanzarla se han desarrollado técnicas basadas en diferentes paradigmas. Algunos ejemplos son los siguientes:

Diseño conducido/restringido etológicamente: Se analiza el comportamiento animal para trasladarlo al campo de la robótica. Para ello se parte de un modelo desde un estudio científico sobre un animal, se modifica para hacerlo realizable computacionalmente y se añaden las capacidades senso-motoras. Los resultados del sistema robótico se comparan con el estudio científico biológico y se establecen las oportunas modificaciones.

Diseño basado en actividades situadas: Las acciones del robot están diseñadas en base a situaciones en las cuales se encuentra. Llevada al extremo, con esta técnica se pretende que el sistema robótico tenga la capacidad de analizar y reconocer cada situación particular y seleccionar la acción para cada posible estado del mundo.

Diseño conducido experimentalmente: Los comportamientos se crean y diseñan siguiendo una metodología *bottom-up*. La forma de proceder se inicia con

un conjunto limitado de capacidades o habilidades, se efectúan los experimentos en el mundo real y analiza el resultado. A partir de ahí, se corrigen las conductas defectuosas y se añaden otras nuevas de manera iterativa hasta que el sistema completo presenta un rendimiento satisfactorio.

En este nivel, el modelado del mundo se limita al máximo para evitar los problemas de sincronismo exhibidos por las arquitecturas SPA. En sintonía con los postulados de Brooks: "El mundo es su mejor modelo" [Brooks, 1991].

2.3.2. Nivel ejecutivo

El nivel ejecutivo es el encargado de seleccionar qué comportamientos reactivos son los más adecuados para cada momento de la ejecución. Son términos empleados igualmente para designar este nivel los de nivel de secuenciación, secuenciador, o nivel de control de la ejecución.

Dentro de las estrategias de activación de comportamientos que se utilizan a este nivel están el análisis exhaustivo, que intenta encontrar la acción adecuada para todas y cada una de las situaciones en las que pueda encontrarse el sistema (sólo es útil en entornos restringidos), y la secuenciación condicional.

La secuenciación condicional trata de reflejar el comportamiento humano, que es capaz de seguir un conjunto de instrucciones en presencia de situaciones no previstas. Para ello, además de los mecanismos habituales de secuenciación como bucles o bifurcaciones condicionales, se incluyen mecanismos de gestión de excepciones y coordinación entre múltiples tareas.

A diferencia del nivel reactivo, en el nivel ejecutivo se hace uso de mecanismos de representación y modelado. Este hecho suele asociarse con la memoria a corto plazo del sistema.

Algunos ejemplos de lenguajes empleados en la construcción de secuenciadores condicionales son RAPs [Firby, 1989], PRS [Ingrand et al., 1992], el Lenguaje de Comportamientos (*Behavior Language*) [Brooks, 1990] o ESL [Gat, 1997*a*].

2.3.3. Nivel deliberativo

El nivel deliberativo es el encargado de establecer los pasos a cubrir para completar la misión del sistema a largo plazo. En este nivel se disponen algoritmos de búsqueda

que pueden poseer constantes de tiempo muy elevadas. Denominaciones alternativas para este nivel son nivel de planificación o nivel de decisión.

Existen dos modos básicos de funcionamiento: el planificador genera planes de forma desacoplada para que sean ejecutados por el nivel de secuenciación, o bien opera de forma síncrona en respuesta a peticiones concretas por parte de dicho nivel.

En el nivel deliberativo se hace un uso intensivo de las técnicas de modelado. El objetivo es aprovechar la información sobre situaciones pasadas para anticipar su evolución. Este nivel suele vincularse con la memoria a largo plazo del sistema.

2.4. La Adaptación Computacional en Sistemas Percepto-Efectores

Un objetivo prioritario de un sistema percepto-efector es alcanzar un comportamiento robusto en el desempeño de las tareas que se le encargan. Se trata de un problema complejo que puede ser abordado en el contexto de diferentes disciplinas, que cubren desde el Control Automático, la Robótica o la Planificación [Jones, 1997]. Una de las principales fuentes de inestabilidad para alcanzar este objetivo de robustez deriva de las relaciones de bajo nivel entre el tiempo de respuesta requerido y la capacidad de cómputo disponible.

Los sistemas percepto-efectores se ven a menudo sometidos a limitaciones de recursos que deben ser gestionados para que las metas propuestas sean alcanzables con las restricciones temporales impuestas [D'Ambrosio, 1989][Horvitz et al., 1989]. Lo ideal es que el sistema degrade su funcionamiento de una manera controlada de forma que, para una cantidad de recursos determinada, el resultado que se obtiene es una configuración que proporciona siempre el "mejor rendimiento posible". Asimismo, es necesario poder recuperar ordenadamente el sistema desde una situación degradada cuando las condiciones así lo permitan. La complejidad del problema es elevada si tenemos en cuenta los siguientes factores:

- Dificultad para estimar los recursos disponibles.
- Complejidad para evaluar la calidad de un resultado.
- Múltiples alternativas de degradación/promoción.

A esto hay que unir la presencia de múltiples tareas en ejecución simultánea con características diferenciadas en cuanto a su naturaleza, prioridad, dependencias o

comportamiento dinámico. Un sistema inteligente que deba reaccionar en entornos con fuertes limitaciones temporales debe cumplir una serie de requerimientos, como los propuestos por Hayes-Roth [Hayes-Roth, 1988]:

- Versatilidad:
  - Conocimiento en múltiples dominios.
  - Razonamiento simultáneo en múltiples direcciones.
  - Razonamiento incremental.
  - Explicación del conocimiento.
- Interacción con un sistema dinámico:
  - Funcionamiento asíncrono.
  - Operación continua.
  - Integración funcional.
- Manejo de la complejidad:
  - Atención selectiva.
  - Rendimiento mínimo garantizable.
  - Razonamiento localizado.
- Rendimiento en tiempo real:
  - Latencias máximas garantizables.
  - Respuesta ante limitaciones temporales.
  - Degradación controlada.
  - Tiempos de respuesta independientes del nivel de aprendizaje.

2.4.1. Niveles en la adaptación

Podemos hablar de adaptación a múltiples niveles. En el nivel más bajo de una arquitectura compleja, las tareas a ejecutar suelen venir predeterminadas desde una etapa de planificación superior. Se conforma así un conjunto estático de tareas cuya única posibilidad de adaptación consiste en la alteración de los requerimientos computacionales demandados de forma interna. En un nivel intermedio, las tareas pueden sustituirse

por otras equivalentes en función de los recursos disponibles, manteniéndose la planificación invariante. En el nivel más alto, sólo la misión del sistema permanece invariante, actuando los mecanismos de planificación como reguladores. En [Hayes-Roth, 1995], estos niveles de adaptación se identifican como adaptación de la estrategia perceptual, del modo de control, de las tareas de razonamiento, de los métodos de razonamiento y de las estrategias de meta-control.

Nuestro interés se centra especialmente en la adaptación computacional de bajo nivel. Los esfuerzos de investigación se concentran aquí en dos aspectos: variación de la carga y planificación de las tareas. Por un lado se buscan esquemas de cómputo para las tareas que permitan variar dinámicamente la carga que suponen para el sistema. Por otro, se intenta aprovechar ese nuevo grado de libertad en los mecanismos de planificación, a fin de permitir un reparto óptimo de los recursos disponibles.

Variación de la carga computacional

La carga de un sistema puede variarse mediante dos vías principales: modificar el número de tareas en ejecución o modificar sus demandas computacionales. La primera de las alternativas requiere normalmente una replanificación del sistema, por lo que se trata de una estrategia de alto nivel, mientras que la segunda puede aplicarse en niveles más bajos.

La modificación de las demandas computacionales de una tarea puede a su vez alcanzarse mediante diferentes estrategias. Algunas alternativas que cabe destacar son las siguientes:

- Algoritmos *anytime*.
- Computaciones imprecisas.
- Diseño en función del tiempo.
- Planificación deliberativa.
- Modificación de la frecuencia de funcionamiento (tareas periódicas).

Los algoritmos *anytime* [Dean y Boddy, 1988] constituyen estrategias de procesamiento iterativas que proporcionan resultados de calidad creciente a medida que aumenta el tiempo disponible. En general, se precisa un tiempo mínimo para que se genere un primer resultado con calidad mínima. A partir de ese instante se van obteniendo refinamientos hasta llegar a la máxima calidad posible.

En la computación imprecisa (*Imprecise Computation*) [Liu et al., 1994] se fraccionan las tareas en una parte obligatoria y una parte opcional. La parte obligatoria siempre se ejecuta, mientras que la parte opcional únicamente se ejecuta si existen recursos computacionales disponibles.

El diseño en función del tiempo (*Design-to-time*) [Garvey y Lesser, 1993] [Wagner et al., 1998], o método de múltiples versiones, se basa en disponer de varias implementaciones alternativas para cada tarea, cada una con diferentes consumos de recursos. En función de los recursos existentes en un momento determinado se van seleccionando para la ejecución implementaciones más o menos exigentes.

En la planificación deliberativa (*Deliberative scheduling*) [Boddy y Dean, 1994] se consideran todas las tareas como algoritmos *anytime.* La ejecución se planifica de manera que se maximice alguna medida de la utilidad (calidad) global del sistema.

Todos estos métodos se basan de una u otra manera en el manejo de perfiles de rendimiento en los que se representa la evolución de la calidad que se alcanza en función de la cantidad de recurso disponible. Otros autores como Musliner et al. [Musliner et al., 1995] trasladan estos resultados al contexto de una arquitectura robótica híbrida intentando eliminar la necesidad de disponer de perfiles de rendimiento. Para ello se combinan soluciones tácticas a nivel reactivo con soluciones estratégicas a nivel deliberativo. Los perfiles de rendimiento son reemplazados por una medida de la calidad global alcanzada.

En tareas periódicas, es posible reducir la carga computacional modificando el periodo de funcionamiento. Si la tarea posee un rango de frecuencias de operación válidas, el desplazamiento hacia frecuencias menores permite relajar las demandas impuestas al sistema, mientras que las frecuencias mayores absorben más recursos.

Planificación de las tareas

Una vez identificados los elementos de ajuste disponibles, el siguiente paso es utilizar esos grados de libertad para alcanzar un comportamiento adecuado en situaciones de carga y recursos computacionales variables.

En los sistemas con restricciones de tiempo real críticas (*hard real-time*), se aplican técnicas de planificación de CPU o *scheduling* que eviten la violación de los límites temporales. Una de las soluciones factibles es la planificación previa a la ejecución basada en el peor caso. Sin embargo, se trata de una alternativa que en la mayoría de las ocasiones resulta excesivamente conservadora, puesto que el peor caso suele estar muy

alejado del comportamiento promedio. Esto deriva en una clara infrautilización de los recursos disponibles en el sistema durante un porcentaje elevado de su tiempo de operación. En algunos casos, pueden incluso fallar las estimaciones para el peor caso, lo que no garantiza una ejecución libre de fallos [Stewart y Khosla, 1997]. Para evitar esto, se precisan mecanismos adaptativos que ajusten el consumo de recursos a los disponibles en cada momento. En esta línea existen múltiples trabajos de planificación dinámica para sistemas de tiempo real [Bondavalli et al., 1993][Ramamritham y Stankovic, 1991].

El objetivo de estas técnicas es normalmente extender los mecanismos de planificación básicos para sistemas de tiempo real duro EDF (*Earliest Deadline First*) de prioridades dinámicas o RM (*Rate Monotonic*) de prioridades fijas, de manera que puedan aprovecharse las características de la carga computacional variable. Algunos ejemplos son los trabajos que se concentran en la fijación de las frecuencias de funcionamiento, como es el caso de [Seto et al., 1996] o [Buttazzo et al., 1998].

Otro enfoque para el problema de la adaptación consiste en hacer énfasis en la integración en sistemas tolerantes a fallos ([González et al., 1997], [Bondavalli et al., 1993]). En estos casos se buscan combinaciones de las técnicas de redundancia para alcanzar un comportamiento adaptativo. Dichas técnicas incluyen TMR (*Triple Modular Redundancy*), PB (*Primary Backup*) y PE (*Primary Exception*). Normalmente se aplican en entornos multiprocesador.

Stewart [Stewart, 1994] propone un mecanismo de planificación híbrido denominado MUF (*Maximum Urgency First*). En este trabajo se combinan las prioridades estáticas del RM con las prioridades dinámicas del EDF.

2.5. Las Arquitecturas Robóticas Híbridas: Estudio de Casos

A continuación, y por su significancia en el modelo propuesto en esta tesis, se van a describir brevemente un conjunto de arquitecturas robóticas híbridas, haciendo especial hincapié en su organización estructural y los mecanismos de adaptación computacional que incorporan.

2.5.1. AuRA

AuRA (Autonomous Robot Architecture) es una propuesta de arquitectura robótica debida a Ronald Arkin. Se trata de una arquitectura híbrida [Arkin y Balch, 1997] de

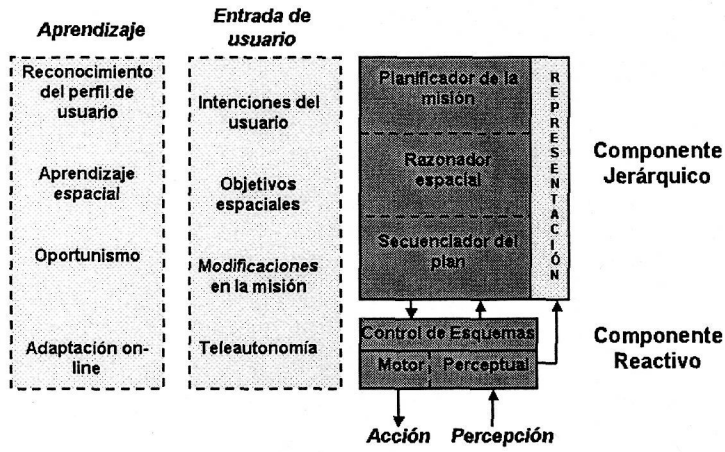

Figura 2.1: Diagrama de bloques de la arquitectura AuRA.

dos niveles especialmente orientada hacia tareas de navegación. AuRA aprovecha ideas derivadas del estudio de sistemas biológicos y de la neurofisiología para trasladarlas a los sistemas robóticos. Se ha empleado esta arquitectura en la construcción de diferentes robots para diversas aplicaciones de navegación, exploración y manipulación.

Estructura

AuRA combina un componente jerárquico de deliberación en el nivel superior con un componente reactivo en el nivel inferior. El componente jerárquico está integrado por un planificador de misión (*mission planner*), un módulo de razonamiento espacial (*spatial reasoner*) y un secuenciador (*plan sequencer*). Enlazado con este nivel se sitúa como componente reactivo un controlador de esquemas motores (*schema controller*). La figura 2.1 muestra un diagrama de bloques con los elementos principales de la arquitectura.

Dentro de la parte deliberativa, el planificador de misión establece las metas de alto nivel para el sistema y las restricciones bajo las que debe operar. El módulo de razonamiento espacial utiliza información cartográfica para construir la secuencia de segmentos de trayectoria que el robot debe seguir para completar su misión. Por último el secuenciador traslada cada trozo de trayectoria en un conjunto de comportamientos motores para comandar su ejecución en el nivel inferior.

En el nivel reactivo, el controlador de esquemas monitoriza y controla en tiempo de ejecución la evolución de los comportamientos de bajo nivel. Cada comportamiento o esquema motor [Arkin, 1989] produce como salida de control un vector de respuesta que

se combina con las respuestas de todos los demás esquemas para producir el comando que llegará finalmente al robot físico. Todos los comportamientos operan de manera asíncrona.

Una vez se comandan las acciones correspondientes al nivel reactivo, el nivel deliberativo queda a la espera de recibir la notificación de tarea finalizada correctamente o bien de algún error. Los errores se intentan resolver ordenadamente, comenzando por el secuenciador y terminando en el planificador de misión si todos los demás han fracasado en la resolución del problema.

La arquitectura es altamente modular, lo que ha permitido ensayar diferentes implementaciones para cada nivel, reemplazando las existentes.

Adaptación

Se han probado en AuRA diferentes mecanismos de adaptación y aprendizaje. Dentro de ellos cabe destacar los siguientes:

- Control homeostático [Arkin, 1992].
- Adaptación dinámica para los comportamientos empleando métodos basados en reglas [Clark et al., 1992].
- Razonamiento basado en casos para gobernar la conmutación de comportamientos en función de las características del entorno [Ram et al., 1992].
- Algoritmos genéticos para el ajuste de los parámetros asociados a los lazos de control [Ram et al., 1994].

El control homeostático merece especial interés en el contexto de este trabajo. El objetivo es mantener bajo control determinadas variables relevantes para garantizar la integridad física del sistema, a modo de "constantes vitales". Se trata de un control autónomo de bajo nivel que opera de forma continua e independientemente de la presencia de bucles de control de más alto nivel. Algunos ejemplos serían la vigilancia del nivel de baterías o el control de la temperatura. Se ha probado fundamentalmente a nivel de simulación.

2.5.2. CIRCA

La arquitectura CIRCA (*Cooperative Intelligent Real-Time Control Architecture*), ideada por David Musliner, Edmun Durfee y Kang Shin, constituye un intento de

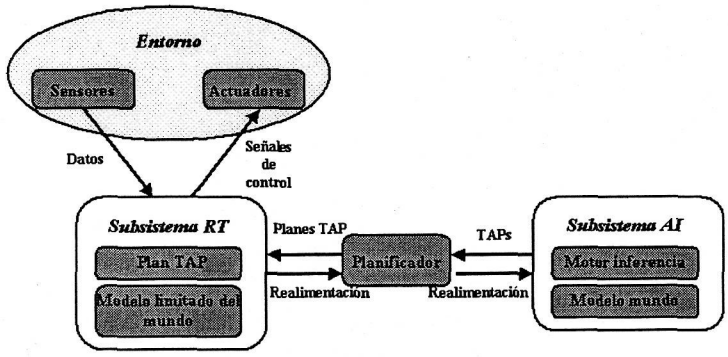

Figura 2.2: Subsistemas de la arquitectura CIRCA.

combinar técnicas de Inteligencia Artificial con restricciones de tiempo real en el mismo sistema.

Estructura

CIRCA [Musliner et al., 1993] se compone de dos subsistemas, el RTS (*Real Time Subsystem*) y el AIS (*Artificial Intelligence Subsystem*), para abordar separadamente los objetivos de control y de tarea respectivamente. La figura 2.2 muestra estos subsistemas y sus interconexiones.

El AIS utiliza métodos de Inteligencia Artificial para descomponer los objetivos a nivel de tarea en objetivos de control. Posteriormente se hace uso de un planificador para intentar garantizar los tiempos de respuesta de esos nuevos objetivos. Se utiliza un grafo dirigido para modelar el comportamiento del entorno y las acciones que debe tomar el RTS a fin de evitar fallos. El modelado se realiza siempre para el peor caso.

El RTS ejecuta una secuencia de TAPs (*Test-Action pairs*) cíclicamente. Dichas secuencias son propuestas por el AIS de modo que cumplan los objetivos de control en dirección a los objetivos a nivel de tarea. El planificador comprueba su viabilidad en función de los recursos disponibles, dando lugar a una planificación de TAPs con tiempos de respuesta garantizados.

Un TAP está formado por un conjunto de precondiciones, las acciones a tomar si se cumplen las precondiciones, datos sobre los sensores y actuadores requeridos, y estimaciones para el peor caso de los tiempos requeridos para comprobar las precondiciones

y ejecutar las acciones. Adicionalmente, las instancias específicas de cada TAP pueden incorporar parámetros extra como son la frecuencia de operación o el tiempo máximo de respuesta admisible.

En un trabajo posterior, CIRCA ha evolucionado para dar lugar a SA-CIRCA (*Self-Adaptive CIRCA*) [Musliner et al., 1999]. En esta propuesta, el AIS pasa a estar constituido por dos módulos, el AMS o planificador de misión adaptativo (*AMP-Adaptive Mission Planner*) y el CSM o módulo de síntesis de controladores (*Controller Synthesis Module*). El AMS razona acerca de las metas a largo plazo en función de las restricciones impuestas para decidir cuáles deben ser los problemas a resolver a corto plazo. Las submetas son enviadas al CSM para que éste sintetice planes de control que cumplan con las especificaciones. Mientras el AMS y el CSM siguen procesando futuras situaciones por adelantado, el RTS recibe los planes de control generados y los ejecuta. El RTS mantiene un mecanismo de almacenamiento de múltiples planes que pueden ser activados como planes alternativos mediante una rápida conmutación de contexto.

Adaptación

Los esquemas de adaptación de CIRCA son limitados al ser un sistema orientado a garantizar restricciones de tiempo real. Aún así, incorpora un mecanismo para ajustar la demanda computacional a los recursos existentes basada en una lista de TAPs no garantizados. Esta lista se compone de TAPs que implementan tareas de baja prioridad. Cuando por alguna causa se detectan recursos disponibles, por ejemplo cuando una TAP perteneciente a la planificación garantizada no se ejecuta, se intenta seleccionar una TAP de la lista no garantizada para cubrir ese hueco.

En [Musliner, 2001] se describen las capacidades de adaptación de SA-CIRCA. Existen mecanismos para alterar la planificación en tiempo de ejecución o bien para seleccionar dinámicamente planes previamente diseñados. Cuando una situación no prevista se presenta se intenta en primer lugar conmutar a un plan ya definido que la trate. Si esto no es posible, se solicita la generación de un nuevo plan.

2.5.3. Arquitectura 3T

La arquitectura 3T ("3 Tiers") es el resultado de la colaboración entre diferentes investigadores: Peter Bonasso, James Firby, Erann Gat o David Kortenkamp, entre otros. Algunos ejemplos de otras propuestas de arquitecturas que han contribuido o están estrechamente relacionados con este proyecto son AAA [Firby et al., 1995*a*] o

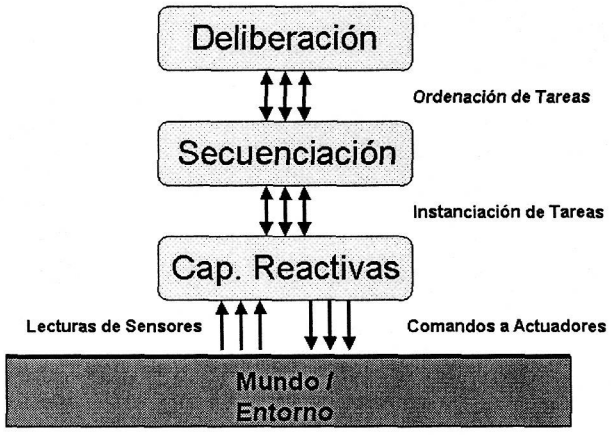

Figura 2.3: Niveles de la arquitectura 3T.

ATLANTIS [Gat, 1992].

El objetivo principal que se persigue en 3T es alcanzar un comportamiento robusto en la resolución de tareas mediante la combinación de reactividad y deliberación. En el contexto de este proyecto se han desarrollado, además, un conjunto de herramientas software para ayudar al programador en el diseño de aplicaciones robóticas. La arquitectura 3T ha sido utilizada en una gran variedad de entornos y aplicaciones [Bonasso et al., 1997], como son búsqueda y reconocimiento de personas, recolección de basuras, navegación en entornos de oficina y simulación de manipuladores.

Estructura

La arquitectura 3T comprende tres niveles:

- Nivel de capacidades o habilidades reactivas.
- Secuenciador.
- Planificador.

La figura 2.3 muestra la organización por niveles de esta arquitectura.

El planificador sintetiza todos los objetivos que se impongan al sistema en un listado de tareas a realizar. Estas tareas, a su vez, se descomponen en uno o más conjuntos de acciones o RAPs (*Reactive Action Packages*) [Firby, 1989], que van siendo

seleccionadas por el planificador para su ejecución. En el nivel de secuenciación, se reciben los RAPs activados y se controla su ejecución. Esto supone obtener los conjuntos de habilidades requeridas y ordenar su activación al nivel reactivo, junto con un conjunto de monitores de eventos. Los monitores permiten detectar el disparo de diferentes eventos, que son notificados al secuenciador. El secuenciador va reemplazando las tareas por otras nuevas, si las hay, a medida que se detectan eventos, se registra una violación temporal o se recibe una nueva planificación desde el nivel superior.

El nivel de las capacidades está formado por un conjunto de acciones de control dependientes del entorno (*situated skills*) que han sido obtenidas de manera sistemática para evitar resultados dependientes de un robot o contexto de aplicación concretos. Se llega así a una representación uniforme para las capacidades que facilita su manipulación. Esta representación incluye los siguientes elementos:

- Especificación de las entradas y salidas.
- Algoritmo de procesamiento.
- Rutina de inicialización.
- Función de activación.
- Función de desactivación.

Las capacidades se controlan por medio de un gestor (*skill manager*) que establece una interfaz uniforme con el secuenciador. A través de esa interfaz circulan comandos dirigidos a las capacidades y eventos que deben ser notificados al secuenciador. El objetivo es liberar al programador de los detalles de coordinación entre las capacidades de bajo nivel para concentrarse en la tarea.

La secuenciación corre a cargo del intérprete de tareas (*RAPs interpreter*). Este intérprete dispone de una librería de RAPs, cada una de las cuales es adecuada para una situación determinada y, por lo tanto, se traducen en un conjunto diferente de habilidades. Una clase especial de capacidades son los eventos, que se encargan de notificar al secuenciador circunstancias especiales relevantes para la evaluación el progreso de la actividad (fin de tarea, cambio de entorno, etc).

Por encima de la combinación secuenciación/reacción, se necesita un elemento que añada perspectiva global al sistema. Esa labor está desempeñada por el planificador, aunque se sirve del grado de abstracción que aportan los niveles inferiores para simplificar su tarea, reduciendo la dimensión del espacio del problema a analizar. Una premisa

importante para lograr los objetivos propuestos en 3T es que los tres niveles definidos operen de manera concurrente y asíncrona.

Una cuestión determinante es cómo decidir a qué nivel debe ubicarse una actividad concreta dentro del sistema. Para ello se propone un análisis basado en cuatro características:

Periodo de repetición: Los periodos típicos utilizados en el nivel reactivo son del orden de milisegundos, los del nivel de secuenciación del orden de décimas de segundo y los del planificador varían de segundos a decenas de segundos.

Ancho de banda: Las habilidades de bajo nivel pueden llegar a intercambiar elevados volúmenes de información, mientras que la interfaz entre niveles debe demandar un ancho de banda reducido.

Funcionalidad: Si una actividad de un cierto nivel incorpora mecanismos ya recogidos en la arquitectura es mejor dividirla y que sea la propia arquitectura la que actúe (habilidades que realizan selección o RAPs que gestionan recursos).

Grado de modificabilidad: Las capacidades suelen ser compiladas y no pueden modificarse en tiempo de ejecución, cosa que sí ocurre en los niveles de secuenciación y planificación que están basados en intérpretes.

Adaptación

La ejecución adaptativa se basa en disponer de múltiples métodos de resolución para satisfacer una determinada tarea. Cada uno de esos métodos está asociado a unas condiciones de aplicación determinadas. La tarea posee un test de éxito para comprobar cuándo la tarea ha finalizado correctamente.

Arquitecturas relacionadas

En [Firby et al., 1995*a*] se aplicaban soluciones similares a las requeridas en la implementación de la capa de bajo nivel de un sistema de visión, para constituir la arquitectura AAA (*Animate Agent Architecture*). Se encapsula el procesamiento visual en módulos denominados rutinas visuales, formadas a su vez por combinaciones de primitivas u operadores visuales. En tiempo de ejecución, las rutinas visuales se emparejan con rutinas de acción dando lugar a bucles cerrados de operación en tiempo real. La figura 2.4 ilustra la organización por niveles de esta arquitectura.

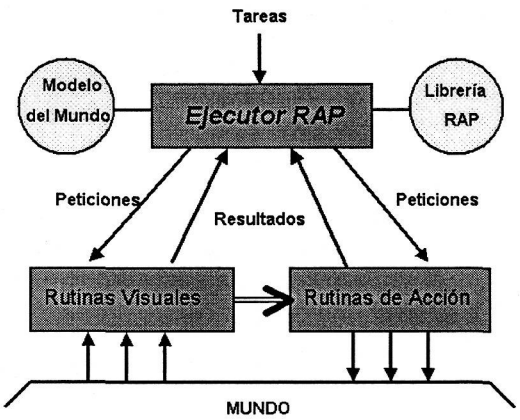

Figura 2.4: Niveles en la arquitectura AAA.

El funcionamiento del ejecutor de tareas en AAA comprende las siguientes fases:

- Selección de la tarea.
- Si es tarea primitiva:
  - Ejecutarla.
- Si no es tarea primitiva:
  - Extracción del RAP correspondiente de la librería.
  - Selección de métodos mediante test de aplicabilidad.
  - Expansión de subtareas.
- Test de finalización.

La arquitectura AAA se ha probado en robots reales en aplicaciones de navegación y recogida de objetos [Firby et al., 1995*b*].

La arquitectura ATLANTIS [Gat, 1992], comparte muchas características comunes con 3T. Las diferencias se centran en aspectos de control como son el hecho de que el secuenciador se encarga de controlar tanto aspectos de bajo nivel como otros propios del nivel deliberativo (ver figura 2.5). ATLANTIS se ha utilizado en la implementación de diferentes sistemas robóticos para aplicaciones en entornos de interior y en exteriores [Miller et al., 1992]. La programación de las aplicaciones en ATLANTIS se realiza mediante el lenguaje de propósito específico ALFA [Gat, 1991].

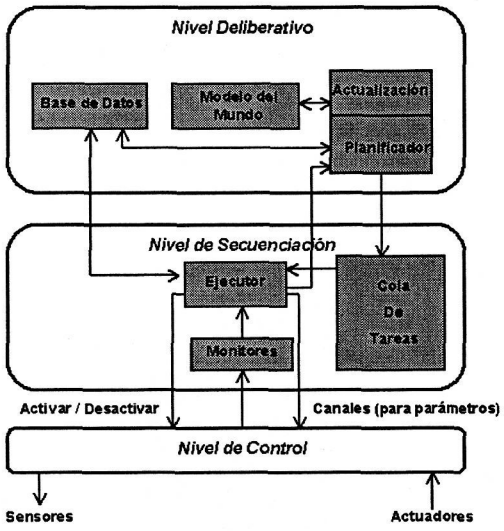

Figura 2.5: Arquitectura Atlantis.

2.5.4. Arquitectura del LAAS

Esta propuesta surge en el seno del LAAS del CNRS. Sus autores son, entre otros, los investigadores Sara Fleury, Rachid Alami, Raja Chatila y Felix Ingrand. El objetivo básico que se plantea es el de la integración para un sistema robótico de módulos de tiempo real en una arquitectura híbrida. Esto se traduce en una serie de exigencias al robot desde el punto de vista funcional, y que pretenden la constitución de un sistema distribuido en tiempo real predecible y extensible.

Esta arquitectura, también referenciada como proyecto GenOM (Generation of Modules), ha sido ensayada tanto a nivel de simulación como sobre robots reales, principalmente en tareas de coordinación [Alami et al., 1995].

Estructura

La arquitectura propuesta [Alami et al., 1998] se organiza en tres niveles:

- Nivel funcional.
- Nivel de ejecución.
- Nivel de decisión.

La figura 2.6 ilustra la estructura de esta propuesta para un ejemplo de bajo nivel que combina seguimiento con evitación de obstáculos.

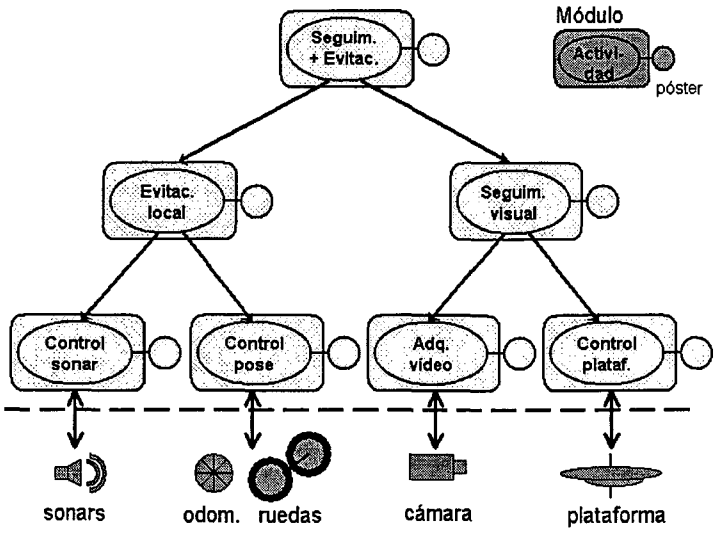

Figura 2.6: Ejemplo de aplicación basada en la arquitectura del LAAS.

El nivel funcional contiene todas las capacidades básicas de percepción y acción. Estas funciones se encapsulan en módulos controlables [Fleury et al., 1997] que se comunican entre ellos y con el nivel físico a través de una capa de abstracción del hardware robótico, el nivel lógico de robot ("logical robot level"). El propósito de esta capa intermedia es independizar el desarrollo de las aplicaciones del robot físico sobre el que vayan a ejecutarse.

El nivel ejecutivo actúa de interfaz entre el nivel funcional y el nivel de decisión. Se trata de un nivel sin capacidad de planificación que recibe las secuencias de acciones a ejecutar desde el nivel de decisión y las traduce en órdenes de control dinámicas sobre el nivel funcional. El nivel ejecutivo se estructura en tres componentes principales:

Monitor de peticiones: Recibe las peticiones desde el nivel de decisión y utiliza la base de datos de peticiones para mapearlas en un conjunto de módulos a activar.

Monitor de respuestas: Vigila la evolución de la ejecución en el nivel funcional para notificar los eventos detectados al nivel de decisión.

Base de datos del estado de la ejecución: Contiene reglas para resolver conflictos en función de la información suministrada por los monitores.

El código empleado en el nivel ejecutivo es crítico para el funcionamiento del sistema en su conjunto, por lo que debe ser sometido a verificaciones formales de tipo lógico y temporal que garanticen su correcto funcionamiento.

Por encima del nivel funcional, el sistema posee un nivel de decisión al que corresponde interpretar la misión encargada al robot descomponiéndola en peticiones al nivel inferior. Este nivel debe ser reactivo e integrar información relativa al conflicto entre recursos y aplicaciones. Para garantizar la reactividad, el nivel de decisión está formado por dos entidades: un planificador y un supervisor. El planificador genera la secuencia de acciones necesaria para alcanzar una determinada meta, y es utilizado como un recurso por el supervisor, que es el encargado de interactuar con el nivel inferior, controlar la ejecución del plan y reaccionar ante los eventos que se produzcan. El supervisor actúa además como interfaz con el usuario.

El supervisor hace uso dos elementos adicionales al plan durante su ejecución: las modalidades de ejecución, para limitar el ámbito de la planificación, y una base de datos de procedimientos dirigidos por el contexto ("situation-driven procedures"). Los algoritmos de deliberación que se ejecuten en el supervisor deben poseer tiempos de ejecución limitados, para evitar que el bucle de control que representa se ralentice en exceso.

El nivel de decisión puede organizarse internamente en múltiples capas dependiendo de la aplicación. Todas las capas, sin embargo, siguen la misma estructura interna planificador/supervisor.

Los periodos típicos de operación de los tres niveles son del orden de centésimas de segundo para el funcional, décimas de segundo para el ejecutivo y segundos para el de decisión. Se ofrecen una serie de herramientas para verificar el funcionamiento del sistema. Entre otras, un programa de test interactivo para interrumpir servicios y consultar resultados, y un estimador de tiempos de cómputo y generador de cronogramas.

Los módulos

Un módulo se define como una entidad software que ofrece servicios relativos a un determinado recurso. Los servicios se atienden en base a un protocolo cliente/servidor, devolviéndose el resultado de la ejecución y una medida de la calidad del mismo. Los módulos reciben peticiones de ejecución para iniciar un determinado procesamiento y de control para modificar su comportamiento. Las relaciones entre los módulos se establecen dinámicamente.

La figura 2.7 muestra la estructura interna de un módulo genérico de esta arquitectura.

Los datos entre módulos se comunican mediante "pósters", que son áreas de me-

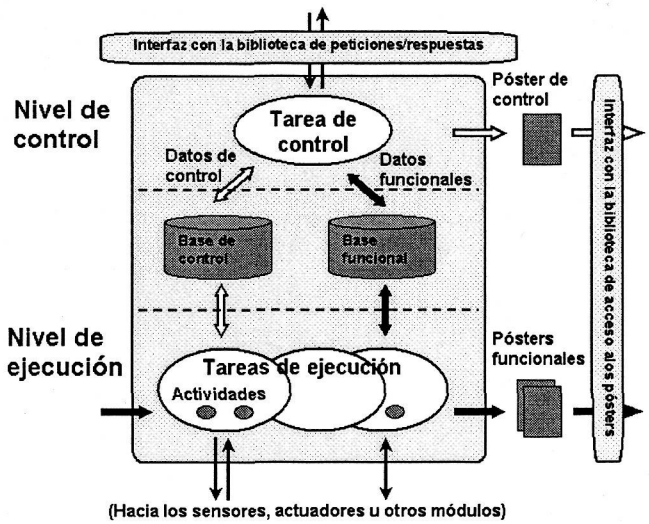

Figura 2.7: Estructura interna de un módulo genérico en la propuesta del LAAS.

moria compartida modificables sólo por el módulo propietario y accesibles a todos los demás. Existen, además de estos pósters funcionales, pósters de control que reflejan el estado de cada módulo.

El estado de ejecución del sistema se representa mediante un árbol de actividades, en el que se refleja una descomposición jerárquica formada a partir del lanzamiento de actividades hijas desde una determinada actividad madre. Las actividades deben integrarse de modo que sean interrumpibles, terminen adecuadamente y sean capaces de detectar fallos. Para esto se define la estructura de módulo genérico, que comprende los siguientes elementos:

Nivel de control: Encargado de la gestión del módulo, este nivel recibe las peticiones de los clientes, las verifica, resuelve los conflictos que se puedan presentar, inicia y finaliza actividades, envía respuestas de vuelta a los clientes. También se encarga de mantener una representación interna del estado del módulo que es exportada mediante los pósters de control.

Nivel ejecutivo: Ejecuta las actividades solicitadas desde el nivel de control. Este nivel está integrado por una o varias tareas de ejecución (*execution tasks*), que pueden ser periódicas o aperiódicas. Estas tareas constituyen el contexto de ejecución de una o varias actividades.

Comunicación internivel: El intercambio de información entre niveles se efectúa mediante dos bases de datos: una base de datos funcional y otra de control.

Una petición de ejecución se traduce en una actividad que evoluciona según un grafo de control cuyas transiciones son provocadas bien desde el nivel de control bien desde el nivel de ejecución. Los estados posibles para una actividad son cinco: "ETHER", "INIT", "EXEC", "ZOMBIE" y "INTER". Las transiciones admitidas entre esos estados son "init", "exec", "failed", "abort" y "ended". El estado "EXEC" de ejecución se descompone a su vez en pasos de ejecución (*codels*), que son atómicos para el módulo. En general, suele existir un paso de inicialización ("start"), otro de progreso de la ejecución, normalmente iterativo ("exec"), otro de finalización ("end"), y por último un paso de fallo ("fail") para situaciones de error.

Los módulos son generados automáticamente a partir de una descripción formal de los mismos. Esta descripción incluye aspectos como son la identificación del módulo, los tipos de datos usados, las solicitudes admitidas y elementos asociados (tipo de petición, parámetros de entrada y salida, listado de códigos del informe de ejecución, detalle de los pasos de ejecución), los pósters, los parámetros de ejecución (periodo, prioridad), etc.

Adaptación

Los mecanismos de adaptación se basan en la estructura interna de los módulos. Éstos pueden incorporar procedimientos de tratamiento de errores en tiempo de ejecución que incluyen estrategias de recuperación o notificación de fallos a los clientes.

2.5.5. Arquitectura DAMN

DAMN (*Distributed Architecture for Mobile Navigation*)es una arquitectura distribuida debida a Julio Rosenblatt. Se basa en la ejecución asíncrona de múltiples comportamientos que acceden al control del robot de forma compartida a través de un mecanismo de votación. El objetivo es integrar mecanismos reactivos con componentes deliberativos sin por ello tener que imponer una estructura jerárquica.

Esta propuesta ha sido utilizada en diferentes aplicaciones de robots operativos que incluyen seguimiento de caminos, navegación en exteriores y teleoperación, combinados con evitación de obstáculos [Langer et al., 1994].

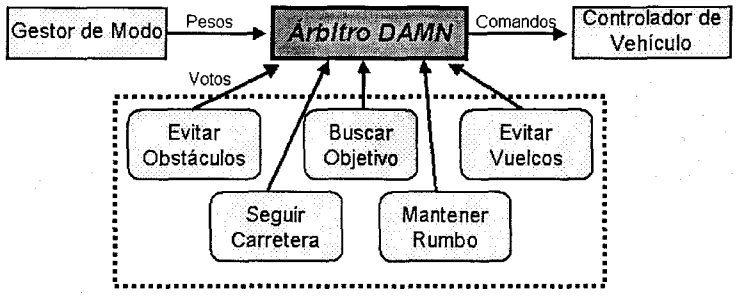

Figura 2.8: Componentes de la arquitectura DAMN.

Estructura

La arquitectura DAMN [Rosenblatt, 1995] está integrada por los siguientes elementos:

- Sistema de control de la votación (*DAMN Arbiter*).
- Comportamientos.
- Controlador de modo de funcionamiento.
- Controlador del vehículo.

La figura 2.8 muestra las interconexiones entre los diferentes componentes de la arquitectura.

Los diferentes comportamientos, con independencia de su nivel de competencia, realizan votaciones positivas y negativas en el espacio de comandos. Aquí se incluyen comandos de giro, velocidad, área de visión, etc. El gestor de las votaciones se encarga de fundir los resultados para elegir la opción más votada. Esta selección es el producto de diferentes procesos que incluyen la combinación de los resultados con los pesos asignados por el controlador de modo, suavizado e interpolación. El funcionamiento es similar a la arquitectura de supresión, si bien aquí pueden emplearse representaciones internas del mundo.

En esta arquitectura los comportamientos se agrupan en tres niveles de competencia:

- Seguridad.
  - Dinámica del vehículo: Límites de giro y velocidad.
  - Evitación de obstáculos: Evaluación de colisión en posibles trayectorias.
  - Comportamientos auxiliares: Movimientos por defecto e inercias.
- Acción.
  - Seguimiento de carretera.
  - Campo a través.
  - Teleoperación.
- Objetivos.
  - Subobjetivos.
  - Campos de gradiente.
  - Planificador de trayectorias.

El diseño de DAMN permite combinar comportamientos con frecuencias de funcionamiento variadas, como corresponde al caso de los comportamientos reactivos frente a los deliberativos. El resultado de la fusión de los comandos generados determina el comportamiento del sistema, de forma que las acciones producidas por módulos con un peso mayor son las que ejercen más influencia. Aún así, pueden emplearse mecanismos de operación de más alto nivel que controlan la asignación de pesos a los comportamientos, dando lugar a un nivel de metacontrol.

Adaptación

La ejecución adaptativa en DAMN se basa en la modificación dinámica de los pesos asignados a cada comportamiento. Es también en este punto donde es posible introducir aprendizaje en el sistema, de manera que la configuración de pesos adecuada se registre para su posterior utilización en situaciones similares.

2.5.6. Sistema PRS

PRS (*Procedural Reasoning System*) es una arquitectura genérica para representación y deliberación [Ingrand et al., 1992] sobre acciones y procedimientos en un entorno dinámico [Ingrand y Coutance, 1993]. Está basada en el concepto de agente reactivo o arquitectura de agentes BDI (*Belief-Desire-Intention*).

PRS se constituye como un sistema de planificación híbrido que intenta combinar deliberación con reactividad mediante el uso de planes de meta-control. Se ha utilizado en diferentes aplicaciones con demandas de tiempo real como son la monitorización de fallos mecánicos en cohetes y aviones [Georgeff y Ingrand, 1989], control de robots móviles [Ingrand et al., 1995], y supervisión y control en redes de telecomunicación.

Estructura

Un módulo PRS está integrado por los siguientes componentes:

- Base de hipótesis.
- Un conjunto de metas.
- Biblioteca de planes.
- Estructura de intenciones.

A partir de las metas y las hipótesis, un intérprete se encarga de seleccionar los planes adecuados para formar parte de la estructura de intenciones a ejecutar por el sistema. Los planes son denominados en PRS áreas de conocimiento o KA (*Knowledge Areas*), y contienen secuencias de acciones y tests necesarios para alcanzar una meta determinada. Cada KA consta de un cuerpo principal y una condición de activación, la cual a su vez está constituida por un contexto de aplicación y un evento.

El ciclo básico del intérprete comienza con la detección de cambios en las metas o hipótesis del sistema. Estos cambios dan lugar a la activación de diferentes KA's, de las cuales se seleccionan algunas para su ejecución. El ciclo termina con la ejecución de un paso de alguna de las KA's en la estructura de intenciones. La figura 2.9 ilustra este ciclo de operación.

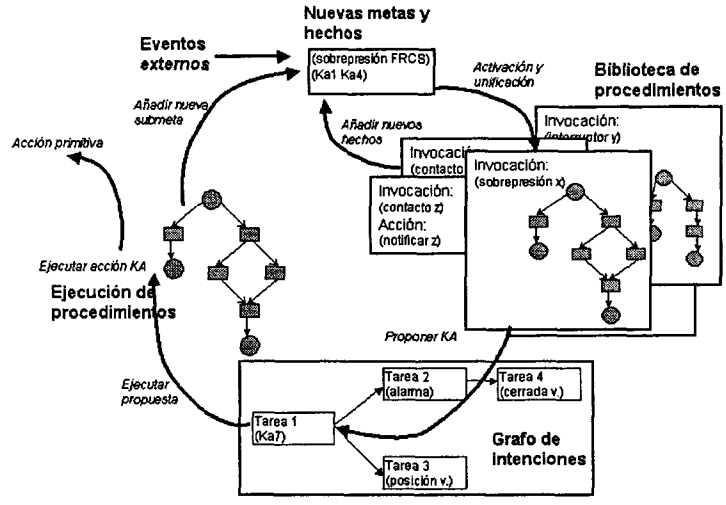

Figura 2.9: Ejemplo del ciclo básico del intérprete PRS.

2.5.7. Arquitectura Saphira

Se plantean como objetivos fundamentales de un agente móvil autónomo la capacidad para atender y seguir (a una persona), la asimilación de información (aprender un mapa del entorno) y la ejecución de tareas (navegar con robustez) [Konolige et al., 1997]. Para cumplir con estos objetivos, una arquitectura debe incorporar tres características básicas: la coordinación, la coherencia y la comunicación. Desde estos presupuestos se ha concebido la arquitectura Saphira, utilizada en la implementación de aplicaciones de seguimiento, navegación e interacción en robots móviles [Guzzoni et al., 1997].

Estructura

Se propone una arquitectura por niveles construida en torno a un mecanismo de representación interna, el espacio perceptual local o LPS (*Local Perceptual Space*). Existe una parte perceptora encargada de trasladar los datos de los sensores al LPS y extraer información de los mismos, y una parte efectora sobre la que se ejecutan los diferentes comportamientos. Se dispone de comportamientos reactivos de bajo nivel, comportamientos dirigidos por metas de nivel intermedio y comportamientos de tareas de alto nivel. La figura 2.9 presenta los diferentes elementos que integran esta propuesta.

La arquitectura está concebida para establecer una relación cliente/servidor con un servidor robótico (*robot server*). Con esto se mejora la transportabilidad, aislando al sistema de las dependencias del hardware.

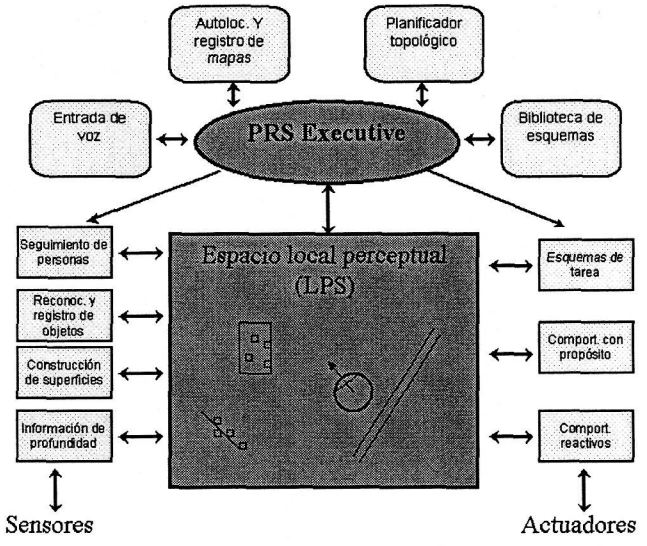

Figura 2.10: Elementos de la arquitectura Saphira.

El control de Saphira se basa en comportamientos. Los comportamientos de bajo nivel (reactivos) se definen y se coordinan empleando lógica difusa [Saffiotti et al., 1997]. En su definición se emplean una serie de reglas difusas y una función de actualización de las variables que intervienen en las mismas.

La acción se selecciona por cada canal de control (variable actuadora) promediando los diferentes valores de acción obtenidos desde las reglas que concluyen sobre dicho canal. La función de ponderación tiene en cuenta para cada comportamiento un valor de prioridad fijo y un contexto de aplicación variable. Este mecanismo de coordinación dependiente del contexto se emplea tanto para los comportamientos reactivos como para los orientados a metas.

La coherencia del sistema se basa en mantener actualizados unos descriptores de objetos del entorno, denominados artificios (*artifacts*), sobre el LPS. Para ello se generan características e hipótesis de objetos, realizándose tanto procesamiento bottom-up para simbolización (características->hipótesis->artificios) como procesamiento top-down de verificación (artificios->hipótesis). Un ejemplo es la construcción de un mapa del entorno extrayendo características lineales a partir de lecturas de sensores ultrasónicos, las cuales son combinadas con información de profundidad para producir hipótesis de objetos tipo pasillo, pared o puerta. Los objetos son comparados con la lista de artificios para su actualización, en caso de coincidencia, o extensión, cuando se detecta un nuevo objeto. Este proceso se denomina anclaje (*anchoring*).

La selección y coordinación de comportamientos la realiza el controlador PRS-

Lite (*Procedural Reasoning System*). Algunas características de este controlador son la capacidad de integración de actividades dirigidas por objetivos y dirigidas por eventos, reactividad o descomposición jerárquica de tareas. Adicionalmente este módulo incorpora capacidades como la gestión de procesos continuos en interacción, un amplio conjunto de mecanismos de control declarativos o el uso de niveles de satisfacción como respuesta tras la realización de una tarea en vez del binomio éxito/fracaso.

PRS-Lite se basa en la utilización de esquemas de actividad (*activity schema*), que se definen como conjuntos ordenados de metas (*goal set*), integrados a su vez por metas simples. Una meta puede pertenecer a dos categorías básicas: acción o secuenciación. Las acciones son del tipo test, asignación, ejecución, espera por condición o expansión/contracción. Este último tipo permite la descomposición jerárquica, dando lugar a estructuras en árbol. Las metas de secuenciación incluyen saltos, condicionales y paralelización.

Se denominan intenciones del sistema al conjunto de esquemas de actividad lanzados. Dentro de ellos, los nodos hojas constituyen los conjuntos de metas actuales, y son los elementos considerados para su ejecución en cada ciclo del sistema.

2.5.8. Arquitectura S*

Como continuación de sus críticas a los sistemas basados en comportamientos [Tsotsos, 1995], Tsotsos propone la arquitectura S*. Esta arquitectura [Tsotsos, 1997] se basa en la definición del concepto generalizado de mundo externo, para hacer referencia tanto al mundo físico como a las diferentes representaciones internas obtenidas a partir del mismo. Cada comportamiento, tanto interno como externo, del sistema se define en base a un modelo SMPA (*Sense Model Plan Act*) ampliado, constituyendo un ciclo SMPA-W.

Estructura

Los elementos que integran un ciclo SMPA-W (ver figura 2.11) son los siguientes:

World: Este nodo contiene dos representaciones: una ventana de eventos y una ventana de acción. Un conjunto de demonios permanecen activos a la espera de detectar cambios en la ventana de eventos y actúan como disparadores de los comportamientos.

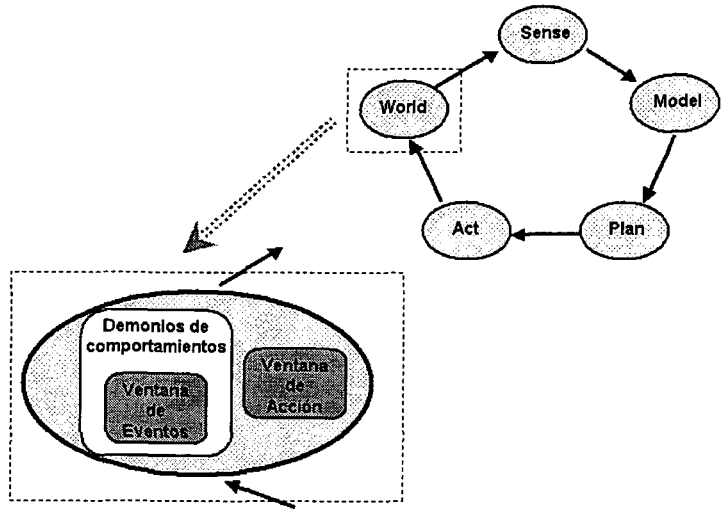

Figura 2.11: Ciclo SMPA-W de la arquitectura S*.

Sense: Elemento que extrae información de la ventana de eventos para su transformación.

Model: Asociación de modelos a subconjuntos de la información extraída.

Plan: Asociación de modelos con comportamientos.

Act: Manipulación generada por los comportamientos sobre la ventana de acción.

A su vez, cada nodo SMPA posee una estructura interna formada por una ventana de eventos y otra de acción conectadas por medio de un elemento de cómputo que asocia los eventos con las acciones. Esta estructura puede expandirse por medio de ciclos auxiliares, lo que permite una descomposición recursiva. Para ello se conecta una salida desde la ventana de eventos hacia la entrada del nuevo ciclo y se recogen sus resultados en una ventana de acción auxiliar. Un módulo de resolución de conflictos debe entonces decidir qué acción (ventana de acción o ventana de acción auxiliar) se traslada a la salida del nodo. Estos ciclos de expansión toman la forma de implementaciones alternativas para obtener un mismo resultado con mayor precisión y, generalmente, más coste.

Una aplicación robótica precisa de una serie de representaciones: R0 de estado, R1 de comandos de actuador, R2 de especificación de la tarea o misión, R3 del entorno, R4 de características extraídas del entorno, R5 de datos de sensores, ER de registro de excepciones, EW de ventanas de eventos y PP de parámetros internos del sistema. Abriendo múltiples ventanas de eventos y acción sobre estas representaciones es posible

implementar políticas tales como mecanismos de atención, ajuste dinámico de parámetros o procesamiento dirigido por objetivos.

La misión del sistema se define mediante un lenguaje que permite especificar comportamientos para su ejecución secuencial o en paralelo, agruparlos en fases, o definir restricciones temporales.

Adaptación

Un aspecto importante es la posibilidad de evaluar si una red de comportamientos puede satisfacer las restricciones de tiempo de ejecución impuestas a la misión. Para ello se define un procedimiento de verificación que parte de la asignación a cada comportamiento de una serie de costes: tiempo de ejecución en función del tamaño de la entrada, tiempo de escritura en la ventana de acción, coste incremental de los comportamientos extendidos y ventana de eventos mínima requerida. Este esquema constituye en realidad un test de aceptación para el peor caso, pero no se exponen mecanismos de adaptación en caso de que el test falle.

2.5.9. Arquitectura multinivel y control experto

Se trata de una arquitectura *híbrida* en tres niveles que pretende dar soporte a problemas de distinta naturaleza, desde misiones de alto nivel a comportamiento reactivos. Esta propuesta es debida a Christensen y Pirjanian [Christensen y Pirjanian, 1997], y se ha ensayado sobre plataformas móviles en aplicaciones de navegación con evitación de obstáculos.

Estructura

En el primer nivel se ubican los comportamientos reactivos. Se trata de conductas básicas de bajo nivel que operan directamente sobre datos de sensores y proporcionan respuestas en tiempo real. Estos comportamientos son autónomos y tienen prioridad sobre los niveles superiores, que simplemente pueden activarlos o desactivarlos.

En el nivel intermedio se sitúan los comportamientos de tarea, encargados de llevar a cabo acciones de navegación y detección y seguimiento de objetos.

El nivel alto se encarga de la planificación de las misiones, descomponiendo el objetivo indicado por el usuario en una serie de tareas a ejecutar. Cuando alguna de las tareas fracasa, se busca una planificación alternativa.

Cada comportamiento se modela como una máquina de estados finita, y la combinación de los mismos como un sistema de eventos discretos (DES [Cassandras, 1993]) aplicados a robótica [Kosecka y Bajcsy, 1993] [Kosecka et al., 1995]. Esto permite la construcción de comportamientos de forma modular y jerárquica, así como predecir la controlabilidad de los mismos.

Para la descripción de la combinación de comportamientos se emplea el modelo de autómata de procesos, siguiendo la propuesta de los esquemas robóticos de Lyons (ver sección 2.6.3). Pueden definirse relaciones de secuencialidad, concurrencia, condición, inhibición o recursión.

El planificador se constituye como un sistema basado en reglas que realiza búsquedas ponderadas en estructuras de grafo con atributos. Estos atributos representan, para cada ubicación, el conjunto de conductas de navegación que deben ser activadas.

El planificador hace uso de dos tipos de información:

Modelos formales de los comportamientos: Modelo DES, precondiciones, condiciones de operación y postcondiciones.

Mapa a priori del entorno: Información topológica.

La descripción de la topología y la funcionalidad de los comportamientos es codificada como conocimiento declarativo para mejorar la adaptabilidad y escalabilidad del sistema.

2.5.10. Arquitectura GLAIR

GLAIR (*Grounded Layered Architecture with Integrated Reasoning*) es una arquitectura de tres niveles basada en comportamientos. La propuesta [Hexmoor et al., 1992] se basa en una organización de comportamientos desde niveles básicos de acciones reflejas inconscientes hasta niveles altos de comportamientos deliberativos, dando lugar a un pirámide de comportamientos similar a las utilizadas en disciplinas como Visión por Computador para reducir el volumen de la información a manipular.

Esta arquitectura se ha probado tanto a nivel de simulación como en implementaciones sobre robots reales [Hexmoor et al., 1993].

Estructura

GLAIR está formada por tres niveles (ver figura 2.12):

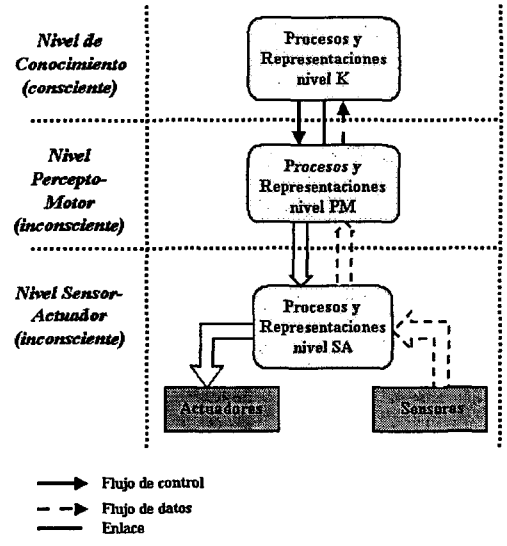

Figura 2.12: Niveles de la arquitectura GLAIR.

- Nivel de conocimiento o KL (*Knowledge Level*).
- Nivel percepto-motor o PML (*Perceptuo-Motor Level*).
- Nivel sensori-actuador o SAL (*Senso-Actuator Level*).

El nivel KL constituye un nivel de planificación que opera sobre objetos de alto nivel, metas, eventos y estados. Este nivel se enlaza parcialmente con el nivel inferior, el PML, donde se manejan representaciones con un mayor nivel de detalle. En el PML se sitúan comportamientos intermedios sobre los que se aplican técnicas de aprendizaje para acabar convirtiéndolos en comportamientos automáticos. En el nivel más bajo, el SAL, se sitúan las acciones reflejas que operan de forma autónoma.

En GLAIR se propone una clasificación de los comportamientos en cuatro tipos:

Reflejos: Operan como ciclos cerrados de percepción-acción, simulando acciones reflejas.

Reactivos: Precisan un nivel mínimo de procesamiento, aunque operan a nivel subconsciente. Equivalen a comportamientos aprendidos.

Situados: Requieren un mayor nivel de procesamiento, con análisis del estado interno y externo.

Deliberativos: Comportamientos de alto nivel con etapas de procesamiento y razonamiento.

Adaptación

En GLAIR se aplican mecanismos de aprendizaje al objeto de trasladar el resultado de comportamientos de alto nivel exitosos a comportamientos reflejos o reactivos que mejoren los tiempos de respuesta y reduzcan el consumo de recursos.

2.5.11. Otras arquitecturas

Evidentemente, existen muchas más propuestas de arquitecturas robóticas que se podrían incluir en este resumen. A continuación se presentan, sin entrar en detalles, algunas de ellas:

ORCCAD: Propuesta del INRIA que ofrece un entorno completo de desarrollo de aplicaciones robóticas. Se parte de la definición de tareas robóticas elementales que se combinan en procedimientos robóticos de más alto nivel [Simon et al., 1997]. Se incluyen herramientas de verificación formal.

GUARDIAN: Control en entornos con tiempos de respuesta críticos (sistemas de soporte vital [Hayes-Roth et al., 1992]) combinado con técnicas de inteligencia artificial adaptativas [Hayes-Roth, 1995].

2.6. Los Lenguajes: Estudio de Casos

Como alternativa a las arquitecturas robóticas específicas, y en muchas ocasiones como parte integrante de su implementación, están los lenguajes de síntesis. El objetivo de estos lenguajes es sistematizar la definición de los diferentes objetos de un sistema de forma que se favorezca su validación. Las aplicaciones generadas pueden ser de esta forma evaluadas en términos de validez formal y comportamiento temporal.

2.6.1. ESL

ESL (*Execution Support Language*) es un lenguaje propuesto por Gat [Gat, 1997*a*] para codificar conocimiento relativo a la ejecución para agentes autónomos. Deriva en gran medida del sistema RAPs, aunque con un enfoque más práctico, orientado a la flexibilidad y comodidad de uso. ESL está implementado como una extensión de Common Lisp.

ESL incorpora construcciones, entre otras, para implementar las siguientes características:

Manejo de excepciones: Se hace uso del concepto de fallo consciente, que expresa la conveniencia de que, puesto que no es posible construir un sistema que no falle bajo ninguna circunstancia, al menos debe intentarse que la situación de fallo siempre sea detectable. Algunas construcciones disponibles son FAIL, WITH-RECOVERY-PROCEDURES y WITH-CLEANUP-PROCEDURES.

Especificación de metas: Permite indicar métodos para alcanzar diferentes metas mediante las construcciones ACHIEVE y TO-ACHIEVE.

Manejo de tareas: ESL soporta la ejecución de múltiples tareas de forma concurrente. Es posible, por ejemplo, especificar grupos de tareas (TASK-NET), sincronizar las ejecuciones mediante eventos (WAIT-FOR-EVENTS, SIGNAL, CHECKPOINT-WAIT, CHECKPOINT), e indicar condiciones de fallo (WITH-GUARDIAN).

Base de datos: Se incluyen instrucciones de manejo de una base de datos lógica, como son ASSERT, RETRACT o WITH-QUERY-BINDINGS.

Mecanismos de bloqueo: Para evitar inconsistencias en el manejo de variables compartidas.

2.6.2. TDL

TDL (*Task Description Language*) [Simmons y Apfelbaum, 1998] es un lenguaje diseñado para simplificar el desarrollo de programas de control para robots. Incluye soporte específico para la implementación del control al nivel de tarea. Desde la perspectiva de su implementación TDL es una extensión de C++ soportada sobre una biblioteca de comunicaciones denominada TCM (*Task Control Management*).

Este lenguaje incorpora características como las siguientes:

- Descomposición de tareas.
- Monitorización de la ejecución.
- Sincronización.
- Manejo de excepciones.

TDL deriva a partir de TCA (*Task Control Architecture*) [Simmons, 1992], una propuesta que combina control a nivel de tarea con comunicación entre procesos empleando paso de mensajes.

La representación básica utilizada en TDL son los árboles de tareas, donde los nodos están asociados a una determinada acción (cuya ejecución puede bien tener éxito o fracasar). Los nodos pueden pertenecer a los siguientes tipos:

Comando: Constituyen comportamientos ejecutables, y se ubican normalmente en las hojas del árbol.

Meta: Esto nodos representan tareas de alto nivel, y su acción consiste en expandir el árbol en nuevos nodos hijo que a su vez pueden ser metas o comandos.

Monitor: Es un nodo cuya acción puede invocarse repetidamente ante la detección de un determinado evento.

Excepción: Son nodos asociados a un determinado tipo de fallo, para el que bien activan acciones de recuperación, bien se limitan a propagarlo.

Desde el punto de vista de su manipulación, un nodo del árbol de tarea puede encontrarse en uno de los siguientes estados : "deshabilitado", "habilitado", "activo" o "completado". El nodo está deshabilitado siempre que no se den las restricciones de sincronización establecidas. Cuando estas condiciones se cumplen, el nodo pasa a estar habilitado y a la espera de que existan recursos suficientes para su ejecución. Si éste es el caso, pasa a estar activo hasta que la ejecución finaliza, transitando al estado "completado". Cuando el estado se refiere a un subárbol completo se habla de expansión, para los nodos tipo meta, o de ejecución, para los nodos tipo comando.

Las restricciones de sincronización son de dos tipos: habilitación y finalización. Estas restricciones indican que un determinado nodo, ante la presencia de un cierto evento, puede iniciar su ejecución (habilitación) o finalizar ésta (finalización).

Veamos algunos ejemplos de código:

a) Expansión y sincronización:

```
t1: spawn a(1);
t2: spawn a(2);
spawn b(3) with
    disable until t1 execution completed,
    disable expansion until t2 handling active;
```

b) Excepciones:

```
Goal navigateToLocn (double x, double y)
{
    with exception
        ("Overheating": handleOverheating(x, y),
         "no path": handlePlannerFailure()) do {
         ...
         }
}
```

c) Monitores:

```
Monitor monitorPickup ()
    max triggers = 1, max activations = 15,
    period = 0:0:1.5
{
    if (mailIsGone()) {
        trigger;
        with (parallel) do {
            spawn speak("Thank you");
            spawn notifySender();
        }
    }
}
```

De forma paralela, se han desarrollado diferentes herramientas de apoyo a la programación, como son un compilador para las definiciones de tareas y diversas utilidades de visualización y depuración.

2.6.3. Robot Schemas

Los esquemas robóticos (RS) [Lyons, 1990] proporcionan una representación basada en procesos para los planes de tarea. Se trata de una representación específicamente diseñada para trabajar con tareas robóticas. RS es un lenguaje desarrollado a partir de la investigación en áreas como el álgebra de procesos y el control en sistemas de eventos discretos, y aportan un sustrato formal sobre el que poder demostrar determinadas propiedades del sistema.

RS proporciona seis operadores de composición de procesos:

Secuencial: $P = R; S$, donde S se ejecuta tras R.

Concurrente: $P = R \parallel S$, donde S y R se ejecutan en paralelo.

Condicional: $P = R : S$, S se ejecuta si R finaliza con éxito.

Desactivación: $P = R\#S$, ejecución en paralelo hasta que uno de los dos procesos termina.

Recurrente síncrona: $P = R :; S$, es una macro que se expande según $R :; S = R : (S; (R :; S))$, con lo que S se ejecuta repetidamente hasta que la ejecución de R falla.

Recurrente asíncrona: $P = R :: S$, es una macro que se expande según $R :: S = R : (S \parallel (R :; S))$, de forma que se lanzan instancias de S en paralelo hasta que la ejecución de R falla.

Un ejemplo para un plan de navegación en RS sería el siguiente:

```
Plan = (NoDestino?:;Navega)#(Timeout?:Fallo)

Navega = (VíaLibre?:MueveHaciaDestino) (NoVíaLibre?:Esquiva)

Esquiva = (ObstáculoIzquierda?:GiraDerecha)(ObstáculoDerecha?;GiraIzquierda)
```

Conjuntamente con RS, se incluye una potente herramienta de análisis. Esta herramienta toma como entrada un plan en forma de secuencia de declaraciones en RS y permite demostrar diferentes propiedades como son la ausencia de bloqueos o la accesibilidad del estado final.

2.6.4. Otros lenguajes

Otras propuestas de lenguajes interesantes son las siguientes:

Colbert: Lenguaje utilizado en la arquitectura Saphira para la definición de comportamientos de bajo nivel [Konolige, 1997].

Maestro: Lenguaje diseñado para definir el controlador de una aplicación robótica [Coste-Manière y Turro, 1997]. Permite aplicar técnicas de validación formal del código antes de su implementación. Maestro incluye operadores de combinación para coordinar acciones robóticas elementales. La definición resultante se compila para producir un controlador Esterel [Berry, 2000].

RPL: Lenguaje derivado de RAPs para especificar planes de alto nivel incluyendo mecanismos de monitorización y tratamiento de excepciones [McDermott, 1991].

2.7. OROCOS

OROCOS (*Open RObot COntrol Software*, http://www.orocos.org/) es una iniciativa reciente para desarrollar código abierto en entornos de robótica. Los objetivos principales de esto proyecto son:

- Código abierto.
- Propuesta estable a largo plazo.
- Componentes software como elementos básicos:
  - Control en tiempo real.
  - Interfaz con el hardware.
  - Comunicaciones.
  - Cinemática y dinámica.
  - Control a nivel de tarea.
  - Percepción.
- Orientado a múltiples niveles de usuario: bajo nivel (infraestructura), diseñadores de componentes, desarrolladores de aplicaciones, usuarios finales.
- Sin restricciones sobre la arquitectura.
- Énfasis en desarrollo de patrones para robótica, sobre múltiples plataformas y lenguajes de programación.

Importantes grupos de investigación dan soporte a este proyecto, incluyendo entre ellos el KTH (http://cogvis.nada.kth.se/orocos/), el LAAS (http://www.laas.fr/ mallet/orocos/) o el FAW (http://www1.faw.uni-ulm.de/orocos/). Actualmente, los esfuerzos se concentran en el desarrollo de la infraestructura de base: comunicaciones y primeros componentes para control de bajo nivel.

Capítulo 3

Propuesta de Sistema Percepto-Efector Distribuido

En este capítulo se describe el modelo arquitectónico de sistema percepto-efector distribuido propuesto en esta tesis. Se realiza una descripción del mismo tanto desde el punto de vista estructural como funcional.

3.1. Introducción: Objetivos de Diseño

Los sistemas percepto-efectores abordan aplicaciones que van siendo cada vez más exigentes. Ello implica una creciente complejidad que precisa de técnicas y herramientas de programación específicas para garantizar el éxito de los desarrollos a largo plazo. Las aplicaciones "ad hoc", aunque efectivas en muchos casos, dejan de ser interesantes cuando se intenta su extensión o su traslado a otros contextos.

Los principales objetivos que se persiguen en este enfoque a largo plazo son múltiples. Así, en la fase de especificación o diseño de las aplicaciones podemos hablar de:

Facilitar el diseño: Deben ofrecerse al desarrollador elementos de construcción claramente definidos que ayuden a diseñar las aplicaciones de una manera rápida y sistemática. La separación de los aspectos relacionados con el control, las comunicaciones y la propia funcionalidad de los elementos o módulos de un sistema aporta ventajas como las siguientes:

- Evita errores de programación en las comunicaciones (normalmente muy difíciles de depurar), lo que incide en una mayor robustez.

- Permite establecer estrategias de control y adaptación - que como se verá - son independientes de los objetivos funcionales de cada módulo y, por tanto, genéricos.

Reutilización del código: La transferencia de resultados entre diferentes equipos de investigación contribuye a la consolidación de etapas y acelera la consecución de nuevas metas.

Para la fase de ejecución de las aplicaciones resultan interesantes los siguientes aspectos:

Reactividad: Se persiguen sistemas que respondan con rapidez a los estímulos que se reciban del entorno.

Robustez: Las condiciones de funcionamiento de este tipo de sistemas son a menudo adversas y cambiantes, lo que demanda mecanismos de tratamiento de excepciones y adaptación.

Los objetivos expuestos han guiado el diseño y la implementación de un sistema percepto-efector distribuido. Se trata de un sistema de propósito general, que permite dar soporte a diferentes tipos de aplicaciones, desde las más simples monotareas hasta aplicaciones jerárquicas distribuidas.

En este capítulo se realiza una descripción detallada del sistema propuesto a nivel básico, tanto desde un punto de vista estructural como funcional. Se dejan para capítulos posteriores los apartados del control/adaptación y el aprendizaje.

3.2. Estructura del Sistema

Ante la diversidad de campos de aplicación y la variedad de hardware y software implicados en los sistemas percepto-efectores, se persigue una estructura fácilmente extensible a múltiples máquinas que no imponga restricciones a priori sobre la arquitectura finalmente implementada.

El sistema percepto-efector que se propone está integrado por un conjunto de elementos modulares que se componen para dar lugar a diferentes entidades funcionales a distintos niveles. Estos elementos básicos incorporan unidades de procesamiento, comunicaciones y control, pudiendo interconectarse entre sí para configurar diferentes arquitecturas.

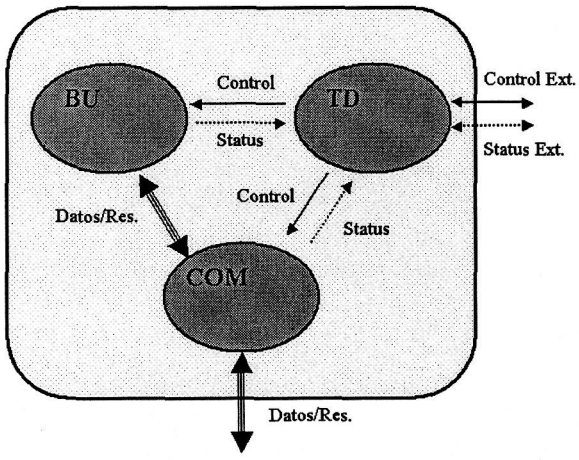

Figura 3.1: Módulo genérico BU-TD-COM.

3.2.1. Los módulos BU-TD-COM

El sistema se estructura en elementos modulares genéricos denominados módulos BU-TD-COM (figura 3.1). Estos elementos pueden considerarse como bloques básicos en la construcción de una aplicación. Cada módulo BU-TD-COM está a su vez constituido por las siguientes unidades:

- Una unidad de procesamiento (Bottom-Up o BU).
- Una unidad de control (Top-Down o TD).
- Una unidad de comunicaciones (COM).

Los módulos BU-TD-COM derivan de una serie de trabajos previos [Cabrera-Gámez, 1994] en el área de la visión por computador y de los sistemas basados en conocimiento [Hernández-Sosa et al., 1999]. Las ideas allí expresadas se extienden ahora al caso de sistemas percepto-efectores genéricos.

La estructura modular permite una clara separación entre los aspectos de procesamiento, control y comunicaciones que es altamente deseable en cualquier sistema percepto-efector. Esta estructura permite asimismo construir sistemas jerárquicos multinivel con diferentes configuraciones, manteniendo, sin embargo, la misma organización interna.

La unidad BU

La unidad de procesamiento o BU se encarga de transformar la información que recibe como datos de entrada en resultados. Estos resultados pueden estar dirigidos bien a otras unidades de procesamiento o hacia efectores finales. La transformación de datos realizada en estas unidades puede ser de naturaleza diversa, tales como la obtención de lecturas directas de los sensores físicos, las transformaciones numéricas sobre las mismas, o las operaciones de simbolización o procesamiento de datos simbólicos.

La función concreta que se desempeñe en una unidad BU dependerá del código que se cargue en la misma para su ejecución. Desde esta perspectiva, una unidad BU puede considerarse una unidad de procesamiento genérica, cuya función concreta depende de las instrucciones que ejecute.

El código de las unidades BU, como se explicará en detalle en la descripción funcional, consta de una sección de inicialización que se ejecuta al activarse el módulo, una sección principal, una sección de error para situaciones de fallo, y una sección de finalización que se ejecuta al desactivarse el módulo.

La unidad TD

La unidad de control o TD es la responsable de verificar el buen funcionamiento del módulo, supervisando la operación de las unidades BU y COM. Estas funciones incluyen el control de la correcta inicialización y finalización del módulo, la monitorización de la frecuencia de ejecución, la detección de bloqueos y otros errores. También se encarga de distribuir la información de estado hacia niveles de jerarquía superiores o de trasladar señales de control hacia módulos subordinados. Para responder a estos objetivos, el TD se estructura por un lado como un proceso de control que se activa periódicamente (en forma de watchdog), y por otro como un bucle de atención de mensajes para atender a las señales que se reciban.

La unidad COM

La unidad de comunicaciones o COM es la encargada de distribuir los datos desde los productores a los consumidores dentro del sistema. Esta transferencia puede tener carácter local o remoto en función de que el destinatario se encuentre o no en la misma máquina, y se realiza de forma concurrente con el procesamiento (BU) y el control (TD).

3.2.2. Flujos de información

El intercambio de información dentro del sistema emplea diferentes mecanismos, siendo lo habitual la utilización de memoria compartida para las comunicaciones internas al módulo y las señales para las comunicaciones entre módulos. Las señales difundidas a través del sistema se establecen según los tres tipos siguientes:

- Señales de datos.
- Señales de control.
- Señales de estado.

Las señales de datos son usadas para transferir el resultado del procesamiento de las unidades BU a otros módulos. La unidad de comunicaciones conduce estas señales desde los módulos productores a cada uno de los módulos consumidores para su procesamiento. Las señales de control contienen comandos dirigidos desde los módulos con mayor jerarquía hacia módulos de nivel inferior con el objeto de configurarlos o alterar su funcionamiento. Por último, las señales de estado son las encargadas de notificar diferentes eventos desde niveles inferiores hacia los controladores de niveles superiores para su análisis, o simplemente son usadas para difundir información de estado a través de todo el sistema.

En la figura 3.1 se muestran estos flujos de señal en el entorno de un módulo BU-TD-COM genérico.

3.2.3. Infraestructura

Todo el sistema ha sido desarrollado utilizando como soporte un sistema de comunicación y control para sistemas distribuidos basado en agentes. Esta infraestructura permite la definición de los módulos, las unidades y las señales intercambiadas entre ellos de forma cómoda y sistemática. Cada unidad es implementada como una hebra independiente, lo que permite respetar los principios de independencia funcional y modularidad sin por ello perder la capacidad de disponer de diferentes mecanismos de intercambio de información.

En la siguiente sección se describe la capa de bajo nivel y su utilización en el soporte del sistema propuesto.

3.3. Infraestructura de Desarrollo: Comunicaciones y Control

El desarrollo del sistema está soportado por un sistema de comunicaciones y control (en adelante SCC) que modela esquemas de control como un conjunto de entidades o agentes débilmente acoplados que interactúan entre ellos [Domínguez-Brito et al., 2000]. El mecanismo de interacción empleado para comunicar entre sí a los agentes son las señales. Cada señal transporta información desde el agente que la envía hasta el que la recibe, permitiéndose una comunicación sin restricciones (todos con todos). Aunque inicialmente empleado para la implementación de esquemas de control para sistemas de visión activa, este entorno puede extenderse de forma natural a otro tipo de sistemas, como es este caso.

El objetivo del SCC es permitir la definición del comportamiento del sistema global y el de los agentes individuales empleando señales. Para cada agente es posible modelar tanto su actividad individual como las interacciones con otros agentes, lo que a su vez define el comportamiento de todo el sistema en su conjunto. El SCC sistematiza la definición de los agentes, qué señales se reciben y qué acciones tomar en respuesta a las mismas. Asimismo se modela la definición de las propias señales y la información que transportan. Se proporcionan esqueletos tanto para los agentes como para las señales, de forma que únicamente es necesario rellenar determinadas secciones de código para alcanzar un sistema en estado operativo.

La herramienta realiza de forma transparente el acceso en paralelo o de forma concurrente a los datos compartidos, de forma que los desarrolladores no tienen que preocuparse por este aspecto. Toda esta funcionalidad hace uso de los recursos típicos que proporciona un sistema operativo multitarea. En sistemas monoprocesador, el paralelismo entre agentes se alcanza por medio de la concurrencia. En sistemas multiprocesador o en sistemas distribuidos, el paralelismo se aprovecha de forma natural distribuyendo los agentes existentes entre los procesadores disponibles.

El SCC permite la asignación de prioridades a cada agente, de forma que pueden definirse jerarquías. Así, es posible asignar prioridades más elevadas a agentes que desempeñan tareas de bajo nivel frente a los que realizan tareas de alto nivel. Otro ejemplo es la priorización de los agentes con baja latencia frente a los que presentan una latencia elevada.

En resumen, esta herramienta permite modelar esquemas de control para sistemas percepto-efectores apoyándose en el concepto de agentes que se comunican por medio de

señales. Una vez el sistema ha sido diseñado, simplemente deben rellenarse los esqueletos de los agentes con las respuestas a las diferentes señales y los esqueletos de las señales con los datos transportados. Como demostración práctica se han desarrollado diferentes sistemas de seguimiento visual [Hernández-Tejera et al., 1999] y aplicaciones en robótica móvil [Cabrera-Gámez et al., 2000].

Esta herramienta se enmarca en un proyecto en pleno desarrollo al que se van añadiendo regularmente nuevas mejoras. El resultado más reciente es la versión denominada CoolBot [Cabrera-Gámez et al., 2001] [Domínguez-Brito et al., 2002].

3.3.1. Tipología de agentes

En el SCC se distinguen los siguientes tipos de agentes:

Agentes fuente: No reciben señales, excepto las de inicio y final. Actúan sólo como emisores de datos, que normalmente corresponden a los suministrados por los sensores del sistema.

Agentes destino: No envían señales, limitándose a recibirlas. En general están asociados con componentes efectores, a los que envían comandos generados a partir de las señales que reciben.

Agentes genéricos: Pueden tanto enviar como recibir señales, sin restricción alguna por parte de la arquitectura.

Agentes distribuidores: Permiten definir "buses virtuales" en el sistema. Su objetivo es únicamente distribuir las señales a través de los agentes, y sólo tienen sentido cuando una misma señal debe ser recibida por múltiples agentes.

Agentes de red: Desempeñan un papel equivalente a los distribuidores pero para el caso de sistemas distribuidos, en los que las señales pueden ser enviadas entre agentes que residen en máquinas diferentes.

3.3.2. Tipología de señales

El SCC proporciona diversos tipos de señales:

Señales de inicio y fin: Son señales que no transportan información. La señal de inicio se emite cuando empieza la ejecución del sistema, y la de fin cuando la

ejecución termina. Ambas señales están dirigidas a todos los agentes del sistema, al tratarse de señales por defecto que son aportadas por la arquitectura de forma transparente.

Señales tipo timer: Esta señales son periódicas y pueden programarse para que el agente desarrolle algún tipo de actividad a intervalos regulares. Son definidas transparentemente por la arquitectura y no transportan información.

Señales genéricas: Constituyen el resto de señales del sistema y pueden ser definidas libremente por el usuario con propósito general.

3.3.3. Guías de diseño

Para usar el SCC en un sistema real debe llevarse a cabo primero un análisis del sistema a implementar. Deben identificarse los agentes, su número, su tipo, etc. Posteriormente han de definirse todas las señales que van a utilizarse, la información que transportan, su origen y su destino. En general, pueden considerarse las siguientes recomendaciones a lo largo de este proceso:

1. Los agentes se corresponden con entidades que poseen objetivos propios.
2. Los agentes interactúan entre ellos y con su entorno, dando lugar al comportamiento global del sistema. Las señales deberían modelar este comportamiento.
3. Típicamente los distribuidores se asocian a señales de elevada frecuencia.
4. Las prioridades elevadas se asignan a agentes de bajo nivel o elevada frecuencia de ejecución y las prioridades bajas a los agentes de alto nivel o baja frecuencia.

El problema fundamental se centra pues en las fases previas a la implementación, en las que deben identificarse acertadamente los agentes que van a formar parte del sistema y las señales que necesitan intercambiarse para obtener el comportamiento deseado.

3.3.4. Implementación del sistema percepto-efector

La concepción modular del diseño de nuestro sistema facilita enormemente su desarrollo sobre el SCC. En esta implementación se han utilizado agentes genéricos

para la definición de las unidades BU, TD y COM. Además, se hace uso de los agentes distribuidores y de los agentes de red en implementaciones distribuidas.

Se han utilizado todos los tipos de señales disponibles: de inicio y fin, genéricas y tipo timer. Las señales de inicio y fin se utilizan en el arranque y la parada del sistema, mientras que las señales genéricas transportan señales de datos, control y estado a través del sistema en ejecución. Las señales se envían directamente entre módulos cuando el origen y el destino son simples, y por medio de distribuidores cuando la misma señal debe llegar a diferentes módulos. Esta última característica es muy interesante a la hora de comandar múltiples módulos de forma simultánea con una gran eficiencia y sin introducir sobrecarga en el sistema. Las señales tipo timer se utilizan para implementar bucles de control internos a cada módulo, tanto para fijar frecuencias de control como de operación.

Se ha comprobado que el overhead introducido por el control de bajo nivel del SCC sobre el sistema final es mínimo, lo que hace que éste no sea un factor determinante en la escalabilidad de las aplicaciones.

3.4. Descripción Funcional del Sistema

La organización funcional del sistema debe alcanzar un equilibrio entre nivel de abstracción y flexibilidad. Por un lado, resulta beneficioso ofrecer al diseñador elementos funcionales que le liberen de tratar con detalles de bajo nivel, lo que permite concentrar los esfuerzos en el diseño de la aplicación. Por otra parte, un nivel excesivo de normalización resta libertad en el desarrollo, limitando los recursos de programación disponibles.

La organización funcional del sistema percepto-efector que proponemos se basa en la siguiente jerarquía de objetos (ver ejemplo en la figura 3.2):

- Estados.
- Tareas.
- Módulos funcionales: Supervisores, sensores, diagnósticos, acciones y actuadores.

Los módulos funcionales son las unidades funcionales básicas reconocidas por el sistema. La combinación de los módulos da lugar a las tareas, donde se coordina la ejecución simultánea de varios de ellos. Por último, los estados agrupan a conjuntos de tareas que deben estar activas al mismo tiempo.

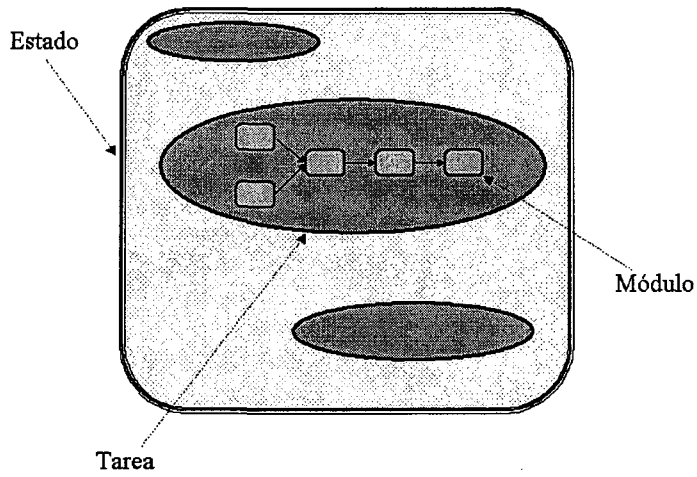

Figura 3.2: Ejemplo mostrando las relaciones entre estado, tareas y módulos funcionales.

3.4.1. Estados y tareas

Los estados constituyen configuraciones fijas de tareas establecidas como parte de la solución al problema que pretende resolver el sistema. Pueden ser el resultado de la descomposición de la misión encargada al sistema por parte de un planificador de alto nivel o bien una configuración fija comandada por el usuario. La evolución del sistema viene determinada por los estados definidos y el grafo que resulta de sus posibles transiciones a otros estados.

En un instante de tiempo determinado existe en el sistema un único estado activo, con todas sus tareas en ejecución y las correspondientes transiciones activadas. Esta situación se mantiene hasta que se dispara alguna de las transiciones asociadas a ese estado, momento en el que pasa a estar activo el estado al que conduce dicha transición. Las transiciones toman la forma de condiciones lógicas de alguno de los siguientes tipos:

- Test sobre el entorno.
- Test sobre variables de estado internas.
- Finalización correcta de una tarea.
- Finalización de una tarea con error.

En la figura 3.3 se muestra de manera esquemática un conjunto de estados y transiciones. Los estados se representan por medio de rectángulos redondeados, mien-

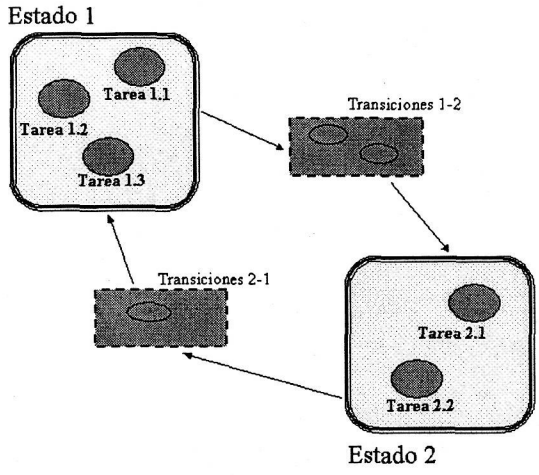

Figura 3.3: Ejemplo de estados y transiciones entre estados.

tras que las transiciones aparecen como rectángulos de borde discontinuo. Las flechas conectan los estados de partida con los estados de llegada de cada transición.

Las tareas surgen de la asociación de las capacidades perceptoras, de procesamiento y efectoras del sistema físico. En la mayoría de los casos se constituyen como bucles de control repetitivos que enlazan, de una manera más o menos directa, los sensores con los actuadores del sistema físico. El sistema puede mantener activas simultáneamente a varias tareas, las cuales se ejecutan de forma paralela o concurrente dependiendo de la relación que exista entre su número y el número de procesadores disponibles. Las tareas están integradas por módulos, que pueden ser o no compartidos en función de los tipos de tareas en ejecución. Los tipos de módulos funcionales que integran las tareas se analizarán más adelante.

Una tarea lleva asociados los siguientes parámetros:

- Nivel de prioridad.
- Límite temporal.

El nivel de prioridad indica la relevancia de la tarea, y se somete a consideración en situaciones de escasez de recursos o de conflicto en el acceso a elementos compartidos. El límite temporal expresa el tiempo máximo que puede transcurrir entre dos ejecuciones consecutivas de la tarea, en el caso de tareas periódicas, o el tiempo de espera máximo para obtener respuesta ante un estímulo de entrada, en el caso de las no periódicas.

Tipos de tareas

Las tareas que pueden ejecutarse en el sistema son muy variadas, aunque se diferencian algunos tipos característicos en función de la relación frecuencia/prioridad:

Tareas críticas: Tareas con alta frecuencia y alta prioridad que suelen ser determinantes para garantizar la integridad del sistema.

Tareas principales: Tareas de frecuencia y prioridad intermedias que desarrollan el núcleo de la misión del sistema.

Tareas accesorias: Tareas de baja frecuencia y baja prioridad que realizan funciones secundarias para la misión principal del sistema (p. e. de medida del rendimiento o autodiagnóstico).

La clasificación anterior contempla sólo tareas periódicas con prioridad fija. Fuera de esta categoría, resulta interesante incluir algunos tipos especiales de tarea:

Tareas no periódicas: Se trata de tareas que se ejecutan de forma irregular, a las que se les exige que una vez recibidos los datos de entrada, la salida correspondiente a los mismos tenga lugar en un tiempo máximo preestablecido.

Tareas posponibles: Son tareas periódicas de frecuencia y prioridad intermedias que poseen un rango de frecuencias admisibles, de forma que en caso de escasez de recursos pueden retardarse hasta un límite máximo sin que ello implique una violación temporal.

Tareas de prioridad variable: Estas tareas tienen un valor de prioridad que es función, normalmente creciente, del tiempo.

Desde un punto de vista topológico, la compartición o no de módulos entre distintas tareas permite diferenciar las siguientes situaciones (figura 3.4):

Sistema desacoplado: En este caso las tareas casi no comparten módulos, con lo que cada una se ejecuta de forma independiente al resto.

Acoplamiento de fuentes: Corresponde a una situación en la que existen pocos módulos productores que suministran datos a múltiples destinos, dando lugar a una estructura de dispersión con un elevado número de tareas.

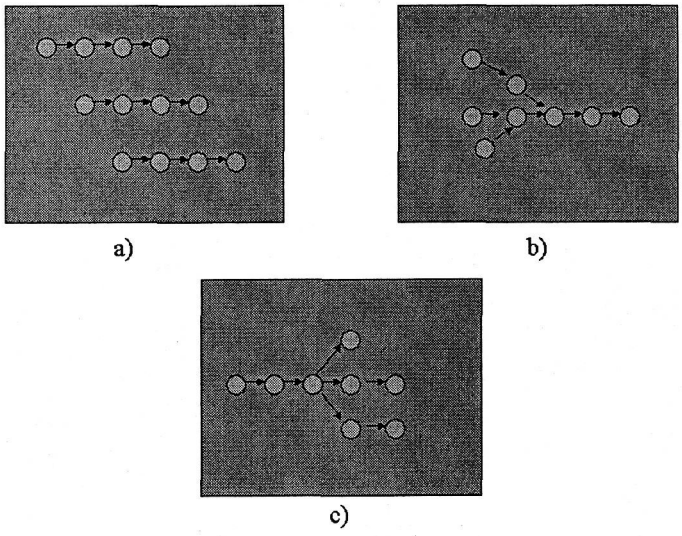

Figura 3.4: Topologías de tareas: a) Desacoplada, b) Concentración, c) Dispersión.

Acoplamiento de destinos: Se origina en el caso de múltiples módulos productores, que suministran datos a un reducido número de tareas, y configuran una estructura de concentración.

Esta clasificación de tareas y, por extensión, del sistema completo es especialmente relevante en los aspectos de control y adaptación. Como se verá en detalle en el próximo capítulo, las acciones de control deben tener en cuenta este análisis para anticipar el efecto que producirá la activación de dichos mecanismos sobre el sistema.

Un ejemplo

Antes de pasar a la descripción de los tipos de módulos, ilustraremos los diferentes elementos presentados a partir de un ejemplo. Supongamos una aplicación para un robot móvil consistente en trasladarse desde una habitación A a otra habitación B, como se ilustra en la figura 3.5.

En una primera aproximación a la solución, podemos pensar en la existencia de dos estados, uno de desplazamiento hacia un punto intermedio especificado por un planificador de alto nivel y otro de reposo. El resultado sería el que se muestra en la figura 3.6. El estado "reposo" simplemente posee una tarea no periódica $T1{,}1$ de notificación de robot listo para nuevo destino, y en el estado "movimiento" existe una tarea $T2{,}1$ en forma de bucle de control que envía comandos a los motores hasta que se alcanza el destino. Existen dos transiciones, $Tr1-2$ y $Tr2-1$, que se muestran en la figura

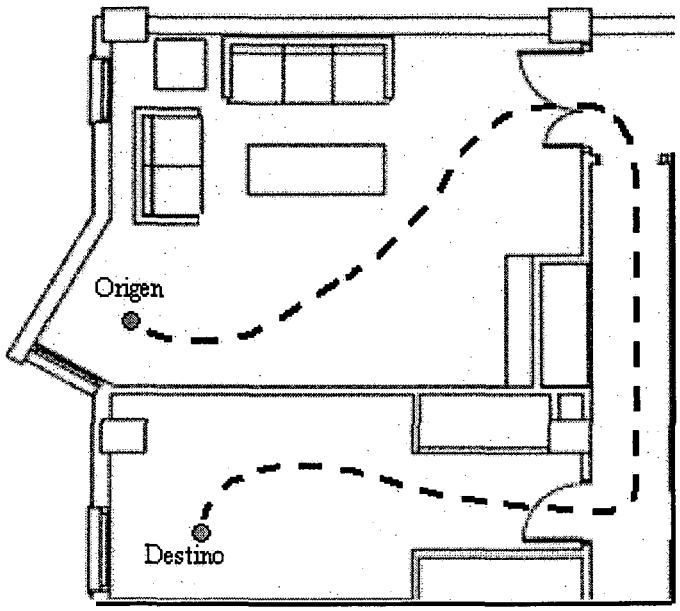

Figura 3.5: Ejemplo de aplicación para un robot móvil.

mediante rectángulos con borde discontinuo. La transición $Tr1-2$ se dispara cuando se detecta la llegada desde el planificador de un nuevo punto de destino a alcanzar, y provoca la salida del estado de reposo y la activación del estado de movimiento. La transición $Tr2-1$ opera en sentido inverso, desde el estado de movimiento al de reposo, cuando la tarea $T2{,}1$ ha finalizado con éxito alcanzando el destino comandado.

Una solución más elaborada consistiría en añadir capacidad de evitación de obstáculos durante la misión. Para ello, como se muestra en la figura 3.7, es necesario incorporar nuevas tareas, estados y transiciones. Al estado "movimiento" se incorpora una nueva tarea, de prioridad elevada, encargada de evitar los obstáculos que aparezcan. Se definen dos nuevos estados, uno al que se transita en caso de que el obstáculo impida el desplazamiento del robot, denominado "bloqueo", y otro al que se llega en caso de fallo en la misión, denominado "error". En el estado de bloqueo se lanza una tarea que trata de alcanzar un espacio libre desde el que volver a intentar llegar al punto objetivo. Al estado de error se puede llegar por distintos motivos, como son la detección de una colisión o el sobrepasar el tiempo máximo asignado a las tareas de navegación o búsqueda de espacio libre de obstáculos.

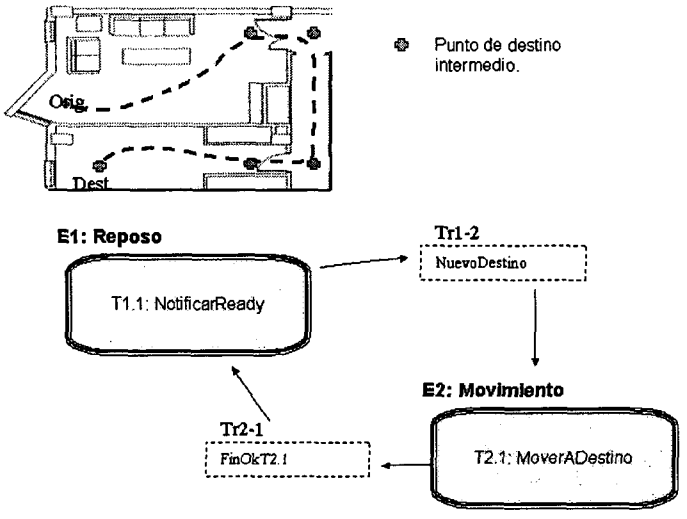

Figura 3.6: Solución simple al problema de navegación.

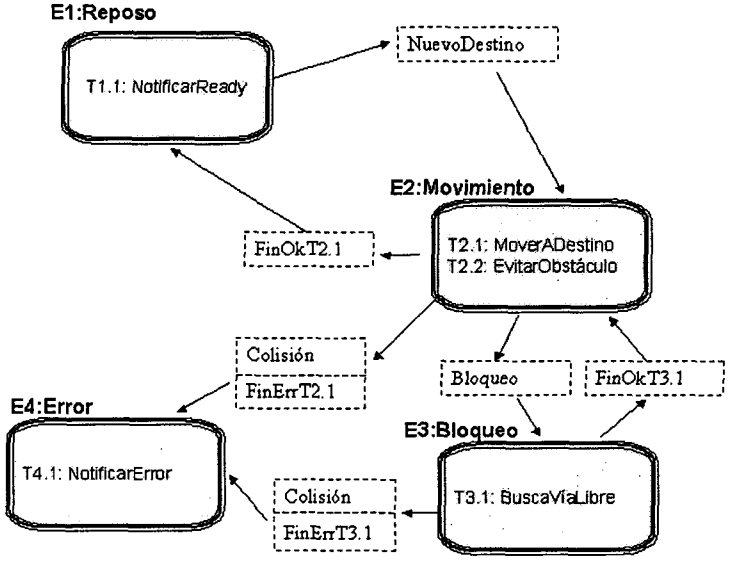

Figura 3.7: Solución para la navegación añadiendo evitación de obstáculos.

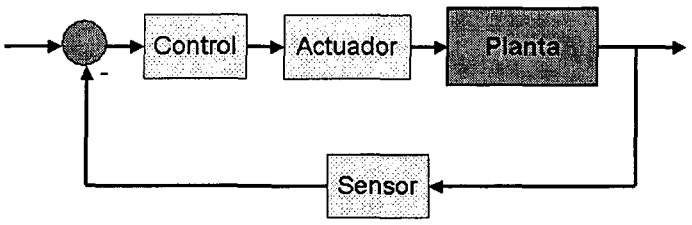

Figura 3.8: Diagrama de un control en bucle cerrado clásico.

3.4.2. Módulos funcionales

Los diferentes tipos de módulos funcionales surgen a partir de la descomposición funcional de las tareas. El objetivo es alcanzar una mayor modularidad en la definición de las tareas, lo que permite sistematizar el desarrollo de las aplicaciones a costa de restar algo de libertad en el diseño. Desde la perspectiva de la claridad de los diseños y la reutilización del software, la conveniencia de esta descomposición está plenamente justificada.

A efectos descriptivos, se puede establecer una analogía comparativa entre una tarea y un sistema de control en lazo cerrado clásico (figura 3.8). Los elementos sensor y actuador del bucle de control tienen una correspondencia directa con los sensores y actuadores que incorpora un sistema percepto-efector y que forman parte de las tareas que éste realiza, como es el caso de los sensores y los actuadores de un robot. En el bucle de control, sin embargo, se asume que el sensor suministra los datos necesarios para incorporarlos directamente a la estrategia de control. En el contexto de un sistema percepto-efector ésta es una visión simplificada, puesto que los datos suministrados por un mismo sensor pueden servir como punto de partida para diferentes tareas, que realizan distintos tipos de procesamiento sobre esos datos. El controlador, por su parte, es quien dicta la estrategia de control, estableciendo las características particulares de cada bucle. Para una tarea, el controlador debe estar representado por un elemento que determine las características de la acción y su evolución.

En sistemas de control multilazo jerárquicos, la coordinación entre los diferentes

bucles viene representada por supervisores de alto nivel. Análogamente, en el sistema propuesto deberán existir elementos similares para mantener la cohesión y la integridad del conjunto.

Por otra parte, consideramos interesante disponer de un mecanismo de comunicación con el sistema cercano al lenguaje natural. Una descripción de tareas en forma de combinaciones de verbo más objeto contribuye a una mayor claridad en la secuencia de comandos, haciéndola más legible. Los verbos son asimilables a la secuencia de control en los sistemas percepto-efectores, mientras que el objeto está relacionado con las percepciones del sistema. Los sensores representan los dispositivos que suministran los datos necesarios para identificar a los objetos en el entorno, mientras que los actuadores son los elementos motores que hacen operativos a los verbos. Esta organización deja la puerta abierta para la incorporación de modificadores de tipo intensificación o atenuación sobre los verbos y los objetos.

En base a la discusión anterior, se definen los siguientes módulos funcionales como objetos básicos en el sistema que se describirán posteriormente en detalle:

- Sensores.
- Diagnósticos.
- Acciones.
- Actuadores.
- Supervisores.

La asociación sensor-diagnóstico está relacionada con los sensores en control clásico y con los objetos en términos lingüísticos, mientras que el conjunto acción-actuador se corresponde con las funciones de comparación y el controlador en control clásico y con el verbo desde la perspectiva del lenguaje natural. Esta separación de la acción y el objeto de la acción facilita de forma directa la modularidad y la reutilización del código. Por ejemplo, puede pensarse en múltiples acciones que se lanzan sobre el mismo objeto, o bien, múltiples objetos que son los destinatarios de la misma acción.

Los módulos estructurales BU-TD-COM son los encargados de hacer operativos estos módulos funcionales, de modo que puedan interconectarse entre sí para construir las tareas que el sistema deba desempeñar. Desde esta relación con los objetos del sistema, un módulo puede encontrarse en uno de los estados que se muestran en la siguiente tabla:

Estado	Descripción
"IDLE"	No se le ha asignado ningún objeto
"READY"	Preparado para la ejecución de un cierto objeto
"RUNNING"	Ejecutando el código asignado
"ERROR"	Situación de error
"SUSPENDED"	Ejecución temporalmente detenida

Tabla 3.1: Descripción de los estados posibles para los módulos del sistema.

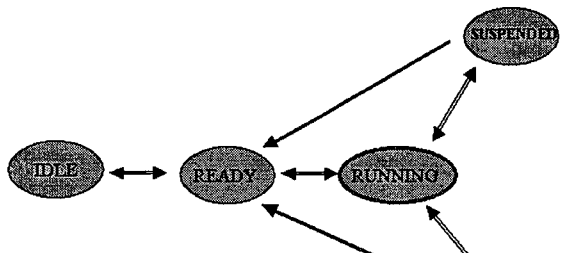

Figura 3.9: Diagrama que muestra las relaciones existentes entre los estados posibles de los módulos del sistema.

La figura 3.9, ilustra los diferentes estados junto con las transiciones que se pueden dar entre ellos. Así, un módulo parte del estado "IDLE" y pasa al estado "READY" cuando se le asigna un objeto determinado para su ejecución. Del estado "READY" se puede pasar bien al estado "RUNNING" para iniciar la ejecución, al estado "IDLE" para desasignar el módulo, o al estado "ERROR" si se ha producido algún error durante la inicialización. Por último, del estado "RUNNING" se vuelve al estado "READY" cuando finaliza la ejecución, se pasa al estado "SUSPENDED" si la ejecución debe interrumpirse temporalmente, o se pasa al estado "ERROR" si se detecta una situación anómala. La transición a cada estado viene determinada por señales externas de control de los supervisores, salvo en el caso del estado "ERROR", al que se puede llegar y del que se puede salir mediante eventos detectados localmente.

Una vez configurados, tendremos módulos sensores, módulos de diagnóstico, módulos de cómputo de acciones, módulos actuadores y módulos supervisores. El código que se ejecuta en cada uno de estos módulos funcionales se organiza en cuatro secciones:

- Sección de entrada.
- Sección principal.
- Sección de error.
- Sección de salida.

La sección de entrada está formada por el código de inicialización del módulo funcional, y consta de rutinas de apertura de dispositivos, reserva de memoria, etc. Es el código que se ejecuta al pasar al estado "READY". La sección de salida contiene código de liberación de recursos y cierre de dispositivos, y se activa al pasar al estado "IDLE". La sección principal es la que contiene el código que se ejecuta, normalmente de forma repetitiva cuando el módulo funcional se encuentra en ejecución, esto es, en el estado "RUNNING". Por último, la sección de error está integrada por los mecanismos de recuperación en caso de fallo. Normalmente, el usuario sólo necesita programar la sección principal, dejando que sea el sistema el que aplique las acciones por defecto para el resto de las secciones. De todas formas, siempre es posible incluir código del usuario en aquellos módulos que precisan condiciones adicionales para su puesta en funcionamiento o recuperación.

Los sensores

Los sensores son elementos que proporcionan datos de entrada de naturaleza fundamentalmente numérica al sistema. Normalmente están asociados a dispositivos reales del tipo cámaras, anillos de ultrasonidos, detectores infrarrojos, sensores de contacto, sensores de temperatura. Estos dispositivos son encapsulados de forma que los detalles particulares quedan recogidos en los controladores de bajo nivel, de forma que su manipulación a alto nivel se presenta homogénea.

Dentro del módulo BU-TD-COM asignado al sensor, la unidad BU ejecuta un código, generalmente muy simple. En la sección de entrada se inicializa el dispositivo físico y se configura para la adquisición. También se reservan los recursos necesarios para la ejecución del código del sensor. En la sección principal se realizan peticiones de datos al sensor y se formatean para su distribución por el sistema, repitiéndose la secuencia a una frecuencia de funcionamiento determinada. La sección de finalización cierra el dispositivo sensor, liberando los recursos captados. La unidad de comunicaciones es notificada cada vez que se genera un nuevo dato y se encarga de su distribución a todos los módulos de cómputo de diagnósticos conectados al sensor.

La unidad de control comprueba periódicamente el estado del sensor físico y la frecuencia de operación. Si se detecta algún error se intenta solventar a nivel local o, si esto no es posible, se notifica a los niveles superiores.

Los diagnósticos

Los diagnósticos son objetos de transformación aplicados a datos de entrada suministrados por sensores o por otros diagnósticos. Los resultados obtenidos son enviados a las acciones para su análisis o a otros diagnósticos para someterlos a un nuevo proceso de transformación. El núcleo de estos objetos está constituido por un algoritmo de procesamiento que opera sobre los datos de entrada.

La unidad BU asociada al cómputo de un diagnóstico ejecuta un código de inicialización con el que se reserva la memoria temporal necesaria para el procesamiento. La sección principal cíclica está formada por instrucciones de procesamiento sobre los datos de entrada o sobre resultados temporales. Por último, la sección de finalización libera las zonas de memoria reservadas.

La unidad de comunicaciones atiende a dos flujos de información. Por un lado, recibe los datos desde los módulos que suministran la información para el cómputo, y que pueden ser tanto sensores como otros diagnósticos. Por otro, transmite los resultados generados a los módulos consumidores, que pueden ser bien acciones, bien módulos de cómputo de otros diagnósticos.

La unidad TD controla la evolución de los tiempos de procesamiento y comprueba si se respeta la frecuencia de operación; además de monitorizar la ausencia de errores y excepciones, tarea común a los TD de todos los objetos del sistema.

Las acciones

Las acciones son objetos de decisión que generan las diferentes salidas que presentará el sistema en función de los estímulos de entrada. Contienen el código de control que analiza el resultado del procesamiento de los diagnósticos y conduce la evolución de la ejecución del sistema, su actividad. Las acciones constituyen, en realidad, el núcleo de las tareas.

En el interior de un módulo BU-TD-COM asociado a una acción, la unidad BU ejecuta las diferentes instrucciones de control presentes en el código, junto con las correspondientes secciones de inicialización y finalización. La unidad COM recibe datos de entrada desde diferentes módulos de cómputo de diagnósticos y envía los comandos

resultantes del procesamiento de la acción a los módulos actuadores conectados a la misma.

La unidad de control, además de guardar la integridad del módulo, tiene en este caso funciones adicionales derivadas de su papel de supervisión de la tarea.

Los actuadores

Los actuadores son dispositivos de tipo terminal que se asocian con los efectores reales del sistema. Son los encargados de traducir las decisiones de control de las acciones sobre el entorno del sistema percepto-efector.

La unidad BU de un módulo actuador ejecuta un código con secciones de entrada y salida, para la inicialización del dispositivo físico y su configuración y la desactivación, respectivamente. En la sección principal se espera por los comandos recibidos desde las acciones y se ejecutan. Las diferentes implementaciones dispondrán de mecanismos para resolver conflictos en caso de existir múltiples fuentes de comandos, pudiendo emplear técnicas como la selección por prioridad, ponderación por voto, fusión difusa, etc.

La unidad de comunicaciones es utilizada únicamente en sentido de entrada para canalizar la información que se recibe desde las acciones. Por último, la unidad de control comprueba el correcto funcionamiento del actuador durante la ejecución de la tarea.

Los supervisores

Los supervisores son objetos encargados de controlar el funcionamiento del sistema en su conjunto. Los supervisores son quienes actúan, además, de interfaz con el usuario, del que reciben las peticiones para la ejecución de tareas. Desde el punto de vista del programador, los supervisores son elementos transparentes que son añadidos de forma automática por el sistema en función de la configuración de estados y tareas que se cargue. A pesar de esto, su comportamiento puede ser modificado a través de un conjunto de parámetros, como es el caso de los mecanismos de adaptación que se verán en el siguiente capítulo.

La supervisión se realiza a múltiples niveles:

- Nivel de tarea.
- Nivel de estado.
- Nivel de sistema.

En el nivel más bajo están los supervisores de la tarea, papel desempeñado por los módulos TD de las acciones. Se encargan de todos los aspectos de monitorización y control ligados al funcionamiento de la tarea como agrupación de módulos. Por encima, están los supervisores de estado, que coordinan la evolución de las tareas pertenecientes al estado de que se trate. Son los responsables de comprobar las transiciones y, en caso de que alguna de ellas se active, enviar las señales correspondientes para que se produzca la transición al estado de destino. En el nivel más alto se encuentran los supervisores de sistema, que interactúan con el usuario y controlan la evolución de los estados que se hayan definido. El esquema habitual consiste en disponer un supervisor por cada máquina sobre la que vaya a distribuirse la ejecución de las tareas, siendo uno de ellos el supervisor principal.

Desde el prisma de la implementación, los supervisores pueden verse como módulos BU-TD-COM que ejecutan dos tipos de código:

Código genérico: Funciones de monitorización y control que son comunes a todas las aplicaciones, con independencia de su naturaleza.

Código específico: Código asociado a las transiciones que se extrae de la definición de los estados suministrada por el usuario.

Construcción de tareas

Las tareas del sistema son el resultado de la combinación de diferentes módulos funcionales. Las tareas constituyen una asociación de uno o más diagnósticos con una acción en la que los diagnósticos representan la parte de procesamiento y la acción la parte de decisión. Algunos ejemplos sencillos serían: "sigue color_rojo", "detecta temperatura_alta", etc.

La información relativa a todas las implementaciones de objetos conocidas por el sistema se recoge en una base de conocimiento. Esta base de datos almacena las características de cada elemento, sus posibilidades de interconexión, requerimientos, etc. El bucle de control de las tareas se completa, a partir de la información contenida en la base de conocimiento, con los sensores que suministran datos de entrada a los diagnósticos y los actuadores que se comandan desde la acción como salidas. Desde la perspectiva del sistema, la acción es quien gobierna la tarea, desempeñando el papel de supervisor de la misma. La figura 3.10 muestra un ejemplo de una tarea simple y las interconexiones entre los diferentes módulos que la configuran.

Para los casos más complejos, la red de sensores y diagnósticos puede crecer y

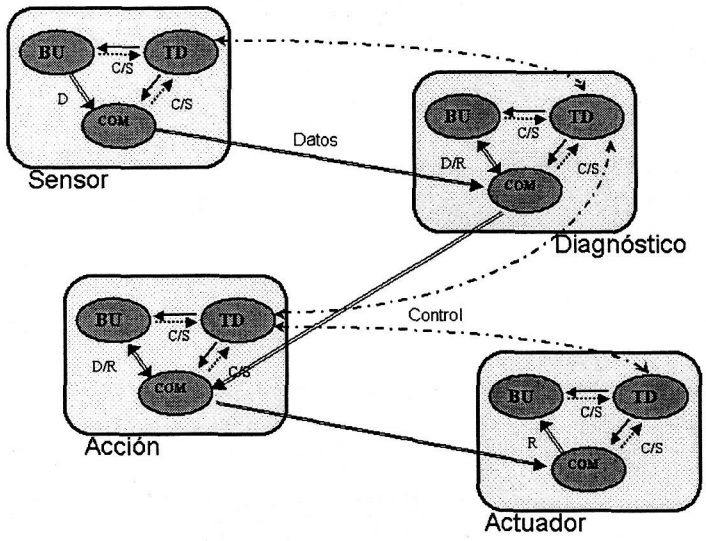

Figura 3.10: Ejemplo de módulos combinados en una tarea simple.

expandirse en múltiples niveles, configurando grafos de procesamiento con elementos compartidos entre múltiples tareas. Nada impide, sin embargo, definir tareas altamente reactivas en las que la conexión entre el sensor y la acción es prácticamente directa, lo que garantiza una latencia mínima entre el estímulo y la respuesta del sistema. Otros tipos especiales de tareas son las acciones puras, en las que no se requieren sensores o el cómputo de diagnósticos, como por ejemplo "parada_emergencia".

El ciclo normal de operación de una tarea comienza con la generación de datos desde el sensor, que son procesados por los módulos de diagnóstico. Los resultados son analizados en las acciones para decidir qué comandos trasladar a los actuadores. Por encima de este nivel, los supervisores se encargan de controlar la evolución del sistema en su conjunto.

3.4.3. La base de conocimiento

La base de conocimiento es el elemento que permite al sistema responder a las peticiones del usuario ejecutando las tareas que se soliciten. En ella se almacena la información relativa a las siguientes entidades:

Tipos de objetos: Donde se describen los diferentes tipos de sensores, tipos de diagnósticos, tipos de acciones y tipos de actuadores que conoce el sistema. Son las referencias utilizadas en la descripción de las tareas.

Implementaciones de objetos: En este caso se recogen cada una de las im-

plementaciones alternativas que existen para cada uno de los tipos genéricos. Se trata de descripciones operativas que incluyen el código que debe ejecutarse en cada caso por parte de las unidades BU.

Tipos de datos: Se realiza una descripción de los diferentes tipos de datos que maneja el sistema. Esta información permite centralizar las referencias que se hagan en cada objeto al *tamaño y tipo de datos de entrada y salida.*

Dispositivos físicos: Consiste en una relación de los controladores que permiten manejar los diferentes dispositivos físicos (sensores y actuadores) conectados al sistema.

Tipos de errores: Donde se define la tipología de errores reconocidos por el sistema.

La organización en tipos e implementaciones resulta fundamental a la hora de pensar en aspectos del sistema como la adaptación o la robustez. Así, el cómputo de un determinado diagnóstico puede tener versiones adaptadas a diferentes entornos de interior o exterior, y una acción de seguimiento puede consumir más o menos energía en función de las velocidades máximas aplicables a los motores. En caso de fallos de dispositivos, esta cualidad resulta aún más importante, pues permite al sistema hacer uso de implementaciones alternativas que no requieran la presencia de los sensores o efectores dañados.

Las condiciones existentes en el momento de la activación guían la selección de las diferentes implementaciones disponibles. Este contexto está formado por la información que el sistema va adquiriendo durante su funcionamiento, y que puede incluir entre otros datos el estado de los sensores y actuadores físicos, o las condiciones y tipo de entorno, como es el caso de los niveles de iluminación y de reserva de baterías.

La información de los tipos de objetos permite establecer relaciones de compatibilidad, de manera que tanto las entradas como las salidas de los diferentes tipos de objetos incluyen información de los tipos compatibles. Esta característica simplifica la definición de las tareas del sistema y permite verificar la consistencia de las conexiones establecidas. Por ejemplo, el diagnóstico "obstáculo" puede computarse tanto a partir de sensores de ultrasonidos como de pares estéreo.

La organización de los errores dentro del sistema en tipos permite aplicarles diferentes tratamientos en los distintos niveles del sistema. Una vez clasificado, el error puede intentar resolverse a nivel local y, si esto no es suficiente, se puede propagar hacia los niveles superiores.

3.4.4. Ejemplos de objetos contenidos en la base de datos

Se presentan algunos ejemplos, sin ánimo de exhaustividad, para ilustrar la estuctura y forma de los objetos.

Tipo de sensor: TS_Visión.

- Sensores: S_CámaraA, S_CámaraB.
- Tipo de dato: Imagen_Color.

Tipo de sensor: TS_Distancia.

- Sensores: S_UltraS, S_Láser.
- Tipo de dato: Mapa_Profundidad.

Tipo de diagnóstico: TD_Obstáculo.

- Diagnósticos: D_ObsCam, D_ObsUltraS.
- Tipo de dato: Mapa_Obstáculos.

Tipo de diagnóstico: TD_ColorVerde.

- Diagnósticos: D_VerdeInterior, D_VerdeExterior.
- Tipo de dato: Imagen_Simbólica.

Tipo de acción: TA_SeguirVisual.

- Acciones: A_SeguirVFast, A_SeguirVSlow.
- Tipo de dato: Comando_Cabeza.

Tipo de actuador: TAct_Cabeza.

- Actuadores: Act_CabezaA, Act_CabezaB.
- Tipo de dato: -

Tipos de error: TE_ComDispositivo, TE_TimeOut.

Ejemplo de tarea de seguimiento/búsqueda visual

Una tarea de seguimiento, podría expresarse de la siguiente forma:

```
SigueVisual ColorVerde
```

El sistema debe entonces seleccionar alguna implementación válida para los tipos de acción y diagnóstico indicados. En esas implementaciones se describen características particulares como son los sensores y actuadores requeridos y el código asociado. Para el ejemplo actual, se activarían un sensor tipo cámara y un actuador tipo cabeza o cuello.

La tarea de seguimiento podría formar parte de una aplicación más compleja en la que se integrasen mecanismos de recuperación en caso de pérdida del objetivo. Por ejemplo, una tarea del tipo

```
BuscaVisual ColorVerde
```

En este caso el objeto se mantiene constante, aunque la acción haya cambiado, lo que contribuye a reducir la cantidad de código necesario para desarrollar la aplicación y permite acelerar los cambios de contexto.

Ejemplo de reutilización de acciones

Como ejemplo complementario está la activación de tareas que aplican la misma acción sobre diagnósticos diferentes. La misma acción de detección en este caso toma como entradas dos diagnósticos complementarios, el color y la textura.

```
Detecta ColorVerde
Detecta TexturaRugosa
```

Una posible utilización de estas configuraciones sería la detección de un objetivo mediante la fusión de múltiples pistas. Bien en la propia acción bien en el actuador, que en este ejemplo de búsqueda pasiva es un simple monitor, se realizaría la combinación de las pistas empleando operadores del tipo and, or, fusión difusa, etc.

3.4.5. Señales

A través del sistema se envían tres tipos de señales: señales de control, señales de estado y señales de datos. Las señales de control fluyen desde los supervisores hacia los

controladores de tarea y de éstos a los módulos sensores, actuadores y de procesamiento de diagnósticos. Algunos ejemplos serían las órdenes de configuración, degradación, iniciar y terminar la ejecución, etc.

Las señales de estado se dirigen desde módulos secundarios a módulos controladores para indicar la detección de un evento determinado o la evolución de alguna variable de interés. Dentro de este grupo tenemos señales como la detección de timeouts o el fin del procesamiento.

Las señales de datos transmiten los resultados que se van generando en el sistema en cada etapa desde los módulos productores a los consumidores. Junto con los datos se transmite información adicional como es la frecuencia de generación, el sello temporal, o el tiempo de procesamiento. Las señales de datos dan al sistema una configuración de máquina de flujo de datos en la que el procesamiento es iniciado en los sensores con la generación de los datos de partida. A partir de aquí, la cadena de procesamiento continúa con los módulos de diagnósticos, que son activados por la recepción bien de datos procedentes de sensores, bien de resultados producidos por otros diagnósticos. El flujo prosigue con las acciones, que son las que finalmente comandan a los actuadores, donde termina el ciclo.

3.4.6. Lenguaje de programación

La ejecución de los diferentes objetos dentro del sistema viene determinada por el código que se procesa en las diferentes unidades BU. Para el caso de los sensores y los actuadores, el código BU es muy simple, consistiendo básicamente en llamadas al controlador de dispositivo correspondiente para realizar bien una adquisición de datos, bien la ejecución de un comando determinado (cuadro 3.2).

Sensores		Actuadores	
Instrucción	**Parámetros**	**Instrucción**	**Parámetros**
Init	-	*Init*	-
Configure	-	*Configure*	-
Acquire	Id. salida	*Command*	Id. entrada
Finish	-	*Finish*	-

Tabla 3.2: Conjunto de instrucciones para los módulos sensores y actuadores.

El código de los diagnósticos contiene referencias a los datos de entrada y a su combinación y procesamiento utilizando determinados algoritmos (tabla 3.3). Simplemente hay que distinguir la obtención de resultados intermedios de la generación de un

```
var_1=Compute_Var(input_1, ''proc_A'')
var_2=Compute_Var(input_2, ''proc_B'')
out_1=Compute_Out(var_1, var_2, ''proc_C'')
```

Figura 3.11: Ejemplo de código de diagnóstico simple. Se genera un resultado a partir de dos datos de entrada (input_1 e input_2) y tres algoritmos de procesamiento (proc_A, proc_B y proc_C).

nuevo resultado final a efectos de su transmisión a las acciones correspondientes (ver el ejemplo de la figura 3.11).

Instrucción	**Datos**	**Parámetros**	**Descripción**
Compute_Var	Id. entrada o var.	Id. var., Id. Proc.	Res. intermedio
Compute_Out	Id. entrada o var.	Id. salida, Id. Proc.	Res. final

Tabla 3.3: Conjunto de instrucciones para los diagnósticos.

El código de las acciones es más rico (tabla 3.4), puesto que debe incluir instrucciones de control de flujo. Básicamente existen procedimientos de análisis (normalmente tests muy simples) de datos de entrada, saltos condicionales, instrucciones de salida e indicaciones de fin de ciclo o fin de procesamiento (ver el ejemplo de la figura 3.12). También existen contadores de propósito general para su utilización en bucles y registro de eventos.

Instrucción	**Datos**	**Parámetros**	**Descripción**
If/ElseIf/EndIf	Id. entrada o variable	Id. proc.	Comparación
While/EndWhile	Id. entrada o variable	Id. proc.	Bucle
Command	Id. variable	Id. salida	Comando de actuador
E_Cycle	-	-	Fin de ciclo
ResetCounter	-	Id. Counter	Puesta a cero de contador
IncCounter	-	Id. Counter	Incremento de contador
EndOk	-	-	Fin de ejecución satisfactoria
EndErr	-	-	Fin de ejecución con errores

Tabla 3.4: Conjunto de instrucciones para las acciones.

La instrucción de fin de ciclo *E_Cycle* se emplea como marca a efectos de control en acciones periódicas. Esta información es utilizada por los controladores y supervisores para realizar verificaciones de frecuencia de operación.

Las señales de finalización de tarea *EndOk* y *EndErr* indican que la tarea ha terminado con éxito o ha fallado, respectivamente. Conviene aclarar que los errores de finalización de tarea son errores lógicos, que no tienen que ver con los errores físicos que puedan tener lugar durante la ejecución de la misma tarea. El programador de la acción

```
While(-, ''true'')
If(input_1, ''test_A'')
Command(out_1, ''cmd_A'')
ElseIf
Command(out_1, ''cmd_B'')
EndIf
E_Cycle
EndWhile
End
```

Figura 3.12: Ejemplo de código de una acción que ejecuta un bucle de envío de comandos hasta que se detecta una determinada condición de salida.

indica con ellos que la tarea no puede completarse satisfactoriamente, a fin de que este hecho se tenga en cuenta. Un ejemplo de error lógico puede ser el que se produce en una tarea de seguimiento que ha perdido el objetivo, lo que probablemente motivará una transición hacia un nuevo estado de búsqueda. Un error físico, en cambio, sería un fallo hardware en la electrónica de una cámara.

3.5. Estudio de Caso: Segmentación de Imágenes

La organización en módulos funcionales se ha estudiado, especialmente para el caso de los sensores y la combinación de diagnósticos, en un conjunto de aplicaciones de segmentación de imágenes [Hernández-Sosa et al., 1995]. Se presentan aquí algunos resultados como muestra de la utilización de la organización modular propuesta en un caso práctico.

El objetivo concreto de este trabajo era la selección, implementación y ajuste de los diagnósticos necesarios para realizar la segmentación de un conjunto de imágenes de exterior. Las figuras 3.13 y 3.14 muestran dos de las imágenes de entrada utilizadas.

Para este caso, el sistema diseñado a partir de los módulos funcionales tiene un nivel básico de captura de datos, donde se sitúa el sensor de cámara. En una primera capa de procesamiento se ubican los diagnósticos que operan al nivel de los pixels, y en una segunda capa los diagnósticos que operan al nivel de los segmentos. Una capa intermedia se encarga de agrupar los pixels con el fin de proporcionar una primera partición de la imagen para los diagnósticos que operan sobre segmentos.

Los diagnósticos utilizados en la primera capa de procesamiento se encargan de extraer en primer término características numéricas que luego se combinan para producir

Figura 3.13: Ejemplo imagen de entrada A.

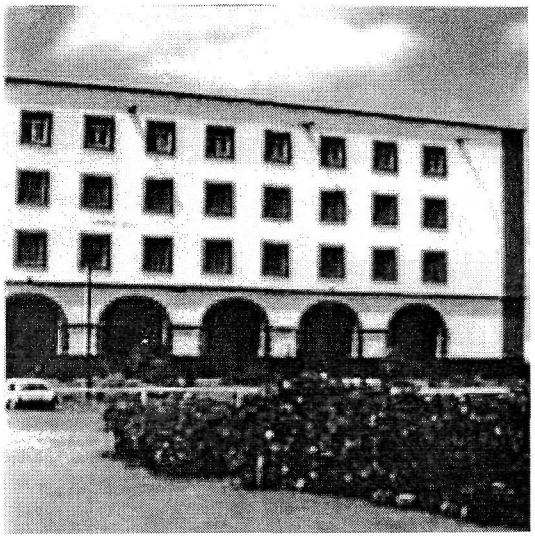

Figura 3.14: Ejemplo imagen de entrada B.

resultados de naturaleza simbólica. Dentro de los resultados numéricos se computan, entre otras, las siguientes características:

- Extracción de planos RGB.
- Extracción de componentes de color: I1, I2 e I3.
- Gradiente.
- Varianza.
- Imagen suavizada.

Los mapas numéricos son tomados por los diagnósticos de simbolización para generar mapas que indican el grado de pertenencia de cada pixel a una clase determinada. Algunos ejemplos son los siguientes:

- Alta varianza.
- Verde hierba.
- Verde árbol.
- Azul cielo.
- Sombra.

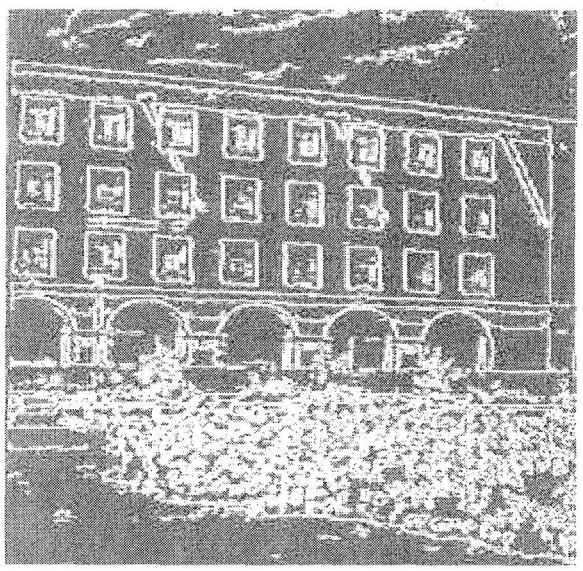

Figura 3.15: Mapa de diagnóstico alta varianza computado para la imagen B. El blanco indica un alto nivel de identificación con el prototipo de diagnóstico.

La imagen 3.15 presenta el diagnóstico simbólico "Alta varianza" extraído de la imagen B.

La segmentación de la imagen se realiza a partir de la combinación de los mapas generados por los niveles inferiores, por agrupación de zonas con propiedades homogéneas. Las figuras 3.16 y 3.17 muestran el resultado del proceso de segmentación aplicado sobre las imágenes A y B, respectivamente.

Los diagnósticos que operan en el nivel de segmentos toman como entrada los resultados simbólicos y los combinan para generar mapas de segmentos etiquetados. Algunos ejemplos de estos resultados son los siguientes:

- Cielo.
- Césped.
- Carretera.
- Ventana.
- Arbusto.
- Fachada.

En la figura 3.18 se resaltan los segmentos etiquetados como fachada sobre la imagen A, y en figura 3.19 los segmentos etiquetados como ventana sobre la imagen B.

Figura 3.16: Resultado de la segmentación de la imagen A.

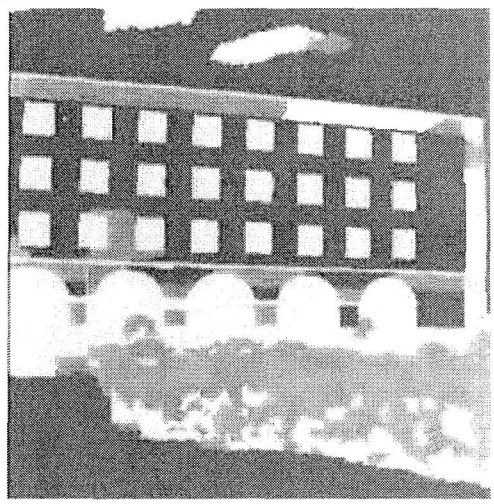

Figura 3.17: Resultado de la segmentación de la imagen B.

3.6. El Sistema en Ejecución

En el marco de este trabajo se ha desarrollado un prototipo de sistema ajustado a los conceptos estructurales y funcionales que se han propuesto. El prototipo admite la definición ordenada de diferentes tipos de objetos e implementaciones y su composición para dar lugar a tareas. Las tareas pueden agruparse en estados para recibir un tratamiento uniforme.

3.6.1. Comandos de tareas

El prototipo admite la definición de tareas agrupadas en estados o de forma individual. Durante la evolución del sistema, las tareas pueden verse afectadas por los siguientes comandos:

Registro: Una tarea se da de alta en el sistema. Esto supone por un lado un proceso de verificación de validez de la tarea y, por otro, una reserva de recursos para permitir la posterior ejecución de la misma.

Activación: Solicitud de ejecución de la tarea.

Suspensión: Petición para detener temporalmente la ejecución de la tarea.

Reactivación: Una tarea suspendida puede reactivarse cuando se den las circunstancias adecuadas.

Figura 3.18: Segmentos etiquetados como fachada en la imagen A.

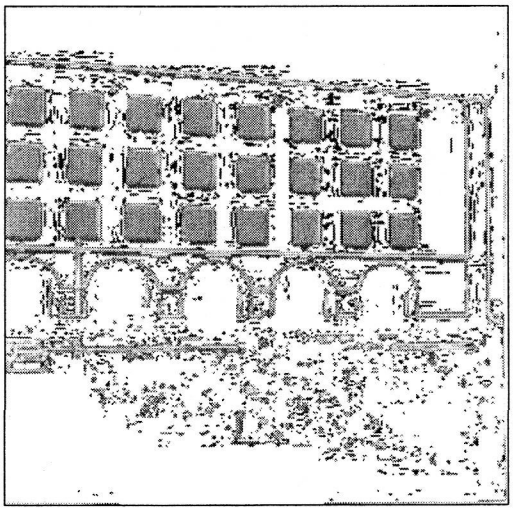

Figura 3.19: Segmentos etiquetados como ventanas en la imagen B.

Eliminación: Cuando la tarea ya no va a ejecutarse más, se liberan los recursos capturados por la misma y se da de baja.

Una tarea se registra en el sistema con un comando tipo "acción + diagnóstico" dirigido al supervisor. Este comando va acompañado de dos parámetros, que son el nivel de prioridad que se desea asignar a la tarea y la frecuencia de funcionamiento requerida. A partir de ese momento se inicia el proceso de asignación de módulos BU-TD-COM libres de un número máximo de módulos previamente activados y marcados como disponibles. Este esquema viene impuesto por el SCC de bajo nivel utilizado, pero no constituye limitación alguna puesto que los módulos libres no suponen más que una mínima sobrecarga sobre el sistema. El siguiente paso es la selección de implementaciones para que la tarea pueda ser ejecutada.

En respuesta a un comando de registro, el supervisor selecciona un módulo libre para el cómputo de la acción, le encarga el resto de la preparación de la tarea, y queda a la espera de recibir el mensaje de que la tarea está lista. El módulo de la acción debe entonces seleccionar una implementación concreta en base al contexto actual de operación, ordenar el cómputo del diagnóstico y de los actuadores, y esperar a recibir la confirmación correspondiente. Pueden darse dos situaciones, que dichos cómputos ya se estén realizando en el sistema para otra tarea o que se trate de elementos nuevos. En el primer caso simplemente hay que comunicar al módulo correspondiente que existe un nuevo destino para el resultado del procesamiento, en el caso del diagnóstico, o un nuevo origen de comandos, en el caso de los actuadores. En el segundo caso es necesario escoger un nuevo módulo para encargarle el cómputo, lo que a su vez supone esperar a

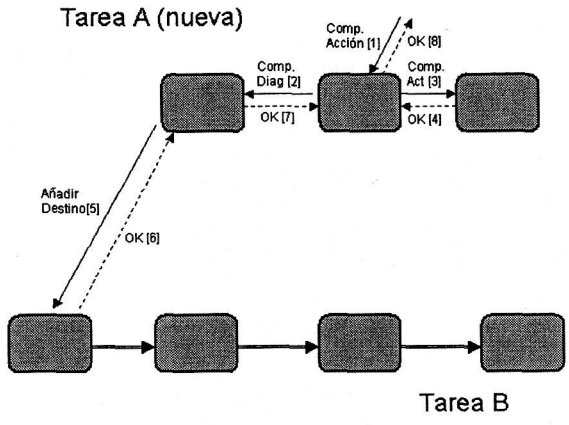

Figura 3.20: Ejemplo de secuencia de activación de una tarea.

que los módulos elegidos finalicen el proceso.

En los módulos de cómputo de diagnósticos, debe seleccionarse una implementación adecuada y comprobar los sensores y/o diagnósticos que se necesitan como datos de entrada. De nuevo puede ocurrir que dichos elementos de entrada estén o no en ejecución en el sistema. En el primer caso se solicita añadir nuevo destino y en el segundo se deben activar nuevos módulos.

Una vez completada la asignación de módulos a la tarea comienzan a enviarse los mensajes de confirmación, de los sensores a los diagnósticos, y de los actuadores y los diagnósticos a las acciones. Finalmente, y si no se ha producido ningún error, el módulo encargado de la acción envía la señal de tarea lista al supervisor, que la registra y la marca como preparada para ejecución. En caso contrario, se notifica el error al supervisor para que éste lo traslade al usuario. Las causas habituales de error son la solicitud de acciones o diagnósticos que no están recogidos en la base de conocimiento del sistema, y la especificación de valores no válidos para los parámetros de prioridad o frecuencia. En la figura 3.20, se muestran las señales implicadas en la activación de una tarea en la que todos los módulos a excepción del sensor han debido añadirse al sistema.

El sistema puede comandar la ejecución de tareas de forma individualizada o globalmente para un estado y todas las tareas y transiciones asociadas al mismo. La ejecución se inicia por los sensores, que empiezan a generar datos a la frecuencia comandada. A partir de ahí, las diferentes unidades de comunicaciones se encargan de suministrar los datos que va generando cada módulo productor a sus módulos consumi-

Algoritmo 1 Algoritmo de recálculo de periodos y prioridades por adición de tarea.

Leer nuevo periodo ($NewPer$) y nueva prioridad ($NewPrio$).
Calcular mínimo periodo actual ($MinPer$) y máxima prioridad ($MaxPrio$).
$CambioConf = falso$
if $MinPer > NewPer$ **then**
  Cambiar periodo módulo.
  $CambioConf = verdadero$
end if
if $MaxPrio < NewPrio$ **then**
  Cambiar prioridad módulo.
  $CambioConf = verdadero$
end if
if $(CambioConf == verdadero)\&(TipoMod! = SENSOR)$ **then**
  Enviar señal de cambio de configuración a las fuentes del módulo.
end if

dores con la frecuencia solicitada por cada uno de ellos. Simultáneamente, las transiciones son analizadas en los supervisores para determinar la evolución del sistema a nivel de estados.

3.6.2. Asignación de prioridades y frecuencias

Tanto los valores de prioridad como la frecuencia de funcionamiento asociados a la tarea se van propagando a través de todos los módulos afectados a lo largo de la fase de inicialización. Cuando los módulos son compartidos, debe tenerse en cuenta que los valores de prioridad y las frecuencias de las tareas involucradas pueden no coincidir, con lo que se hace necesario mantener un mecanismo de recálculo de los mismos cada vez que se produce una variación en la configuración del sistema. Así, un módulo productor con múltiples módulos consumidores posee una prioridad que es igual a la mayor de las que corresponden a sus destinos, mientras que su frecuencia de funcionamiento se ajusta a la más rápida de las demandadas. Además de esto, los módulos compartidos deben detectar cuándo dejan de demandarse sus servicios para finalizar su ejecución. Los algoritmos 1 y 2 muestran de forma resumida los pasos a seguir en caso de activación y eliminación de tareas, respectivamente.

3.6.3. Control de errores

Los errores físicos que pueden detectarse siguen un esquema de propagación jerárquico, de forma que siempre intentan ser resueltos por el supervisor situado al nivel

Algoritmo 2 Algoritmo de recálculo de periodos y prioridades por eliminación de tarea.

```
if Última conexión eliminada then
  Enviar señal de finalización a las fuentes del módulo.
else
  Leer periodo (OldPer) y prioridad (OldPrio) de tarea a eliminar T_i.
  Calcular mínimo periodo (MinPer) y máxima prioridad (MaxPrio) sin considerar
  T_i.
  CambioConf = falso
  if MinPer > OldPer then
    Cambiar periodo módulo.
    CambioConf = verdadero
  end if
  if MaxPrio < OldPrio then
    Cambiar prioridad módulo.
    CambioConf = verdadero
  end if
  if (CambioConf == verdadero)&(TipoMod! = SENSOR) then
    Enviar señal de cambio de configuración a las fuentes del módulo.
  end if
end if
```

de generación del error y, si no es posible la resolución en ese punto, se traslada al supervisor de nivel superior. Este proceso es posible gracias a la declaración de los tipos de error dentro del sistema. Algunos ejemplos de tipo de error, organizados por niveles, son los siguientes:

- Nivel módulo:
  - Error de comunicación con dispositivo físico.
  - Error en la operación del dispositivo físico.
  - Error de comunicación con unidad BU.
  - Error de comunicación con unidad COM.
  - Fallo de memoria.
  - ...
- Nivel tarea:
  - Error irrecuperable en sensor.
  - Error irrecuperable en diagnóstico.
  - Actuador no responde.

- ...

- Nivel estado:
  - Error irrecuperable en tarea.
  - Supervisor de tarea no responde.
  - ...

- Nivel sistema:
  - Error irrecuperable en supervisor.
  - Supervisor de estado no responde.
  - ...

Los errores detectados a nivel de módulo suponen una transición al estado "ERROR" del mismo. Se aplican entonces las acciones de recuperación suministradas por el usuario, si existen, o las acciones por defecto. Puesto que los errores a este nivel están relacionados normalmente con fallos hardware de dispositivos, las acciones por defecto consisten en tratar de reinicializar el dispositivo un cierto número de veces. Si los intentos de recuperación fracasan, se traslada el error al nivel superior indicando que no ha podido resolverse a nivel local. En el nivel de tarea se puede intentar entonces reemplazar el sensor por uno equivalente, o bien el diagnóstico por otro que suministre la misma salida. Si no puede recuperarse la operatividad de la tarea, se notifica al nivel superior. En el nivel de estado se decide entonces si la tarea es prescindible o si es necesario trasladar el error al nivel superior para una replanificación de la misión del sistema.

Capítulo 4

Adaptación Computacional y Control

En este capítulo se profundiza en uno de los aspectos fundamentales en el rendimiento de un sistema percepto-efector, como es su capacidad para ofrecer un comportamiento robusto en un entorno cambiante. A partir de las estimaciones de la carga y los recursos de adaptación disponibles se plantean diferentes técnicas y políticas de control. Los dos últimos apartados están dedicados al multiprocesamiento y el aprendizaje.

4.1. Introducción

Uno de los aspectos que se considera fundamental en un sistema percepto-efector es la capacidad para garantizar un comportamiento robusto. Esto implica que en condiciones del entorno cambiantes o incluso desfavorables, la integridad del sistema en primer lugar, y su eficacia en segundo, deben disponer de elementos de salvaguarda. Los mecanismos de adaptación operan sobre el sistema en tiempo de ejecución a fin de conseguir un comportamiento más adecuado en función de una cierta señal de referencia externa. La definición de ese "comportamiento adecuado" deriva de los diferentes problemas a los que debe hacer frente el sistema como son gestionar recursos escasos, conseguir un comportamiento estable, reaccionar ante fallos, reducir el tiempo de respuesta, adaptarse a los cambios del entorno, etc.

La concepción y el diseño del sistema propuesto permite hablar de adaptación a dos niveles: alto nivel y bajo nivel. En el nivel más alto, la organización de objetos por tipos permite seleccionar para la misma tarea diferentes implementaciones en función de las condiciones de operación. Algunos ejemplos de adaptación a alto nivel en sistemas

percepto-efectores serían los siguientes:

- En el caso de un sistema percepto-efector visual, al cambiar las condiciones de iluminación, el cómputo de algunos diagnósticos como el color pueden requerir el uso de algoritmos diferentes.
- Para un sistema percepto-efector visual que admita el trabajo tanto en entornos de interior como de exterior, pueden existir asimismo versiones de código especializadas en cada situación.
- El caso de un sistema robótico móvil con cambios en las condiciones de navegación, pasando de corredores a espacios abiertos.

En el nivel más bajo, la adaptación se produce manteniendo el código, es decir, conservando las implementaciones ya seleccionadas para cada objeto (sensor, diagnóstico, acción o efector), pero modificando sus demandas computacionales.

En este capítulo, nos centraremos en el estudio de los recursos de adaptación de bajo nivel, analizando como caso particular la acomodación a los recursos computacionales disponibles.

4.2. Adaptación de la Carga

Nuestro sistema se caracteriza por la ejecución simultánea de múltiples tareas con diferentes frecuencias de funcionamiento y niveles de prioridad distribuidas sobre diferentes máquinas. Cada una de esas tareas, como se detalló en el capítulo anterior, está formada por la interconexión de objetos de tipo sensor, diagnóstico, acción y actuador. A su vez, cada objeto es implementado por un módulo BU-TD-COM, desempeñando la unidad TD de la acción el papel de controlador de la tarea.

En un entorno multitarea como el descrito se produce siempre una competencia por recursos comunes que, de no ser gestionada convenientemente, puede conducir a una pérdida de rendimiento en el sistema y, en caso extremo, a un bloqueo del mismo. Ante una situación de escasez de recursos computacionales, el sistema debe reaccionar de modo que las tareas actualmente en ejecución reduzcan su rendimiento en función de su prioridad. Es deseable que esta reducción sea homogénea para evitar desequilibrios dentro del sistema, esto es, situaciones de acaparamiento de recursos por parte de unas tareas frente a otras. Asimismo, se precisan acciones de recuperación para el caso en que las condiciones de funcionamiento mejoren ante un eventual incremento en los recursos

disponibles. Como en el caso anterior, también aquí es deseable que el progreso de la acción sea homogéneo.

El objetivo es plantear un esquema de adaptación para un contexto de tiempo real blando. Para garantizar una mayor aplicabilidad, se busca que los requerimientos impuestos al hardware y sistema operativo de soporte sean lo menos restrictivos posible. Se evitará, por lo tanto, la utilización de recursos privativos de sistemas operativos de tiempo real o planificadores (*schedulers*) especializados.

Por otra parte, dentro de la estrategia de control, es interesante arbitrar políticas tanto locales como globales. Las primeras permiten mejorar los tiempos de respuesta y reducir los overheads, en tanto que las políticas globales, ofrecen una mejor selección de los objetivos del control. En función de qué factores se consideren más relevantes se podrá optar por unas u otras.

4.2.1. Bucles de control

En el marco de trabajo definido son planteables diferentes bucles de control con el objeto de alcanzar un comportamiento adaptativo del sistema en tiempo de ejecución. Cada uno de estos bucles se corresponde con las siguientes señales de referencia básicas:

Timeouts: Verificación de la frecuencia de funcionamiento deseada para cada tarea. Supone, en principio, un bucle de control específico para cada tarea.

Nivel de carga deseado: Valor de referencia para la carga global del sistema. Constituye un bucle de control global para todo el sistema.

Homogeneidad de la carga: Se persigue una distribución temporal uniforme de la carga del sistema.

Tiempo de respuesta: Definido como el intervalo que transcurre entre la aparición de un estímulo en los sensores de entrada y la generación de la salida correspondiente en los actuadores finales de la tarea.

Los controladores de cada uno de esos bucles son siempre las unidades TD, bien como supervisores de tarea, de estado, o de sistema. La unidad TD es la encargada de la monitorización y el control de la ejecución en cada módulo a nivel local. A partir de la evolución del procesamiento en la unidad BU, la unidad TD de control verifica los tiempos invertidos y toma acciones de control directas o bien notifica los eventos detectados a los niveles superiores. A nivel de tarea el TD de la acción puede asimismo

coordinar acciones de control que afectan a los módulos pertenecientes a dicha tarea. A nivel superior son los TD de los supervisores de estado y de sistema los que son responsables de las acciones de control que exceden el ámbito de la tarea.

4.2.2. Modelado de la carga

Antes de detallar las posibles acciones y políticas de control aplicables al sistema, se introducirá un modelo simple para la carga computacional. Este modelo se utilizará tanto como referencia en la descripción de los recursos de control como en la propia implementación de los mismos.

Asumiendo un entorno monoprocesador, la carga es una variable que toma el valor cero cuando la CPU está ociosa y uno cuando está ocupada. Esta medida instantánea no es de interés para los esquemas de control que se desean plantear, por lo que se buscará una medida promediada en un intervalo de tiempo determinado. Por consiguiente, en lo sucesivo, las referencias a la carga del sistema se entenderán siempre como carga media integrada.

La carga a la que está sometido un sistema puede determinarse, de manera simplificada, a partir de las cargas aportadas por cada una de las tareas y módulos que se ejecutan en él. Definiremos las siguientes variables:

nm: Número total de módulos en ejecución.

Tt_i: Periodo de funcionamiento de la tarea t_i.

Tm_i: Periodo de funcionamiento del módulo m_i.

Tem_i: Tiempo de ejecución, medido como el tiempo transcurrido desde que el módulo i inició la ejecución hasta que ésta finaliza.

Tpm_i: Tiempo de procesamiento del módulo m_i. Coincide con Tem_i cuando la CPU está ocupada únicamente por dicho módulo. Normalmente $Tem_i > Tpm_i$.

La carga media evaluada en un intervalo Tm_i que un módulo m_i induce en la CPU (Cm_i) puede aproximarse por

$$\widehat{Cm_i} = \frac{Tpm_i}{Tm_i}$$

donde el periodo del módulo se calcula como el mínimo de los periodos asociados a las tareas en las que está incluido.

$$\widehat{Tm_i} = \text{mín}\{Tt_j / m_i \in t_j\}$$

La carga media de una tarea (Ct_i) puede estimarse entonces como

$$\widehat{Ct_i} = \sum \{\widehat{Cm_j} / m_j \in t_i\}$$

Si se trata de una tarea no periódica, aunque repetitiva, la carga puede estimarse para el peor caso asimilando el tiempo de respuesta como periodo de repetición. En el caso de tareas esporádicas, simplemente se considera que no representan carga para el sistema o, alternativamente, se reservará un cierto porcentaje de uso de la CPU como se indicará más adelante.

Debido a que determinados módulos pueden estar compartidos entre diferentes tareas, no es posible estimar la carga media total del sistema a partir de las cargas aportadas por cada tarea. En cambio, la carga media global del conjunto (C) se obtiene como resultado de la acumulación de las cargas medias aportadas por todos los módulos que se ejecutan en el sistema sobre un periodo determinado. Para que la medida sea estable debe tomarse como periodo de evaluación al menos el mayor de los periodos de las tareas del sistema, siendo lo ideal considerar ese periodo como el mínimo común múltiplo de todos los periodos T_{MCM} ó T^*. Se tiene entonces

$$\widehat{Cm_i^*} = \frac{Tpm_i^*}{T^*}$$

donde Tpm_i^* es el tiempo de procesamiento consumido por el módulo m_i en el intervalo T^*.

Considerando que todas las tareas respetan sus periodos de funcionamiento resulta $Tpm_i^* = n_i \times Tpm_i$, con $n_i = T^*/Tm_i$. Finalmente se llega a

$$\hat{C} = \sum_{i=1}^{n} \widehat{Cm_i^*}$$

donde cada aportación de carga está evaluada sobre un intervalo equivalente al mínimo común múltiplo de todos los periodos de repetición de los módulos considerados.

La aproximación presentada es conservadora, puesto que no se tienen en cuenta los costes adicionales que surgen cuando el número de tareas crece, como son los debidos a la conmutación de contexto o la distribución de los datos a los diferentes destinos. Tampoco se tiene en cuenta la carga generada por los módulos supervisores que, aunque

siempre será reducida, existe. Según esto, se tiene $C > \hat{C}$.

4.2.3. Medida de la carga real

La estimación de la carga real del sistema es un factor fundamental a la hora de aplicar los mecanismos de adaptación. Permite cerrar los diferentes bucles de control generando las señales de realimentación necesarias para su comparación con las correspondientes señales de referencia.

En este sistema se consideran dos tipos de medidas básicas: de carácter local y de carácter global. Las medidas locales corresponden a estimaciones internas a las tareas y los módulos, mientras que las medidas globales suministran estimaciones generales para todo el sistema.

Medidas locales

Como estimación local de la carga se toma la relación que existe entre el tiempo de ejecución de cada módulo (Tem_i) y el tiempo máximo disponible, esto es, su periodo (Tm_i).

$$\widehat{Clm_i} = \frac{Tem_i}{Tm_i}$$

Un valor cercano a la unidad indica un sistema sobrecargado, mientras que valores cercanos a cero reflejan un sistema con gran cantidad de recursos disponibles. En el caso extremo, un módulo puede detectar que no ha sido capaz de respetar la frecuencia de operación impuesta cuando el tiempo de ejecución supera al periodo. Se produce entonces una violación temporal que indica que las capacidades disponibles en el sistema no son suficientes para responder a la carga computacional demandada, situación que debe intentar solventarse lo antes posible.

Como complemento a esta medida, puede obtenerse la aportación del módulo a la carga total del sistema. Esta cantidad es calculable igualmente en tiempo de ejecución a partir del tiempo consumido en CPU como

$$\widehat{ACl m_i} = \frac{Tpm_i}{Tm_i} \tag{4.1}$$

La comparación entre la carga global estimada y la aportación de carga da una idea del nivel de ocupación de la CPU durante la ejecución del módulo. De hecho, el

número medio de módulos que comparten la CPU durante la ejecución del módulo puede aproximarse por

$$\widehat{NumMod}_i = \frac{Clm_i}{AClm_i}$$

Hay que tener en cuenta que no todos los módulos son adecuados para estimar la carga del sistema. Los módulos con mayor contribución de carga y menor prioridad medirán valores más altos que los de baja carga y alta prioridad. Es por ello que la medida de carga estimada localmente se emplea fundamentalmente para la detección de violaciones temporales. Aún así, la medida de los tiempos de ejecución en un entorno multitarea es generalmente ruidosa, por lo que resulta interesante disponer de algún método para relajar las condiciones en las que se miden los timeouts dentro del sistema. En esta línea, es posible definir un cierto umbral de tolerancia de modo que una simple violación aislada no dispare las acciones de control, siendo necesaria para que esto se produzca la acumulación de un cierto número de evidencias en un intervalo de tiempo determinado. Ese umbral se puede especificar tanto a nivel global del sistema como asociado a cada tarea.

Otra medida que puede estimarse localmente es la homogeneidad de la distribución temporal de la carga. Para ello se comparan los tiempos empleados por cada módulo en completar el procesamiento sobre diferentes intervalos de tiempo. Si el tiempo de procesamiento de un módulo es pequeño en comparación al resto, la medida puede establecerse a partir de diferentes ejecuciones. Si el tiempo de procesamiento es grande, la estimación debe hacerse analizando el progreso de la ejecución en el tiempo. En este segundo caso se puede tomar como medida de homogeneidad la simetría del procesamiento, definida como la diferencia entre el tiempo que tarda un módulo (principalmente los de cómputo de diagnósticos) en procesar la primera y la segunda mitad de sus datos de entrada. Sean las siguientes cantidades definidas para un módulo m_i:

Tem_1_i: Tiempo de ejecución para la primera mitad de los datos.

Tem_2_i: Tiempo de ejecución para la segunda mitad de los datos.

Tem_i: Tiempo de ejecución total ($Tem\_1_i + Tem\_2_i$).

La simetría del procesamiento en el módulo m_i es una cantidad comprendida entre +1 y -1 que viene dada por la siguiente expresión:

$$SimP_i = \frac{Tem\_1_i - Tem\_2_i}{Tem_i} \tag{4.2}$$

Un valor de 0 indica que la carga de la CPU durante la ejecución del módulo m_i ha permanecido estable.

Medidas globales

Como medida global de la carga (C_{tt}) se emplea una tarea de test a la que se le asigna la prioridad mínima y una frecuencia de operación elevada (F_{tt}). Comparando el número de ejecuciones (NEx_{tt}) de dicha tarea en un intervalo de tiempo dado ($IntAn$) con el máximo esperado ($IntAn \times F_{tt}$) se obtiene una aproximación al nivel de carga del sistema. En este caso, la medida se formula como sigue:

$$C_{tt} = 1 - \frac{NEx_{tt}}{IntAn \times F_{tt}}$$

Los valores de carga resultantes de esta medida han sido comparados con los suministrados por la herramienta de monitorización del rendimiento del sistema en Windows, obteniéndose comportamientos similares. Las ventajas del esquema propuesto son las posibilidades de parametrización para adecuarlo a los diferentes entornos de ejecución y su independencia del sistema operativo de soporte.

El problema fundamental de las estimaciones de carga es el compromiso que debe alcanzarse entre reactividad y fiabilidad de la medida. Un intervalo de análisis de la carga corto proporciona tiempos de respuesta reducidos aunque con medidas poco fiables. Así, se corre el riesgo de tomar decisiones de control basadas en valores espúreos o ruidosos, que inducen oscilaciones en el sistema. En cambio, tomando intervalos de análisis mayores se consiguen medidas más fiables, a costa de un mayor tiempo de respuesta. A su vez, la fiabilidad de las medidas está condicionada a la frecuencia de operación de las diferentes tareas del sistema, lo que hace imprescindible condicionar la selección del periodo de análisis a este dato.

Otro aspecto fundamental es la necesidad de disponer de algún mecanismo de estudio de la estabilidad del sistema. En la implementación se ha optado por mantener un buffer de n medidas ($\{C_{tt}[n]\}$) que almacenen un histórico del funcionamiento del sistema durante un periodo determinado. Este conjunto de datos permite determinar cuándo el sistema se encuentra estabilizado analizando la variabilidad de las medidas. Una alternativa que se ha elegido para la implementación del prototipo es medir el rango de variación de la siguiente forma:

$$\%VAR\_C_{tt} = \frac{\text{máx}\{C_{tt}[n]\} - \text{mín}\{C_{tt}[n]\}}{media\{C_{tt}[n]\}} \times 100$$

4.2.4. Acciones de control

Como resultado de los diferentes bucles de control, pueden detectarse errores que una vez analizados por los controladores se traducen en acciones de control. Las acciones de control disponibles para modificar la carga computacional de las tareas que se ejecutan en el sistema son las siguientes:

- Degradación/Promoción en calidad.
- Degradación/Promoción en frecuencia.
- Desplazamiento temporal.
- Suspensión/Reactivación.

El sistema operativo sobre el que se ejecutan las tareas debe suministrar un planificador que utilice Round-Robin con colas de prioridad (tipo Linux, Windows NT/2000), dado que las estrategias de control planteadas se apoyan en este supuesto para su correcto funcionamiento.

Control en calidad

Dentro de las acciones de control que afectan a la calidad de los resultados producidos tenemos la variación del tamaño o la resolución de los datos de entrada y los cambios del nivel de precisión en el cómputo de los resultados. La primera de las acciones se aplica principalmente sobre los sensores que suministran volúmenes de datos elevados, como es el caso de las cámaras. En esos casos, la variación del tamaño de los datos de entrada tiene un efecto sustancial sobre la carga computacional del sistema, pues afecta a todos los módulos que se encuentran dentro del área de influencia del flujo de datos que se origina en ese punto. La degradación del sistema consistirá en la reducción del tamaño o resolución de los datos de entrada, mientras que con la promoción se emplearán tamaños o resoluciones de datos mayores.

Desde el punto de vista del modelo presentado para la carga del sistema, la reducción del tamaño de los datos de entrada supone un menor tiempo de procesamiento para todos los módulos que dependan de ese sensor y, por tanto, una menor contribución a la carga global. La relación que exista entre el tiempo de procesamiento y el tamaño de los datos de entrada (lineal, exponencial) determinará el efecto alcanzado.

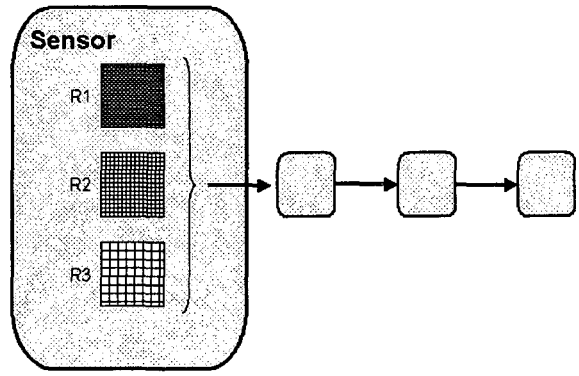

Figura 4.1: Ejemplo de control en calidad - Resolución.

Los algoritmos de procesamiento, a su vez, pueden proporcionar resultados con distinto nivel de precisión, lo que también afecta a los recursos computacionales demandados. En este caso, son los módulos de cómputo de diagnósticos los que concentran este tipo de acciones de control, al ser los puntos de mayor consumo de tiempo de CPU dentro del sistema. La promoción consistirá en emplear mayores niveles de precisión en los algoritmos, mientras que con la degradación se forzará la utilización de niveles menores. En el modelo de carga esta acción se traduce nuevamente en una variación del tiempo de procesamiento, aunque en este caso localizado en un módulo concreto cuyo efecto se trasladará al sistema completo. Se trata, pues, de una acción de control orientada a módulos.

En base a lo anterior, la aplicación de las acciones de control en calidad está condicionada a que las diferentes implementaciones de sensores, diagnósticos, etc., ofrezcan la posibilidad de trabajar con distintas resoluciones y niveles de precisión.

Formalmente, estaríamos considerando un esquema de funcionamiento del tipo procesamiento *anytime*. Este sistema se basa en la utilización de algoritmos capaces de suministrar resultados en diferentes instantes de tiempo, de forma que la calidad de los mismos crece, hasta un valor máximo, con el tiempo disponible para el procesamiento. Dentro de este tipo de algoritmos se puede hablar de dos aproximaciones [Zilberstein, 1996]:

Algoritmos *anytime* interrumpibles: Suministran un resultado en cualquier instante de tiempo, siempre por encima de un valor mínimo, con calidad creciente.

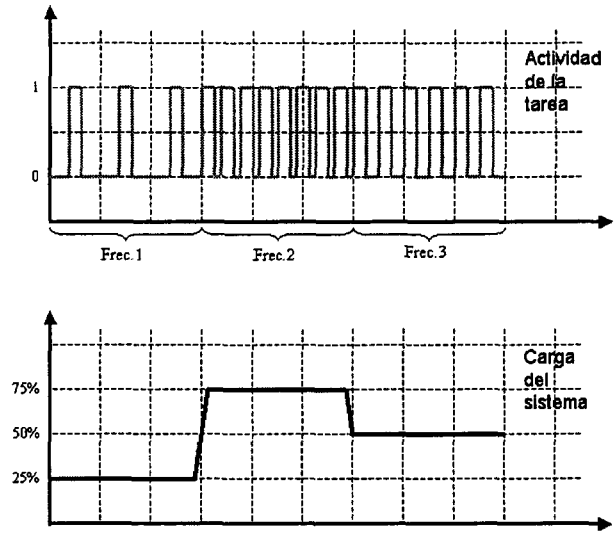

Figura 4.2: Ejemplo de cronograma y nivel de carga en el control en frecuencia.

Algoritmos *anytime* de contrato: Admiten como parámetro de entrada el tiempo disponible para el procesamiento. Transcurrido ese tiempo de ejecución, se garantiza la entrega de un resultado con una calidad determinada. Es posible construir esquemas interrumpibles a partir de algoritmos de contrato [Zilberstein et al., 1999].

Nuestra propuesta de sistema computacional adaptativo se basa en algoritmos *anytime* tipo contrato, donde el tiempo disponible es aquél que permita a la tarea verificar su frecuencia de funcionamiento.

Control en frecuencia

Las acciones de control que afectan a la frecuencia de funcionamiento consisten en la alteración del periodo de las tareas, lo que implica una variación en la carga computacional demandada por las mismas (ver figura 4.2). En general, esta acción debe ser usada con precaución, pues puede comprometer la integridad del sistema; por ejemplo, en aplicaciones de detección de obstáculos o en lazos de control que impliquen movimiento.

El control en frecuencia es una acción de control orientada a tarea, lo que significa que se aplica simultáneamente a todos los módulos pertenecientes a la tarea afectada. Desde el punto de vista de la implementación, sin embargo, la promoción/degradación en frecuencia es más simple que las acciones en calidad, no imponiendo especiales requerimientos a las tareas que vayan a correr sobre el sistema. Simplemente se incrementa

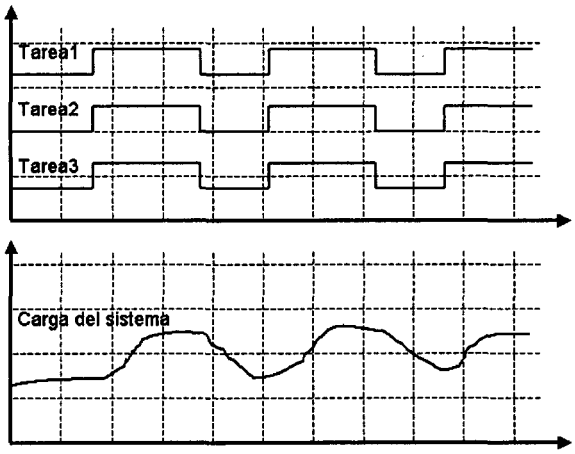

Figura 4.3: Ejemplo de la influencia del offset - Sistema desequilibrado.

el periodo de funcionamiento para degradar y se decrementa hasta un valor mínimo (normalmente el especificado por el usuario al lanzar la tarea) para promocionar.

Considerando el modelo para la carga del sistema, el control de la frecuencia implica una variación en el periodo de la tarea (Tt_i). Con las simplificaciones establecidas, el efecto sobre la carga es proporcional a la variación introducida.

Desplazamiento temporal

Otra posibilidad estudiada ha sido el análisis de la distribución temporal de la carga en el sistema. Se ha comprobado como esta situación puede provocar la aparición de zonas de congestión, cuando coinciden en el tiempo múltiples tareas, acompañadas de intervalos de baja ocupación de la CPU. La solución al problema consiste en la detección de dichas zonas de congestión y en intentar reducirlas. El desplazamiento en el tiempo de algunas tareas puede suavizar el perfil de carga, manteniendo unos niveles más homogéneos a lo largo del tiempo de ejecución. Este aspecto es interesante, puesto que las violaciones del periodo de funcionamiento de las tareas pueden no deberse a una insuficiencia de recursos computacionales, sino a una distribución de la demanda poco equilibrada.

Una ventaja adicional de esta acción de control es la reducción de los retardos debidos a la conmutación de tareas en la CPU. Este hecho, aunque en pequeña escala, contribuye también a decrementar la carga global del sistema.

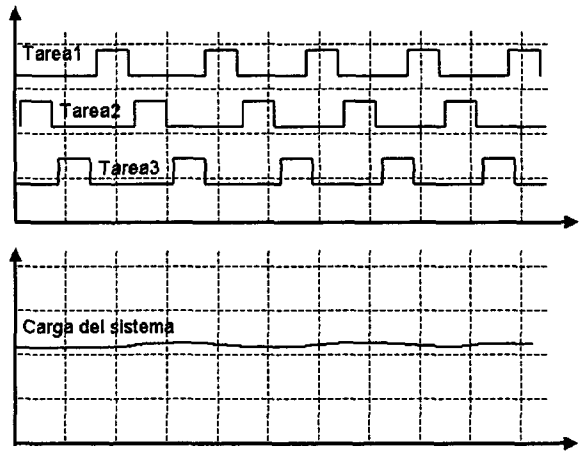

Figura 4.4: Ejemplo de la influencia del offset - Sistema controlado.

Suspensión/Reactivación

Finalmente, y como recurso extremo, el sistema puede decidir que la configuración actual de tareas no es sostenible, con lo que la única solución posible es la suspensión o sustitución de algunas de las tareas demandadas al sistema. En caso de ser suspendidas, dichas tareas podrán ser retomadas cuando las condiciones lo permitan.

Lógicamente, la suspensión de tareas sólo es efectiva cuando se aplica a tareas que están en ejecución. En un sistema sobrecargado las tareas con menor nivel de prioridad no tienen ocasión de ejecutarse, por lo que no tiene sentido aplicarles esta acción de control.

4.2.5. Caracterización de la adaptación computacional

La evaluación de la calidad de la adaptación de los sistemas puede realizarse a través de diferentes medidas como son su flexibilidad y su progresividad. La primera de las medidas hace referencia a la variación de carga que sobre el sistema supone ir desde el nivel de máxima degradación hasta el de funcionamiento a pleno rendimiento con calidad máxima. La medida de la flexibilidad tendría la forma siguiente:

$$F = \frac{C_{max} - C_{min}}{100}$$

donde C_{max} es la carga para la calidad máxima y C_{min} la carga correspondiente a la

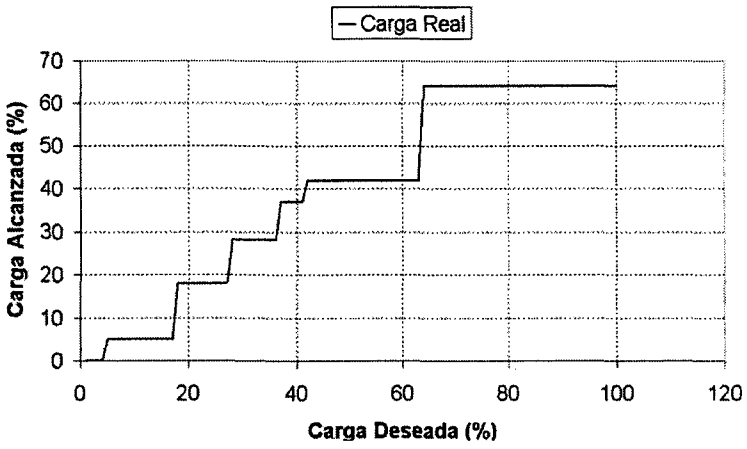

Figura 4.5: Relación entre la carga deseada y la carga correspondiente a los diferentes niveles de calidad del sistema.

calidad mínima.

Los efectos de la cuantización en los niveles de calidad se traducen en la imposibilidad de alcanzar todos los niveles posibles de carga en el sistema. En consecuencia, se produce un agrupamiento de niveles de tiempos de procesamiento disponibles o carga deseada que se mapean sobre un único valor de tiempo asignado o carga resultante. Asumiendo que el sistema de control es tal que se genera siempre una carga menor o igual a la deseada, se obtiene una gráfica escalonada como la del ejemplo mostrado en la figura 4.5. Si se define el grado de utilización de recursos como el cociente entre la carga/tiempo asignado y la carga/tiempo disponible puede trazarse una curva sobre todo el recorrido del sistema. La figura 4.6 muestra esta curva para un sistema sencillo con 5 niveles de calidad distribuidos desde el 10 hasta el 90 % de carga.

La progresividad mide la relación que existe entre el tiempo disponible y el tiempo asignado, evaluada sobre un intervalo de carga determinado. Una posible medida consiste en evaluar el área bajo la curva de utilización y compararla con el valor máximo alcanzable. Un sistema con perfiles continuos poseerá una progresividad con valor próximo a la unidad, mientras que un sistema con pocos niveles de calidad disponibles tomará valores cercanos a cero. La fórmula de cálculo es la siguiente:

$$P = \frac{\text{Área bajo la curva}}{\text{Área máxima}}$$

Podemos buscar una expresión simplificada para el caso en que se tengan n ni-

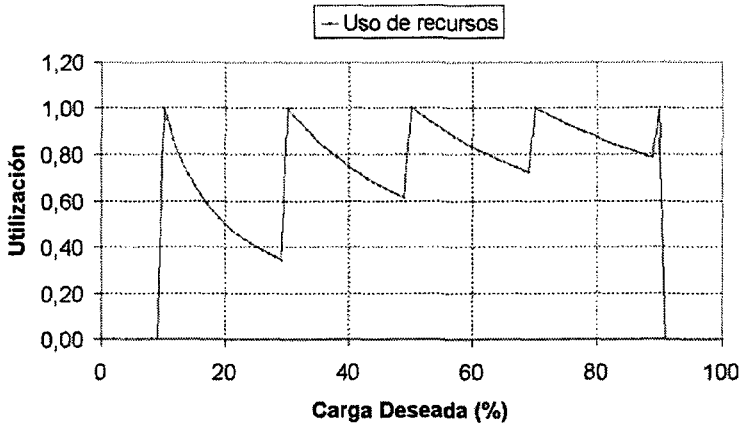

Figura 4.6: Efecto de la cuantización en el uso de los recursos disponibles.

veles de calidad distribuidos uniformemente (en el caso general no tiene porqué ser así). Tenemos entonces n intervalos de carga de ancho constante ΔC y extremos representados por los valores de carga c_i. El área puede aproximarse por una sucesión de trapecios de la forma

$$A \backsimeq \Delta C \sum_{i=1}^{n} \frac{1 + c_{i-1}/c_i}{2}$$

En el ejemplo de la gráfica de la figura 4.6 el área bajo la curva es de 0.77 unidades, mientras que la fórmula precedente da un valor de 0.8. En general esta aproximación da un buen resultado siempre que los niveles no se encuentren excesivamente dispersos. Como $c_i = i \times \Delta C$, queda

$$A \backsimeq n\Delta C - \frac{\Delta C}{2} \sum_{i=1}^{n} \frac{1}{i}$$

La progresividad es el cociente entre este área y el área máxima ($n\Delta C$).

$$P = \frac{A}{A_{max}} = 1 - \frac{1}{2n} \sum_{i=1}^{n} \frac{1}{i}$$

El producto de las dos medidas nos da un índice de adaptabilidad del sistema $IA = F \times P$. En el ejemplo de la gráfica 4.6, la flexibilidad toma un valor de 0.8, la progresividad vale 0.77, y el índice de adaptabilidad estaría en torno a 0.62 unidades.

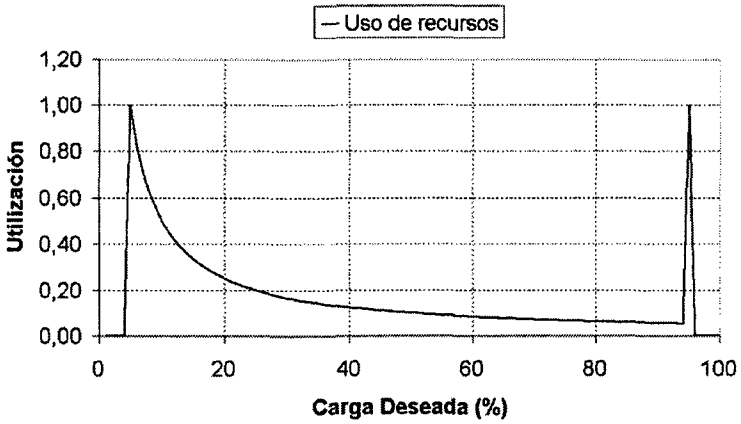

Figura 4.7: Sistema con buena flexibilidad y mala progresividad. El índice de adaptabilidad resultante es 0.25.

Algunos ejemplos de situaciones extremas son las que se ilustran en las figuras 4.7 y 4.8. En el primer caso se muestra la gráfica de un sistema con una buena flexibilidad pero una baja progresividad. La figura 4.8, por el contrario, muestra un sistema con una alta progresividad pero muy poco flexible. En ambos casos, las fórmulas dan como resultado un valor para el índice de adaptabilidad bajo: 0.25 y 0.2, respectivamente.

4.2.6. Calibración del sistema

Dentro de los planteamientos generales del problema de la adaptación de la carga podemos hablar de dos situaciones claramente diferenciadas:

- Sistema calibrado.
- Sistema no calibrado.

En el primer caso, los perfiles de rendimiento de las diferentes implementaciones están calibrados, por lo que es posible estimar con antelación la relación que existe entre el tiempo de procesamiento asignado y la calidad de los resultados obtenidos. En el segundo caso, simplemente se dispone de la información relativa a las posibilidades de reducción del coste computacional de cada implementación, pero no su repercusión sobre la carga del sistema. En cada una de las situaciones se plantearán esquemas de control que intentan hacer uso de la información existente. Otro problema interesante es el que

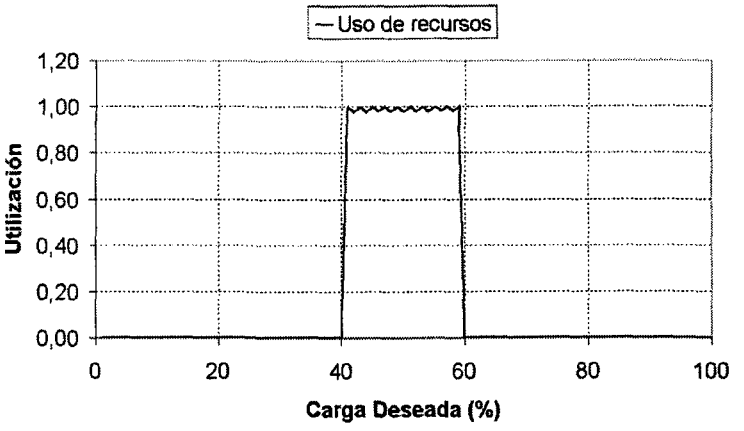

Figura 4.8: Sistema con buena progresividad y mala flexibilidad. El índice de adaptabilidad está en torno a 0.2.

se presenta cuando, partiendo de un sistema no calibrado, se va obteniendo información en tiempo de ejecución que permite realizar una calibración online o autocalibración.

Describiremos en este capítulo en primer lugar el caso de los sistemas no calibrados, para posteriormente pasar a los sistemas calibrados. Finalmente se plantea la forma de abordar la transición de un sistema no calibrado a otro calibrado.

4.3. Adaptación Computacional en Sistemas no Calibrados

En un sistema no calibrado, no se dispone de información detallada acerca de la relación que existe entre la carga y el nivel de calidad de cada módulo. Simplemente se conocen para cada módulo sus posibilidades de degradación. La ordenación de la degradación dentro del sistema se basa en asumir un comportamiento homogéneo entre diferentes módulos situados en niveles de degradación iguales. Esto es, se supone que a saltos iguales en los niveles de degradación de los diferentes módulos se originan variaciones similares en la contribución a la carga total del sistema.

4.3.1. Niveles de degradación

Como ya se ha expuesto con anterioridad, resulta indispensable arbitrar mecanismos que eviten situaciones de monopolio o acumulación de recursos por parte de ciertas tareas frente al resto. Por ello se ha diseñado un esquema de organización de los niveles de degradación/promoción de cada módulo en el sistema en base a umbrales de homogeneidad utilizados tanto a nivel local como global. Dichos umbrales consisten en dos límites, inferior y superior, que indican cuáles deben ser los niveles de degradación mínimos y máximos que pueden presentar en un momento determinado los módulos que se encuentran en ejecución en el sistema. El uso de estos umbrales se detallará más adelante.

Cada alternativa de implementación para los diferentes objetos dentro del sistema es declarada con los niveles de degradación disponibles, los cuales hacen referencia a las posibilidades de reducción de la carga computacional existentes en cada caso. Así, para el caso de la degradación en calidad, un sensor se registrará con los diferentes tamaños de datos de salida que puede suministrar, un diagnóstico con sus niveles de precisión, etc. Dentro del sistema se definen un conjunto de niveles de operación estándar, de forma que todos los módulos son comparables entre sí en base a una escala común.

Hay que tener en cuenta que en un sistema real no cabe esperar un número de niveles de calidad elevado para las diferentes implementaciones. Esto hace que las acciones de control provoquen un comportamiento a saltos discretos, por lo que no todos los niveles de carga van a ser alcanzables.

4.3.2. Políticas de control

Partiendo de la base de las medidas de carga explicadas anteriormente, el sistema debe hacer operativas políticas de control que coordinen las respuestas del sistema ante las diferentes señales de entrada posibles. Describiremos a continuación la implementación de una política de control que combina las acciones de regulación disponibles en el sistema.

Los eventos considerados como causas de la activación de las distintas acciones de control son los siguientes:

- Aparición de timeouts dentro del sistema.
- Diferencias entre la carga deseada y la carga actual.

- Distribución no homogénea de la carga.

Los bucles de control definidos hacen referencia al control de las violaciones temporales, el mantenimiento de un nivel de carga predeterminado y la estabilización del sistema.

Criterios de selección de candidatos

La eficacia de la adaptación depende en gran medida de los criterios que se apliquen a la hora de seleccionar alguna tarea o módulo candidato para la promoción/degradación. Dentro de las acciones de control en calidad, la reducción del tamaño de los datos tiene un efecto más rápido, aunque también más violento, sobre la carga del sistema, mientras que con el uso de niveles de precisión inferiores se produce una acción de control más paulatina. En el caso de la promoción/degradación de diagnósticos, el efecto depende además de que la contribución de ese módulo a la carga del sistema sea significativa, esto es, que presente un tiempo de procesamiento elevado en relación con su periodo.

La topología afecta igualmente a la magnitud de las correcciones, de modo que un sensor cuyos datos de salida se suministran a módulos de cómputo de diagnósticos tiene tanta más relevancia como elemento de descarga del sistema cuanto mayor es el número de dichas conexiones.

La frecuencia de operación de cada tarea también condiciona el resultado de la acción de control aplicada, siendo lógicamente las respuestas más rápidas e intensas las que se obtienen al degradar las tareas con frecuencias más elevadas. Otra característica a considerar nuevamente es el hecho de que no todas las tareas cargan igualmente al sistema, por lo que conviene buscar tareas con un elevado tiempo de procesamiento en relación con su tiempo de ciclo si se desea conseguir un mayor impacto.

En los experimentos realizados, que se muestran en el capítulo 5, se han seleccionado generalmente las acciones de control más intensas, comenzando por los módulos de sensores para pasar luego a los módulos de cómputo de diagnósticos.

Introducción de nuevas tareas

Otro problema a considerar es la introducción de una nueva tarea dentro de un sistema ya estabilizado. Esta situación puede ser tratada de forma homogénea como una perturbación en la carga del sistema, dejando que las propias políticas de control

la lleven al nivel de degradación correspondiente. Otra posibilidad es anticiparse a ese control asignándole ya de entrada a la nueva tarea el nivel de degradación medio al que estén sometidas las tareas de su misma prioridad.

4.3.3. Control de las violaciones temporales

El algoritmo de control de violaciones temporales se activa de forma local cuando un timeout es detectado por algún TD dentro del sistema. Como ya se ha indicado, la "sensibilidad" de esa detección puede ajustarse para evitar que el sistema de control reaccione ante situaciones de sobrecarga puntual o medidas ruidosas. La solución al problema puede alcanzarse bien dentro de un ámbito local o bien a nivel global.

Puesto que se trata de un sistema de tiempo real blando, no se precisan mecanismos de detección más exigentes, como serían los demandados en un entorno de tiempo real duro [Stewart y Khosla, 1997].

Control local

En primera instancia, se aplican las acciones de control locales, cuyo margen de actuación se encuentra delimitado por los umbrales de homogeneidad. De esta forma, si el controlador de la tarea encuentra algún módulo degradable dentro de los límites establecidos, lo añade a la lista de candidatos. Una vez completado el recorrido por los módulos pertenecientes a la tarea, se selecciona uno de ellos para la degradación. Si esto no es posible, se agotan los recursos a nivel local, por lo que genera una señal para notificar el evento al nivel superior. El algoritmo 3 forma parte del código del bucle de control en las unidades TD, y resume la secuencia seguida en el control local de las violaciones temporales.

Dentro del control de timeouts en la implementación de prueba se ha considerado prioritario el intentar respetar las frecuencias de funcionamiento especificadas por el usuario, por lo que se van a aplicar sólo degradaciones en calidad. La degradación en frecuencia se reserva para el caso del control basado en el nivel de carga. Esta no es, por supuesto, la única alternativa. Otra política puede consistir en enviar simplemente señales de degradación a los diferentes módulos y que localmente se decida si se va a aplicar degradación en calidad o en frecuencia.

Algoritmo 3 Algoritmo de control de timeouts local (tarea t_i).

```
ControlActivado = falso
if Timeouts detectados en t_i then
  for m_j ∈ { Módulos en tarea t_i} do
    if Degradación m_j dentro de límites then
      Añadir m_j a Lista_Candidatos.
    end if
  end for
  if Lista_Candidatos no vacía then
    Seleccionar candidato a degradar m_d de Lista_Candidatos.
    Enviar señal degradación a m_d.
    ControlActivado = verdadero
  else
    Notificar al nivel superior.
  end if
end if
if ControlActivado == verdadero then
  Deshabilitar verificación de timeouts en intervalo Tt_i.
end if
```

Control global

El algoritmo de control global (algoritmo 4) para los timeouts toma como dato de entrada la violación temporal detectada por los niveles inferiores e intenta seleccionar candidatos para la degradación que se encuentren en el mismo nivel de prioridad en el que ha sido detectado el error. Si la búsqueda no tiene éxito, se desplazan los umbrales de homogeneidad hacia niveles de mayor degradación y se repite el proceso. Finalmente puede ocurrir que ni en el nivel máximo de degradación sea posible respetar las frecuencias de funcionamiento deseadas, con lo que la única solución es la reconfiguración del sistema o la suspensión de algunas de las tareas actualmente en ejecución.

La anchura de la banda de homogeneidad debe reflejar un compromiso entre rapidez de respuesta y estabilidad. Una banda ancha ofrece un margen de maniobra mayor para las acciones locales a costa de incrementar los desequilibrios dentro del sistema. Una banda estrecha, por el contrario, implica un sistema muy homogéneo pero incrementa el tiempo de respuesta al obligar a que todas las tareas evolucionen de forma acoplada.

Algoritmo 4 Algoritmo de control de timeouts global.

```
for M_i ∈ { Módulos en nivel de prioridad Nivel_P} do
  if Degradación M_i dentro de límites then
    Añadir M_i a Lista_Candidatos.
  end if
end for
if Lista_Candidatos no vacía then
  Seleccionar candidato a degradar M_d de Lista_Candidatos.
  Enviar señal degradación M_d.
else
  if Umbrales de homogeneidad en máxima degradación then
    Suspender tareas en Nivel_P.
  else
    Desplazar umbrales.
  end if
end if
```

4.3.4. Control del nivel de carga

Otra de las situaciones en la que se activan los mecanismos de acomodación es el establecimiento de un nivel de carga de referencia para el sistema. El propósito puede ser, por ejemplo, liberar recursos para activar nuevas tareas, garantizar un margen de seguridad para absorber picos de carga ocurridos incluso con una configuración de tareas fija, etc. En este caso, las políticas de control aplicadas son exclusivamente globales.

El control se inicia con la comparación entre el nivel de referencia externo especificado (carga deseada) y la carga medida para el sistema. Pueden darse tres situaciones:

- **Carga deseada menor que carga medida**: El sistema debe reducir el consumo de recursos mediante acciones de degradación.
- **Carga deseada mayor que carga medida**: El sistema puede aumentar su rendimiento, por lo que se activan las acciones de promoción.
- **Niveles similares**: No se actúa sobre el sistema.

Control por degradación

En el caso de la degradación, la estrategia de control se inicia intentando seleccionar alguna tarea de la más baja prioridad degradable en calidad, si esto no es posible se desplazan los límites de homogeneidad hacia niveles más altos de degradación y se repite la búsqueda. Alcanzados los límites máximos de degradación en calidad, se intenta

Algoritmo 5 Algoritmo de control de nivel de carga (degradación).

```
Deg = falso
Nivel_P = Min_P
while (Deg == falso)&(Nivel_P < Max_P) do
  Degradación en calidad:
  while (Deg == falso)&(Lim_Hom < Max_DegL) do
    if Algún módulo degradable en calidad en Nivel_P then
      Enviar señal degradación.
      Deg = verdadero
    else
      Incrementar Lim_Hom.
    end if
  end while
  if NoDeg then
    Degradación en frecuencia:
    if Alguna tarea degradable en frecuencia en Nivel_P then
      Enviar señal degradación.
      Deg = verdadero
    else
      Incrementar Nivel_P.
    end if
  end if
end while
if Deg == falso then
  Suspensión de tareas:
  Elegir tarea menos prioritaria T_i.
  Suspender T_i.
end if
```

realizar una degradación en frecuencia, comenzando por las tareas menos degradadas. Una vez agotados todos los recursos en este nivel de prioridad, y si aún es necesario seguir degradando el sistema, se pasa al nivel de prioridad superior y se repite el proceso. Cuando todas las posibilidades de degradación han sido activadas, el único mecanismo posible para conseguir reducciones adicionales de carga es la suspensión de las tareas, comenzando nuevamente por las de menor prioridad.

El algoritmo 5 muestra, de forma resumida, el código utilizado para la degradación en el control del nivel de carga.

Control por promoción

Cuando el nivel de referencia de carga se encuentra por encima del nivel actual el mecanismo de control que se dispara es el inverso, esto es, la promoción. Ésta se inicia

con la recuperación de posibles tareas suspendidas, comenzando por los niveles más altos de prioridad y teniendo en cuenta que la reactivación debe producirse en el mínimo nivel de calidad ofertado por la tarea considerada en cada caso. Cuando todas las tareas se encuentran en ejecución, se intenta restablecer los periodos de funcionamiento fijados inicialmente para las mismas, empezando nuevamente por las más prioritarias. Si la promoción en frecuencia finaliza y el nivel de carga especificado lo sigue permitiendo, se pasa a la promoción en calidad en orden decreciente de prioridad. Prolongando esta situación favorable en el tiempo, el sistema llegará a funcionar a pleno rendimiento en todas las tareas.

El algoritmo 6 presenta de forma esquemática el código utilizado para la promoción en el control del nivel de carga.

4.3.5. Algoritmo de control para la distribución temporal de la carga

Los mecanismos de homogeneización del nivel de carga basados en el desplazamiento temporal pueden diseñarse a partir de medidas globales o locales. El caso global presenta una mayor complejidad y supone una mayor sobrecarga en el conjunto del sistema, por lo que se ha preferido un enfoque local. En la implementación realizada, se toma como señal de realimentación la medida local de simetría de procesamiento 4.2. El objetivo es desplazar la tarea hacia zonas de menor saturación dentro del sistema. De esta manera, si la medida de simetría presenta valores cercanos a -1 (tiempo de ejecución en la primera mitad de los datos significativamente menor que en la segunda) la acción de control intenta adelantar la ejecución de la tarea en el siguiente ciclo, mientras que si la medida se acerca a 1 (tiempo de procesamiento es significativamente menor para la segunda mitad de los datos) se introduce un retardo con el propósito de llevar a la tarea hacia intervalos temporales de baja ocupación de CPU.

El problema fundamental de estas técnicas es que sólo son efectivas en situaciones muy determinadas. Es necesario primeramente que los periodos de las tareas en ejecución sean múltiplos, puesto que de otro modo la propia evolución de las tareas provoca un desplazamiento de los picos de carga. Por otra parte, se requieren niveles medios bajos de consumo de recursos para permitir un resultado eficaz. Cumplidos estos requisitos, sin embargo, la acción de control produce resultados adecuados. El algoritmo 7 presenta un resumen del código correspondiente a este esquema de control.

Algoritmo 6 Algoritmo de control de nivel de carga (promoción).

```
Prom = falso
if Tareas suspendidas then
  Reactivación de tareas:
  Elegir tarea suspendida más prioritaria T_i.
  Reactivar T_i.
  Prom = verdadero
end if
if Prom == falso then
  Nivel_P = Max_P
  while (Prom == falso)&(Nivel_P > Min_P) do
    Promoción en frecuencia:
    if Alguna tarea promocionable en frecuencia en Nivel_P then
      Enviar señal promoción.
      Prom = verdadero
    end if
    if Prom == falso then
      Promoción en calidad:
      while (Prom == falso)&(Lim_Hom > Min_DegL) do
        if Algún módulo promocionable en calidad en Nivel_P then
          Enviar señal promoción.
          Prom = verdadero
        else
          Decrementar Lim_Hom.
        end if
      end while
    end if
    if Prom == falso then
      Decrementar Nivel_P.
    end if
  end while
  if Prom == falso then
    Sistema a pleno rendimiento.
  end if
end if
```

Algoritmo 7 Algoritmo de desplazamiento temporal.

```
if Tem_i/Tm_i > 1/3 then
  Retornar.
end if
if SimP_i > 1/2 then
  Retrasar ejecución en Tm_i/2.
else
  if SimP_i < −1/2 then
    Adelantar ejecución en Tm_i/2.
  else
    Retornar.
  end if
end if
```

4.3.6. Coordinación de las políticas de control

La coordinación de los diferentes bucles de control debe analizarse para garantizar la efectividad de las acciones aplicadas. El control de la homogeneización de la carga, sin embargo, y en los casos en que sea aplicable, puede plantearse como un bucle de control de bajo nivel que siempre está operativo y no entra en conflicto con otras acciones. De hecho, un entorno de carga uniforme es la situación ideal para la ejecución de los restantes bucles de control. En la práctica esa situación ideal exige una distribución de las tareas que no siempre se puede alcanzar.

La presencia de violaciones de periodo de funcionamiento implica un nivel de carga elevado (puntualmente del 100 %) y un sistema inestable, por lo que no tiene sentido fijar un nivel de carga de referencia hasta que los timeouts hayan desaparecido y, en todo caso, hacia niveles de carga inferiores. Esto significa que el bucle de control para el nivel de referencia deja de estar operativo siempre que sistema presente timeouts. La figura 4.9 muestra un esquema con la organización de las políticas de control.

Para evitar efectos avalancha en la detección de sobrecargas en el sistema que provoquen una sobrecorrección, la activación de una acción de control deshabilita los mecanismos de corrección durante un intervalo de tiempo determinado (del orden del periodo de la tarea afectada). Esta solución reduce el nivel de autonomía de las políticas locales, pero es necesaria para evitar oscilaciones en el comportamiento del sistema.

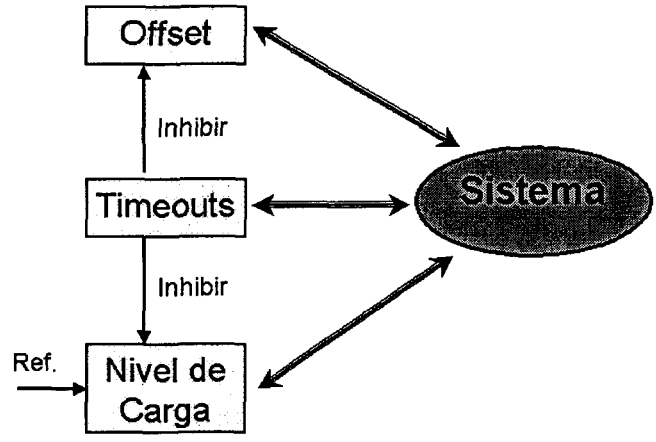

Figura 4.9: Diagrama de coordinación de las políticas de control.

4.4. Adaptación Computacional en Sistemas Calibrados

A diferencia de los sistemas no calibrados, en los sistemas calibrados se cuenta con datos relativos a la correspondencia entre calidad y coste computacional, de forma que el comportamiento de los diferentes algoritmos de procesamiento está caracterizado mediante perfiles de rendimiento. Disponer de esta información a priori permite plantear esquemas de configuración o compilación del procesamiento solicitado con antelación a su ejecución en el sistema, evitando los inconvenientes del ajuste basado en el ensayo-error. El objetivo final es obtener una distribución del tiempo de procesamiento entre los diferentes módulos en ejecución que cumpla con las restricciones de la carga máxima admisible proporcionando un nivel de calidad aceptable.

4.4.1. Tiempo de procesamiento y calidad

En un contexto de procesamiento *anytime* la calidad de un resultado es una cantidad, normalmente expresada en el intervalo [0,1], que mide la bondad del producto de un determinado algoritmo en términos de cualidades tales como su certeza o su precisión/especificidad. La relación que existe entre tiempo de procesamiento y calidad queda determinada por los denominados perfiles de rendimiento. Esta representación muestra cómo evoluciona la calidad del resultado de un algoritmo *anytime* a medida que el tiempo de procesamiento disponible aumenta. La forma típica de estas curvas

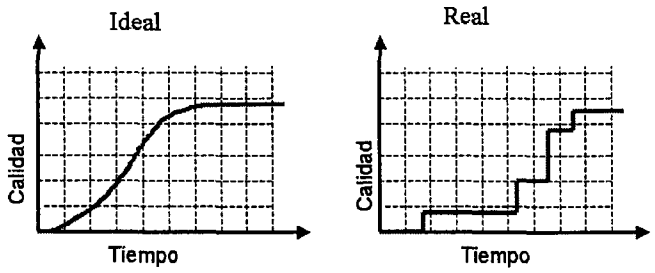

Figura 4.10: Ejemplos de perfiles de rendimiento.

(figura 4.10) presenta un crecimiento en calidad rápido en los instantes iniciales de procesamiento, que se va suavizando hasta saturarse en el nivel de calidad máximo. Se trata siempre de funciones monótonas crecientes, puesto que la calidad siempre debe aumentar a medida que el algoritmo dispone de más tiempo de procesamiento.

Los perfiles ideales son continuos. En la práctica, sin embargo, nos encontraremos con funciones escalonadas, puesto que no será posible aprovechar cada unidad de tiempo adicional para incrementar la calidad. Serán las características de cada implementación las que determinen el número de niveles o saltos de calidad disponibles.

Estas representaciones de calidad/tiempo constituyen una descripción más cercana a los algoritmos *anytime* interrumpibles. Como ya se ha indicado, nuestro sistema se basa en algoritmos de contrato, pero puede igualmente obtenerse esta información realizando test de las diferentes implementaciones para distintos tiempos de ejecución.

Los perfiles de rendimiento pueden además estar condicionados por la calidad de los datos de entrada. De esta forma, se convierten en funciones que no sólo dependen del tiempo disponible para el procesamiento, sino de la calidad de sus entradas. Un ejemplo de estos perfiles condicionados se ilustra en la gráfica 4.11, donde se muestran los perfiles para tres calidades de entrada diferentes.

Composición y compilación

La obtención del nivel de calidad de una tarea pasa por la combinación o composición de los perfiles de calidad de los módulos que la integran. Los sensores cabe

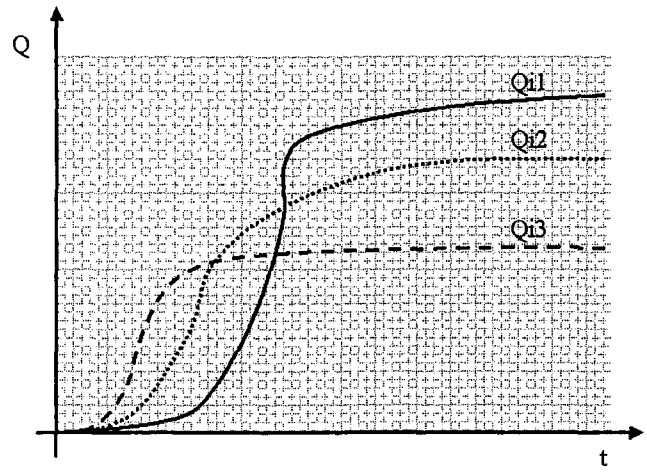

Figura 4.11: Ejemplo de perfil de rendimiento condicionado (Qi1>Qi2>Qi3).

considerarlos como una "calidad de entrada puntual" al sistema, puesto que no consumen prácticamente tiempo de procesamiento; desde el punto de vista del ancho de banda consumido por las comunicaciones, sin embargo, sí pueden existir diferencias importantes. Esta calidad de entrada se combina entonces con los perfiles de los diagnósticos incluidos en la tarea. La calidad resultante es muy cercana a la de la tarea completa, pues las acciones y los actuadores no contribuyen significativamente a su alteración al tratarse de elementos finales con escaso coste computacional.

A fin de ensayar diferentes esquemas de distribución de carga en sistemas calibrados, se ha desarrollado un simulador que permite definir diferentes perfiles de rendimiento y funciones de combinación para topologías arbitrarias de módulos y tareas. Los perfiles implementados son de diverso tipo:

Lineal: Se define un tiempo mínimo, un tiempo máximo y la calidad correspondiente a cada uno de esos extremos (normalmente para el tiempo máximo se alcanza el valor 1 de calidad).

Seno-potencia: Seno con frecuencia parametrizable elevado a una potencia determinada.

Sigmoide: Perfil tipo sigmoide parametrizado.

Constante: Perfil uniforme para módulos sencillos con calidad independiente del tiempo de procesamiento asignado.

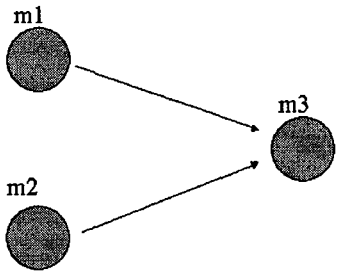

Figura 4.12: Tarea simple utilizada en los ejemplos.

Asimismo se han implementado versiones cuantizadas de cada uno de los tipos para permitir un modelado más próximo a la situación real.

Las posibles funciones de combinación de perfiles de calidad son múltiples. El objetivo es derivar el perfil a la salida de un módulo a partir de los perfiles de las entradas y el propio del módulo. Algunos ejemplos de composición que se han implementado en el simulador son:

- Promedio total.
- Producto del promedio de las entradas por el perfil propio.
- Mínimo total.
- Producto del mínimo de las entradas por el perfil propio.

En la figura 4.12 se muestra una tarea simplificada, constituida por dos módulos productores (m1 y m2) que suministran datos de entrada a un módulo consumidor (m3). La tabla 4.1 muestra el resultado de combinar las calidades de los tres módulos de dicha tarea empleando las funciones comentadas anteriormente.

Vemos como la opción del promedio total es la que proporciona valores más elevados, mientras que el producto de los mínimos es la opción más conservadora.

El problema de la compilación [Zilberstein y Russell, 1996] consiste en el reparto de un tiempo máximo de ejecución para una tarea entre los diferentes módulos que la

Calidad m1,m2,m3	Prom. Total	Prod. Promedio	Mín. Total	Prod. Mínimo
0.5, 0.5, 0.5	0.5	0.25	0.5	0.25
0.5, 0.5, 0.25	0.42	0.125	0.25	0.125
0.5, 0.25, 0.5	0.42	0.19	0.25	0.125
0.125, 0.25, 0.5	0.29	0.09	0.125	0.06

Tabla 4.1: Comparación entre diferentes funciones de composición.

forman. En general, el reparto está condicionado por la maximización o minimización de alguna función objetivo. Tanto la composición como la compilación son los elementos sobre los que se basan los esquemas de control planteados para los sistemas calibrados.

Relación con la carga computacional

Una vez caracterizados en términos de un perfil de rendimiento, la adaptación puede plantearse en términos de tiempo o de calidad: fijar un tiempo máximo y comprobar la calidad resultante, o bien exigir una calidad mínima y verificar el tiempo necesario para alcanzarla. Sin embargo, el control se establece también en términos de la carga computacional a la que está sometido el sistema. Para ello es necesario establecer la relación existente entre calidad y carga.

A partir del modelo 4.1 para tareas periódicas, vemos que la relación entre la carga y la calidad es directa y lineal. Por lo tanto, el perfil calidad/tiempo es equivalente al perfil calidad/carga. Esto no tiene porqué ser así en todos los casos. Cuando los algoritmos incorporan tiempos de espera, por ejemplo, se rompe la linealidad entre la carga y el tiempo, lo que hace que la forma de la curva calidad/carga cambie. Otro ejemplo de no linealidad ocurre en los sensores, puesto que diferentes calidades (en este caso resoluciones) no implican variaciones significativas en la carga inducida en el sistema, que en cualquier caso es muy baja.

Desde la perspectiva del control, lo ideal sería disponer de implementaciones en las que la carga varía linealmente con la calidad, con una relación cercana a la unidad. La peor situación se presenta, por contra, cuando se tienen perfiles en los que la calidad cae bruscamente en cuanto se reduce el nivel de carga. Algunos ejemplos de estos perfiles se muestran en la figura 4.13.

Ejemplos

Veamos algunos ejemplos de la relación entre la asignación de tiempo/carga a los módulos de las tareas y la calidad final resultante.

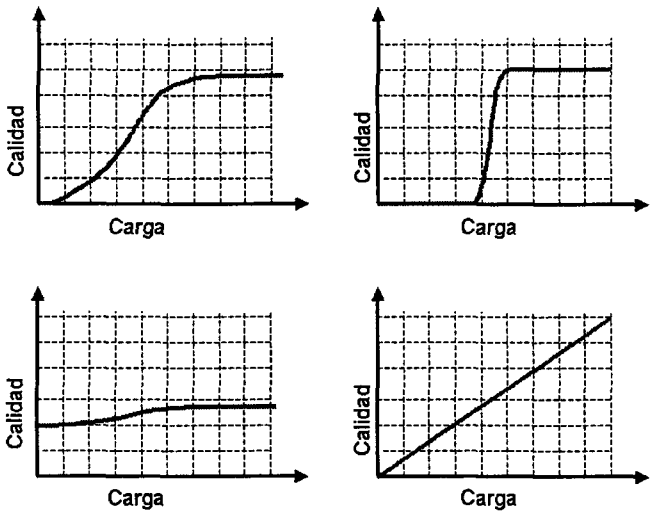

Figura 4.13: Ejemplos de curvas de calidad frente a carga.

Considérese de nuevo la tarea simplificada mostrada en la figura 4.12. Empleando un perfil constante para el módulo cabecera y perfiles lineales para las fuentes se puede comprobar la calidad obtenida para diferentes distribuciones temporales. La gráfica 4.14 muestra la calidad resultante de todas las posibles asignaciones de 10 unidades de tiempo entre los módulos de entrada. Como función de combinación se emplea el producto de la calidad del módulo por el mínimo de la calidad de las entradas.

La gráfica 4.15 muestra la curva que se obtiene para la misma distribución de módulos, pero ahora con perfiles lineales cuantizados

Si se considera que el nodo cabecera también consume recursos, pasamos de una curva en dos dimensiones a una superficie. La gráfica 4.16 muestra el resultado de las simulaciones para perfiles lineales en el módulo cabecera y una de las fuentes y seno-potencia en la otra.

La gráfica 4.17 muestra el resultado para la misma configuración anterior, pero con perfiles cuantizados. Puede apreciarse la existencia de algunos máximos locales en la superficie de calidad.

4.4.2. La compilación de tareas

El problema de la compilación consiste en distribuir el tiempo de procesamiento disponible entre el conjunto de módulos que integran una tarea determinada de modo que la calidad resultante sea máxima. Para un conjunto de $NumM$ módulos, se trata

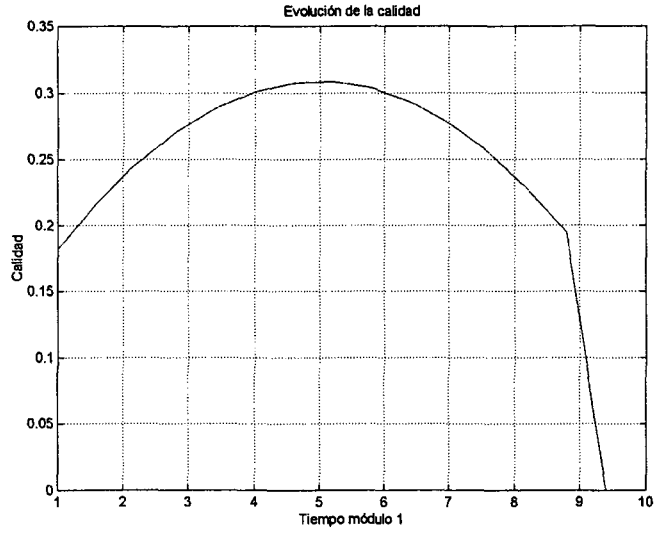

Figura 4.14: Ejemplo de gráfica de calidad para perfiles lineales (2 nodos).

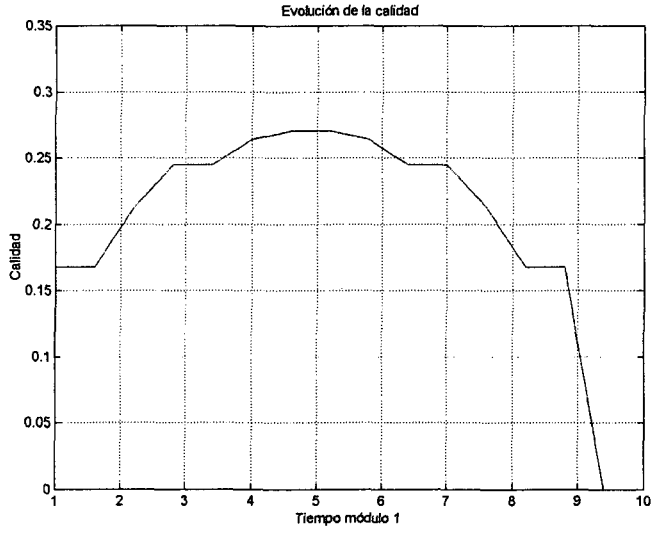

Figura 4.15: Ejemplo de gráfica de calidad para perfiles lineales cuantizados (2 nodos).

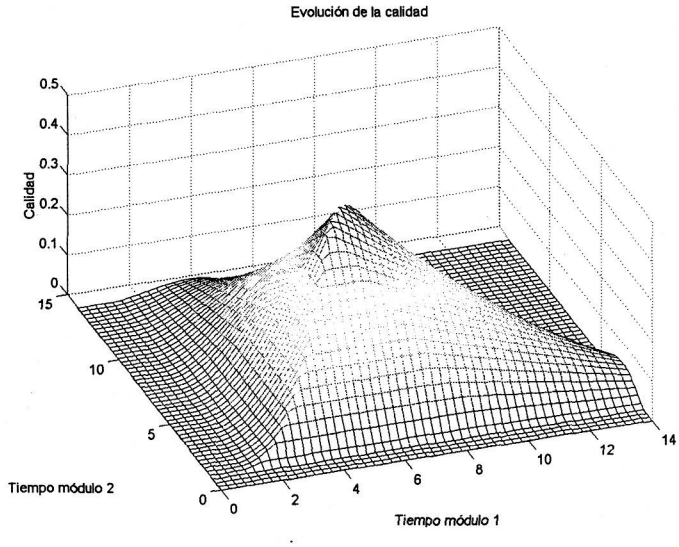

Figura 4.16: Ejemplo de gráfica de calidad para perfiles lineales y seno-potencia (3 nodos).

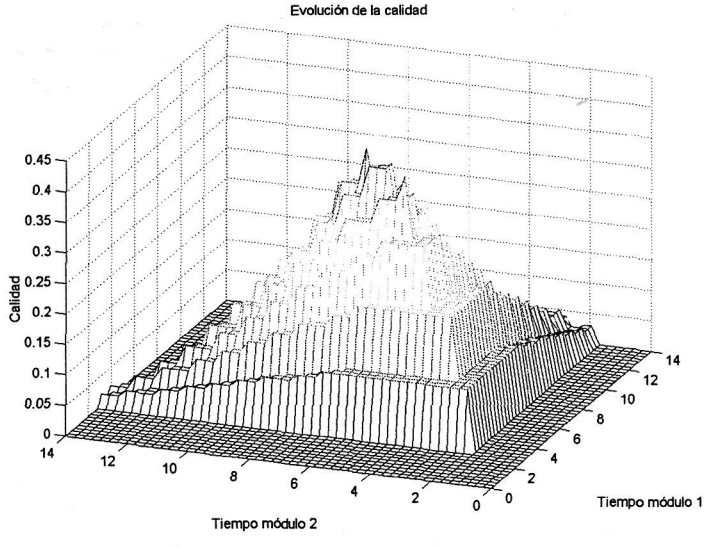

Figura 4.17: Ejemplo de gráfica de calidad para perfiles lineales y seno-potencia cuantizados (3 nodos).

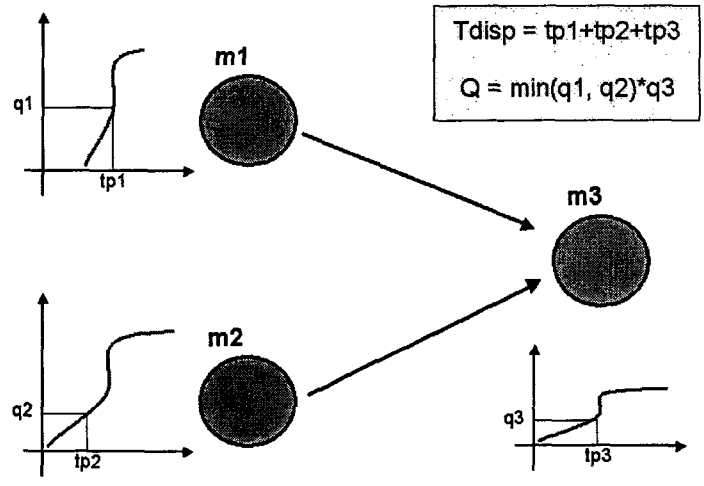

Figura 4.18: Ejemplo de distribución de tiempo y calidad para una tarea con tres módulos.

de encontrar la configuración de tiempos de procesamiento para cada uno de ellos ($T = Tpm_1, ..., Tpm_{NumM-1}$) de forma que deben cumplirse las siguientes condiciones:

$$\begin{cases} T_{disp} = \sum_{i=1}^{NumM} Tpm_i \\ \\ Q(T) \text{ es máxima} \end{cases}$$

donde T_{disp} es el tiempo total disponible para procesamiento, y Q la calidad final resultante para la tarea.

La gráfica 4.18 ilustra el problema para un caso simplificado de una tarea con tres módulos (m_1, m_2 y m_3). Para la combinación de calidades se emplea la regla del producto del mínimo de las entradas por la calidad del módulo.

Este planteamiento supone un problema de optimización del tipo NP-Completo, por lo que una solución mediante prueba exhaustiva de todas las posibles combinaciones no es viable cuando la dimensión del problema crece. En vez de ello, pueden plantearse algoritmos de compilación local, en los que el ámbito de la optimización se reduce, haciendo que el problema global sea computacionalmente tratable. En los trabajos de Zilberstein ([Zilberstein, 1996], [Zilberstein y Russell, 1996]), se presentan algunos algoritmos de compilación local cuya complejidad es polinomial. El autor demuestra además que el resultado de la compilación local (al menos para el caso en que no existan expresiones repetidas) proporciona una solución óptima. Se realizará una traslación de los

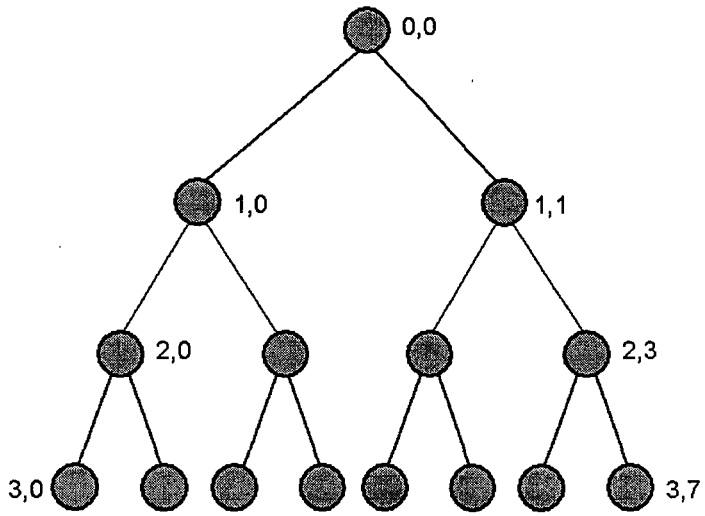

Figura 4.19: Representación de una tarea como árbol binario.

resultados al contexto del sistema propuesto.

Una tarea se puede representar como un grafo dirigido acíclico con nodos fuente para los sensores y nodos terminales para los actuadores. Para simplificar las expresiones resultantes, sin embargo, se prescindirá de los actuadores (que como ya se ha indicado no consumen prácticamente recursos computacionales) y se transformará la representación en un árbol binario completo. La figura 4.19 muestra el resultado, donde los nodos hoja son los módulos sensores, los nodos internos los diagnósticos y la raíz la acción de la tarea.

La calidad de un módulo (i, j) es una función que depende a su vez de las calidades de sus entradas y del tiempo disponible $Q_{i,j}(Q_{i+1,2j}(.), Q_{i+1,2j+1}(.), t)$. El objetivo de la compilación global será maximizar la calidad del nodo raíz de la tarea para un tiempo máximo dado t. Para un árbol de profundidad n resulta

$$Q_T^G(t) = arg\,\text{máx}_{t_{i,j}}\, Q_{0,0}(.), \quad \sum_{i=0}^{n} \sum_{j=0}^{2^i-1} t_{i,j} = t$$

donde $Q_{0,0}(.)$ se obtiene reemplazando recursivamente las funciones de calidad por sus expresiones correspondientes.

Frente a esta optimización global, la compilación local se centra en un nodo y sus entradas directas, de la forma

$$Q_{i,j}^L(t) = arg \max_{t_1,t_2} \{Q_{i,j}(Q_{i+1,2j}^L(t_1), Q_{i+1,2j+1}^L(t_2), t - t_1 - t_2)\}$$

Para los nodos hoja, el resultado de la compilación es directamente la calidad de entrada utilizada, y la calidad final de la tarea es

$$Q_T^L(t) = Q_{0,0}^L(t)$$

Puede probarse que este resultado coincide con el que se obtiene con la compilación global. Siguiendo un razonamiento por inducción, esto es trivial para árboles de profundidad 1 y lo suponemos cierto para árboles de profundidad $n-1$. Para un árbol de profundidad n consideraremos el nodo raíz y los tiempos asignados a los subárboles derecho e izquierdo:

$$t_l = \sum_{i=1}^{n} \sum_{j=0}^{2^{i-1}-1} t_{i,j}$$

$$t_r = \sum_{i=1}^{n} \sum_{j=2^{i-1}}^{2^i-1} t_{i,j}$$

$$t = t_l + t_r + t_{0,0}$$

Se tiene entonces, por definición:

$$Q_T^G(t) = Q_{0,0}(Q_{1,0}^G(t_l),\ Q_{1,1}^G(t_r),\ t_{0,0})$$

Por hipótesis de inducción:

$$Q_{1,0}^G(t_l) = Q_{1,0}^L(t_l)$$

$$Q_{1,1}^G(t_r) = Q_{1,1}^L(t_r)$$

Con lo que resulta:

$$Q_T^G(t) = Q_{0,0}(Q_{1,0}^L(t_l),\ Q_{1,1}^L(t_r),\ t_{0,0})$$

Por definición de compilación local:

$$Q_{0,0}(Q_{1,0}^L(t_l),\ Q_{1,1}^L(t_r),\ t_{0,0}) \leq Q_{0,0}^L(t)$$

Por definición tenemos:

Algoritmo 8 Algoritmo de compilación local de tarea.

```
Asignar tiempos iniciales T_0 a partir del tiempo disponible T_disp.
Calcular calidad inicial Q_0 y tiempo de intercambio s.
while s > s_min do
  for Cada par de nodos i,j do
    t'_i = t_i − s
    t'_j = t_j + s
    Q'_0 = Calidad(T')
    if Q'_0 > Q_0 then
      t_i = t'_i
      t_j = t'_j
      Q_0 = Q'_0
    end if
  end for
  s = s/2
end while
```

$$Q^L_{0,0}(t) = Q^L_T(t)$$

Y por definición de compilación global:

$$Q^L_T(t) \leq Q^G_T(t)$$

En conclusión llegamos a:

$$Q^G_T(t) \leq Q^L_T(t) \leq Q^G_T(t)$$

Con lo que se demuestra que la compilación local es óptima.

Algoritmo de compilación de tareas

El algoritmo 8, inspirado en el trabajo de Zilberstein [Zilberstein y Russell, 1996], presenta un esquema que realiza una compilación local limitada a un módulo fuente y sus destinos directos, de manera que la calidad resultante sea máxima. Para ello implementa un mecanismo tipo *hill climbing* intercambiando tiempo de procesamiento entre los distintos módulos de forma que se obtenga una ganancia en la calidad final. El proceso parte de una distribución inicial de tiempo y aplica intercambio de unidades temporales de tamaño decreciente en cada iteración.

Para la obtención de la calidad final de la tarea en función de las calidades de

los módulos se pueden emplear las diferentes funciones indicadas en el apartado de combinación de calidades.

La asignación inicial de tiempos tiene importancia puesto que una elección acertada puede acortar los tiempos de convergencia. Adicionalmente, se evita en lo posible el inconveniente que representan los mínimos locales. Algunas estrategias que pueden plantearse son las siguientes:

Lineal en $Tmin$, $Tmax$: En ella se asignan inicialmente los tiempos mínimos a todos los módulos y el tiempo restante de forma proporcional a la diferencia entre $Tmax$ y $Tmin$.

Tiempo para carga media: En este caso se distribuye el tiempo proporcionalmente a los tiempos correspondientes a la carga media de los módulos.

El algoritmo se invoca recursivamente desde los módulos finales hasta los módulos iniciales de la tarea, para los que la calidad viene determinada directamente en función del tiempo asignado. El resultado final es la mejor distribución de tiempos posible en términos de la calidad alcanzada. Existen versiones de este algoritmo adaptadas al caso de módulos compartidos por múltiples tareas, situación en la que sólo es necesario reservar tiempo una vez, puesto que se realiza una única ejecución seguida de la distribución de los datos generados a los diferentes consumidores.

Hay que destacar aquí que, aunque la compilación local resuelve el problema de la complejidad exponencial asociada al esquema global, no por ello deja de ser un proceso computacionalmente costoso. Su aplicación, pues, debe considerarse detenidamente, especialmente en el caso de sistemas muy sobrecargados, donde los beneficios potenciales de la información suministrada por la compilación quedarán contrarrestados por la perturbación debida a la ejecución del algoritmo.

Análisis y ejemplos de compilación

Consideraremos ahora una serie de ejemplos para ilustrar los resultados de la compilación. En todos los casos se empleará la regla del producto del mínimo en la combinación de calidades.

En la figura 4.20 se muestran dos tareas con el mismo perfil de rendimiento para todos los módulos. Dicho perfil es lineal y toma el valor de calidad 0.1 para un tiempo de 1 segundo y la calidad máxima para un tiempo de 10 segundos. El resultado de la

compilación aplicada a este caso para diferentes tiempos disponibles se muestra en la tabla 4.2.

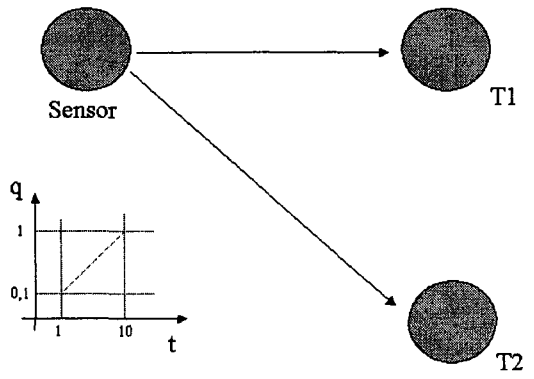

Figura 4.20: Perfil lineal y 2 tareas simples.

Si hacemos que el módulo T1 pase a tener un perfil de calidad que tome el valor 1 en 20 segundos, los resultados son los que se muestran en la tabla 4.3. En ella puede comprobarse como los recursos se desplazan hacia el módulo T1, cuya calidad crece ahora más lentamente en función del tiempo, para no comprometer la calidad global.

Si se considera ahora que es el módulo sensor el que ve modificado su perfil de calidad de la forma explicada anteriormente, resulta la tabla 4.4. En este caso, la calidad global se ve afectada negativamente al ser el módulo afectado un productor de datos.

Por último, la figura 4.21 ilustra un ejemplo más complejo con seis módulos. Existen dos perfiles de rendimiento lineales entre uno y 10 segundos, el perfil 1 varía entre la calidad mínima 0.1 y 1, mientras que el perfil 2 lo hace entre 0.7 y 1. La tabla 4.5 presenta los resultados obtenidos por el algoritmo de optimización con el perfil 1 vinculado a los módulos sensor, diagnóstico 1, acción 1 y acción 3, y el perfil 2 para los módulos diagnóstico 2 y acción 2.

Intercambiando los perfiles asignados en el ejemplo anterior se obtienen los valores recogidos en la tabla 4.6. En estos dos ensayos puede verificarse que la asignación de tiempos se desplaza hacia los módulos más críticos desde el punto de vista de la calidad, que son los asociados al perfil 1.

Tiempo total	Tiempo Sensor	Tiempo T1	Tiempo T2	Calidad final
30	10	10	10	1.0
3	1	1	1	0.01
15	5	5	5	0.25

Tabla 4.2: Resultados de la compilación para 2 tareas.

Tiempo total	Tiempo Sensor	Tiempo T1	Tiempo T2	Calidad final
40	10	20	10	1.0
30	10	13	7	0.67
15	7	5	3	0.20

Tabla 4.3: Resultados de la compilación con dilatación del perfil del módulo T1.

Tiempo total	Tiempo Sensor	Tiempo T1	Tiempo T2	Calidad final
40	20	10	10	1.0
30	13	8	8	0.54
15	7	4	4	0.15

Tabla 4.4: Resultados de la compilación con dilatación del perfil del sensor.

4.4.3. Algoritmos de distribución de tiempo a las tareas

A partir de la información de calibración, se plantean para el sistema diferentes estrategias de distribución del tiempo de procesamiento disponible entre todas las tareas y los módulos en ejecución con el fin de que la carga del sistema no supere un valor máximo determinado. Se va a asumir que el tiempo de procesamiento asignado a un módulo determinado se consume como ciclos activos de CPU, sin incluir tiempos de espera. De esta forma, la carga varía linealmente con el tiempo de procesamiento y se puede establecer una relación directa con el ciclo de trabajo.

El problema de la distribución de carga/tiempo para un entorno de múltiples tareas con diferentes prioridades y frecuencias de operación en ejecución simultánea se refleja en tres niveles:

Reparto por niveles de prioridad: Cómo se distribuye el tiempo total disponible entre los diferentes niveles de prioridad presentes en el sistema.

Reparto por tareas: Dentro de cada nivel de prioridad, cómo se reparte el tiempo asignado a dicho nivel entre las tareas que lo integran.

Reparto por módulos: Para cada tarea, cómo se distribuye el tiempo que se le ha asignado entre los módulos que forman parte de la misma.

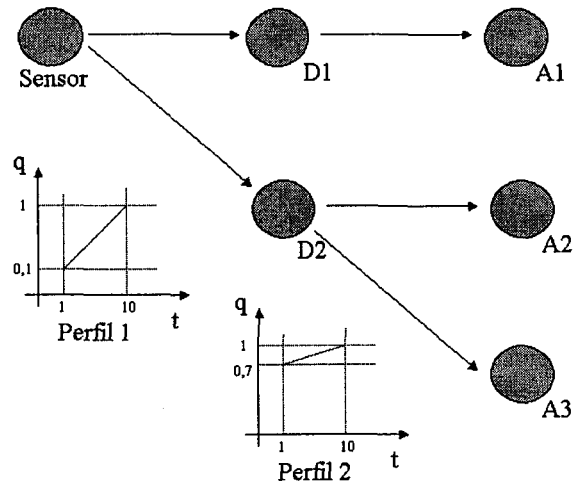

Figura 4.21: Grafo de 3 tareas con perfiles lineales y 6 módulos.

Tiempo total	T.Sensor	T.Diagnósticos	T.Acciones	Calidad final
60	10	10, 10	10, 10, 10	1.0
30	10	6, 1	6.5, 1, 5.5	0.39
15	4	3.5, 1	3.5, 1, 2	0.05

Tabla 4.5: Resultados de la compilación para 6 módulos.

Tiempo total	T.Sensor	T.Diagnósticos	T.Acciones	Calidad final
30	9.5	2, 8	2.5, 7, 1	0.54
15	1.5	1, 5.5	1, 5, 1	0.2

Tabla 4.6: Resultados de la compilación con intercambio de perfiles.

Caben dos aproximaciones a la solución. La primera alternativa consiste en imponer un reparto de carga por niveles preestablecido, mientras que la segunda se basa en extender el proceso de compilación a múltiples tareas, con lo que la distribución de carga es la que ofrece un nivel de calidad final máximo.

Distribución por niveles de prioridad

El primer algoritmo propuesto para la distribución de los tiempos de procesamiento (algoritmo 9) intenta realizar un reparto uniforme por niveles. Los datos de entrada necesarios son los siguientes:

- Nivel de carga global deseado para el sistema Cd.

Algoritmo 9 Algoritmo I de distribución del tiempo de procesamiento.

```
Leer parámetros de entrada.
for Cada nivel j de prioridad (de mayor a menor prioridad) do
  for Cada tarea i en nivel de prioridad (de mayor a menor frecuencia) do
    MAX_Tpt_i = T_i * Cd * PC_j/Numt_j
  end for
end for
```

- Reserva de porcentajes de carga relativos PC_i para cada uno de los $NumNP$ niveles de prioridad. Se cumple $0 <= PC_i <= 1$ y

$$\sum_{i=0}^{NumNP} PC_i = 1$$

- Periodos (Tt_i) y nivel de prioridad (NPt_i) de cada tarea.

A partir de aquí se determina, comenzando por las tareas más prioritarias y de mayor frecuencia, el tiempo de procesamiento máximo asignable a cada tarea t_i en un nivel de prioridad j. Este valor (MAX_Tpt_i) se calcula como

$$MAX\_Tpt_i = \frac{T_i * Cd * PC_j}{Numt_j}$$

donde $Numt_j$ es el número de tareas que se encuentran en el mismo nivel de prioridad j.

Este algoritmo proporciona una solución en la que todas las tareas dentro del mismo nivel de prioridad reciben un reparto homogéneo de carga. Un posible refinamiento consiste en iterar dentro de cada nivel de prioridad redistribuyendo tiempo de procesamiento desde aquellas tareas que no alcanzan su cuota de carga máxima, aún a pleno rendimiento, hacia aquellas que se ven obligadas a reducir su calidad para no exceder dicho límite. Si finalizadas las iteraciones en un nivel sigue sin alcanzarse la cuota máxima, se puede optar por redistribuir esos recursos sobrantes entre los niveles inferiores o simplemente descartarlos.

La segunda propuesta está representada por el algoritmo 10, donde se realiza una asignación ordenada por niveles de prioridad, continuando el proceso mientras queden recursos disponibles. En este caso los datos de entrada son:

- Nivel de carga global deseado para el sistema Cd.
- Periodos (Tt_i) y nivel de prioridad (NPt_i) de cada tarea.

Algoritmo 10 Algoritmo II de distribución del tiempo de procesamiento.

```
Leer parámetros de entrada.
Carga_Admisible = Cd
Nivel_Actual = Maximo_Nivel
while (Carga_Admisible > 0)&(Nivel_Actual >= 0) do
  Calcular Carga_Nivel y Tpt_i para máximo rendimiento.
  if Carga_Nivel < Carga_Admisible then
    Compilar cada tarea t_i usando Tpt_i.
  else
    Compilar cada tarea t_i usando Tpt_i* = Carga_Admisible/Carga_Nivel.
    Carga_Nivel = Carga_Admisible
  end if
  Restar Carga_Nivel a Carga_Admisible.
  Decrementar Nivel_Actual
end while
```

Una mejora para los algoritmos presentados consiste en tener en cuenta que la distribución de módulos entre tareas no tiene porqué estar balanceada, ni en número de módulos ni en carga demandada. Un criterio de reparto más razonable se basa en calcular para cada tarea su carga evaluada para un valor de calidad media $CQMt_i$ como

$$CQMt_i = \sum_{\text{Módulos vinculados a } t_i} \frac{Tpm_j(QMm_j)}{Tm_j}$$

donde QMm_j es el valor de carga media generada por el módulo m_j. En este caso la carga total del sistema para una calidad media vendrá dada por la expresión

$$CQM = \sum_{i=1}^{Numt} CQMt_i$$

Suponiendo un valor de carga máxima admisible C_d, el tiempo MAX_Tpt_i a distribuir entre los módulos vinculados a la tarea t_i es

$$MAX\_Tpt_i = C_d \frac{CQMt_i}{CQM} T_i$$

Distribución por optimización de la calidad

Una distribución alternativa consiste en tratar de alcanzar una calidad óptima para todo el conjunto de tareas, dada una carga máxima admisible. Para implementar este método, sin embargo, deben salvarse una serie de inconvenientes. En primer lugar,

no todas las tareas poseen la misma frecuencia de operación, por lo que no es correcto distribuir tiempos de forma directa. La distribución de tiempo máximo disponible se hace siempre con respecto a la referencia de la frecuencia de operación para poder estimar la carga, con lo que ante referencias distintas no podría emplearse la misma escala temporal. Algunas soluciones aplicables son:

- Añadir al problema de optimización una restricción adicional de la forma

$$C_d <= \sum_{i=1}^{Numt} \frac{Tpt_i}{T_i}$$

  Esto garantiza que la carga final que resulta es la esperada.

- Añadir al sistema $n_i - 1$ réplicas de cada tarea, siendo $n_i = T_{MCM}/T_i$ y

$$T_{MCM} = \text{Mínimo común múltiplo } \{T_i\}$$

- Modificar los perfiles de rendimiento.

La tercera de las soluciones parece la más adecuada por su facilidad de integración en el esquema de compilación local propuesto sin apenas introducir sobrecargas adicionales. La modificación de los perfiles se basa en considerar que las tareas más rápidas van a ejecutarse n_i veces en el interior del nuevo periodo. Para alcanzar el mismo nivel de calidad, por tanto, precisará una reserva de tiempo n_i veces mayor que la que se requiere para una única ejecución. Esto supone escalar los perfiles en el eje temporal multiplicándolo por ese mismo factor. La gráfica 4.22 muestra el resultado de escalar un perfil de rendimiento en un factor 2.

Con estas modificaciones, el tiempo disponible (T_{disp}) para la distribución en el sistema es simplemente el producto de la carga deseada (C_d) por el nuevo periodo.

$$T_{disp} = C_d \times T_{MCM}$$

El siguiente inconveniente es la necesidad de disponer de una medida de la calidad global de todo el conjunto. Este problema tiene fácil solución, pues basta con hacer que todas las tareas confluyan en un módulo de salida común ficticio con perfil de carga constante y unitario, de forma que su calidad sea la combinación de las calidades de todas las tareas. La figura 4.23 muestra el esquema resultante para un caso simple. El algoritmo de compilación puede tratar el nuevo problema sin precisar modificación

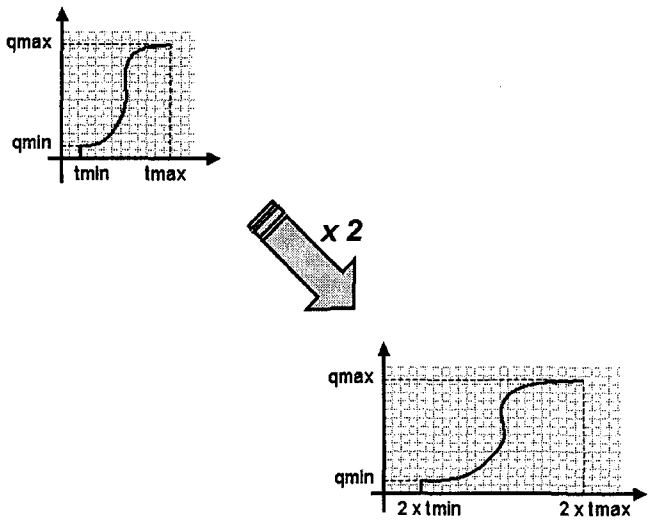

Figura 4.22: Ejemplo de un perfil de rendimiento escalado en un factor de 2.

alguna, puesto que en la optimización local no se asume ninguna estructura previa en la red de módulos. La combinación de calidades, como ya se ha visto con anterioridad, puede estar basada en una función de promediado, un cálculo de mínimo, etc.

Finalmente queda un aspecto por resolver, que es la influencia de la prioridad en la asignación de carga. De alguna forma, un sistema con pocos recursos debería reflejar una relación directa entre calidad y prioridad. Una posibilidad que se ha probado es afectar la medida de la calidad final de cada tarea por un factor que dependa de su prioridad. Si ese factor es menor a medida que la prioridad es mayor se consigue el efecto deseado, puesto que el sistema asigna más recursos a las tareas más prioritarias y una menor cantidad a las menos prioritarias. La separación relativa que exista entre los factores de ponderación de la prioridad determina la naturaleza de la distribución. Así, es posible desde obtener una distribución independiente de la prioridad, a otra en la que sólo se asignan recursos a un nivel de prioridad cuando todos los superiores han sido cubiertos al 100 % de su calidad. En el primer caso estaríamos hablando de factores iguales para todos los niveles, mientras que el último corresponde al empleo de factores que están claramente dispersos.

Tomando como referencia el ejemplo de la figura 4.23, la tabla muestra el resultado de distribuir 30 unidades de tiempo entre las dos tareas para diferentes valores de prioridad. Se define el mismo perfil de rendimiento para los módulos m1 y m2, lineal entre las coordenadas de tiempo-calidad (0,0) y (30,1).

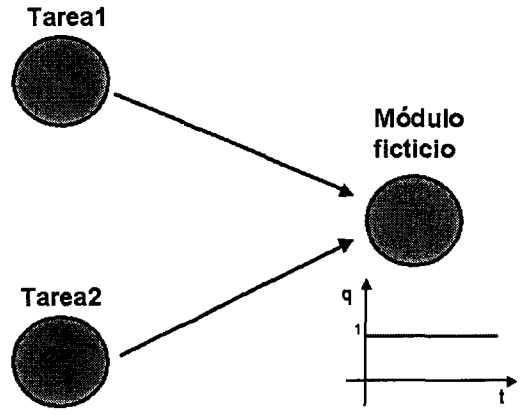

Figura 4.23: Estructura para la optimización de múltiples tareas.

Prioridades m1,m2	Tiempo m1	Tiempo m2
1, 1	15.0	15.0
1, 2	10.2	19.8
3, 1	22.2	7.8

Tabla 4.7: Influencia de la prioridad en la distribución temporal.

La distribución de tiempos basada en la optimización de la calidad, aunque constituya el esquema más adecuado desde el punto de vista de la calidad final alcanzada, presenta el inconveniente de que una modificación en cualquiera de los parámetros del problema puede llevar a una reconfiguración del reparto de tiempos para todo el sistema, lo que puede representar un coste y unos retardos considerables. En estos casos, un esquema como el visto en el apartado anterior permite una reconfiguración localizada, aunque no proporcione la máxima calidad posible.

4.4.4. Políticas de control

Una vez establecida la distribución temporal para las tareas de la aplicación, el sistema debería comportarse en base al modelo teórico. Sin embargo, puede ocurrir que durante la ejecución aparezcan variaciones en la carga no esperadas. Las causas de estas diferencias pueden tener distintos orígenes, tales como:

- Imperfecciones en el modelado.
- Errores en la calibración.

- Cargas extras no anticipadas.

Los diferentes bucles de control a emplear coinciden con los ya vistos para el caso de los sistemas no calibrados. De hecho, en el control del desplazamiento temporal puede mantenerse el mismo esquema de funcionamiento, puesto que los recursos de bajo nivel empleados son los mismos en ambos casos.

Aunque caben múltiples enfoques, el objetivo principal de los bucles de control planteados por el sistema será el mantenimiento de un nivel de calidad homogéneo dentro de cada nivel de prioridad en todo el sistema. La diferencia fundamental con el caso de los sistemas no calibrados es que los umbrales de homogeneidad se plantean en este caso de forma directa sobre las medidas de calidad de las tareas, y no sobre el número de niveles de degradación disponibles.

De nuevo el coste de la compilación de tareas juega un papel importante, en especial cuando el número de módulos activados en el sistema es elevado. Hay que tener en cuenta la demanda computacional que representa este proceso para prever sus consecuencias. Suponiendo que no existen máquinas externas sobre las que lanzar la compilación en paralelo, puede adoptarse alguna de las dos posturas siguientes:

- Ejecución en mínima prioridad.
- Reserva de recursos.

La ejecución en mínima prioridad consiste en asignar a la compilación de tareas la prioridad mínima dentro del sistema. Esto significa que ninguna tarea se verá desplazada de la CPU por la compilación y podrán aprovecharse los recursos disponibles en su totalidad. Como aspecto negativo está el hecho de que en un sistema sobrecargado el tiempo transcurrido hasta obtener el resultado de la compilación puede ser considerable, llegando al extremo de no poder completarse si se llega a la saturación de recursos con las tareas en ejecución.

La segunda alternativa es la reserva de un cierto porcentaje de carga a fin de anticiparse a la demanda del algoritmo de compilación sin comprometer la estabilidad del conjunto. Para lograr esto, debe acotarse la carga introducida por la compilación a un valor máximo. Una solución es la programación del algoritmo para conseguir una ejecución fraccionada, de modo que el intervalo de reanudación del cómputo permita controlar la carga. La reserva de recursos permite garantizar un tiempo de respuesta para la compilación a costa de limitar la capacidad máxima del sistema.

En cualquier caso, la ejecución de la compilación de tareas debe notificarse de manera que los mecanismos de adaptación no traten de reaccionar como si se tratase de una carga externa no modelada.

Control de violaciones temporales

La detección de un timeout en un nivel de prioridad determinado admite un control en dos niveles, local y global. Los umbrales de calidad mínima y máxima determinan el margen de maniobra de las acciones locales, de forma que se notifican los errores al supervisor cuando dentro de la tarea se han agotado los recursos de control.

A nivel local, la magnitud de la acción de control puede establecerse en función de la diferencia que existe entre el nivel de calidad actual y el nivel de calidad mínimo admisible. La selección del módulo a degradar afecta también a la magnitud de la corrección. A igualdad de calidad final, la acción más efectiva es la que provoca una mayor reducción en el tiempo de ejecución final de la tarea.

A nivel global, las posibilidades de selección de candidatos a la degradación aumentan. Respetando los límites de homogeneidad, el efecto más intenso se obtiene con las tareas cuya contribución de carga al sistema completo sufre una reducción más importante.

Control del nivel de carga

En el control del nivel de carga intervienen tanto mecanismos de promoción como de degradación. La ordenación de las acciones es guiada por la prioridad de las tareas, de forma que las más prioritarias son las primeras en promocionar y las últimas en degradarse, mientras que con las menos prioritarias ocurre todo lo contrario.

El comportamiento del bucle de control depende de la selección de candidatos. Pueden adoptarse dos posturas:

Primar la homogeneidad: Se seleccionan siempre los candidatos con mayor calidad para degradar y con menor calidad para promocionar.

Primar la rapidez de respuesta: Se seleccionan siempre los candidatos con mayor efecto sobre la carga del sistema.

Algoritmo 11 Algoritmo para la incorporación de una nueva tarea sin alterar la configuración actual.

$CDisp = CargaDisponible()$
Test admisibilidad:
$CMin = CargaMin(T_{new})$
if CMin>CDisp **then**
  Salir.
end if
Calidad entrada:
$QMPrio = CalidadMediaPrio(Prio_{new})$
$QCDisp = CalidadTarea(T_{new}, CDisp)$
$QTarea = Min(QMPrio, QCDisp)$
Activar tarea con calidad $QTarea$.

Admisión de nuevas tareas

En un sistema estable sobre el que se están ejecutando múltiples tareas, surge otro problema interesante cuando se desea introducir una nueva. Caben múltiples aproximaciones, que varían en función del tipo de algoritmo de distribución empleado y el grado de perturbación generado sobre la configuración actual.

En un primer escenario se impondrá que la configuración de las tareas que actualmente se ejecutan en el sistema no sufra alteración. Tomando además la carga máxima deseada, bien por nivel de prioridad o para el sistema completo, puede realizarse un test de admisibilidad previo. Este test consiste en comprobar si la tarea, con su nivel de calidad mínimo, tiene hueco suficiente para entrar en el sistema. Si es así, hay que decidir el nivel de calidad con el que la nueva tarea ingresa en el sistema, que estará entre el mínimo anterior y un máximo que viene dado por la menor de las siguientes calidades: la calidad media de las tareas en su mismo nivel de prioridad y la calidad con la que se cubre el nivel de carga disponible. El algoritmo 11 resume los pasos a seguir en este caso.

El siguiente supuesto en la línea de nivel de perturbación creciente sería la modificación de las tareas ya existentes únicamente en el nivel de prioridad de la nueva (P_{new}). Esto puede hacerse si existe una reserva de carga por niveles. En este caso, el test de admisibilidad consiste en comprobar si con todas las tareas en P_{new} a su nivel de calidad mínimo no se supera el porcentaje de carga máximo admitido. Superado el test, se aplica la compilación en busca de la calidad óptima para las tareas implicadas, siendo el tiempo de procesamiento a distribuir el correspondiente a la carga límite.

En caso de que no se disponga de la especificación de límites de carga por niveles

de prioridad se puede considerar tanto el nivel de prioridad de la tarea nueva como todos los inferiores. El test de admisibilidad situará a todas las tareas incluidas en dichos niveles en su calidad mínima y comprobará si hay recursos en el sistema para la ejecución simultánea de todas ellas. Si no es así, existen dos posibilidades: rechazar la nueva tarea o suspender la ejecución de tareas menos prioritarias para darle cabida. Asumiendo que el test es superado, se aplica la compilación con factores de prioridad y límite de tiempo definido por la carga disponible.

Finalmente, el máximo grado de alteración se produce cuando todo el sistema es reconfigurado teniendo en cuenta la nueva tarea. Como en el caso anterior, también aquí puede aplicarse un test previo de entrada para garantizar que al menos con la calidad mínima es posible la ejecución de todas las tareas. Posteriormente, podrá aplicarse la compilación en función de las prioridades y el tiempo de procesamiento disponible.

En todos los casos planteados, la existencia de módulos compartidos complica los algoritmos. La introducción de una tarea no implica sólo una nueva demanda de carga a satisfacer, sino que puede provocar la modificación de las prioridades y/o las frecuencias de operación de algunos módulos ya presentes en el sistema. Esta alteración afecta tanto a la carga total como al reparto de carga por niveles de prioridad, lo que debe ser tenido en cuenta en la medida de la carga actual y la carga disponible. En este sentido, hay que recordar que los módulos compartidos toman como valor de prioridad el más alto de todas sus tareas asociadas, que generalmente serán las que presenten además una frecuencia de operación más elevada.

4.5. Transición de Sistemas no Calibrados a Sistemas Calibrados

A medida que el sistema evoluciona, es posible recoger información relativa al funcionamiento de los diferentes módulos en ejecución. Esta información puede incorporarse a los mecanismos de adaptación para mejorar su funcionamiento. Un ejemplo habitual lo constituye la operación con implementaciones que no han sido previamente calibradas. Inicialmente el sistema asume un perfil de rendimiento por defecto que puede irse modificando hacia su forma real a medida que el sistema evoluciona. El resultado es un proceso de autocalibración del sistema que no precisa de la intervención del usuario. Lógicamente no cabe esperar unos resultados equiparables a los que resultarían de un procedimiento sistemático de calibración, pero sí ofrece una referencia válida para caracterizar los perfiles de rendimiento de manera aproximada.

Este planteamiento tiene sentido cuando consideramos un sistema que va a operar, si no de forma continua, sí por periodos de tiempo prolongados. Hay que considerarlo, por tanto, como un esquema de refinamiento de la adaptación a largo plazo, más que un esquema reactivo.

Uno de los inconvenientes de la autocalibración es que no existen garantías de obtener una descripción homogénea del sistema. Aquellos módulos que hayan podido ejecutarse en diferentes condiciones presentarán una caracterización densa, mientras que otros pueden estar representados por un único punto. En realidad, la propia incertidumbre del entorno hace que no deba perseguirse el obtener una calibración precisa de cada módulo con pruebas de ejecución en multitud de condiciones diferentes. En la mayoría de las ocasiones, un par de puntos pueden dar una aproximación razonable al comportamiento del módulo.

Puede definirse una medida que indique el grado en que un sistema se encuentra calibrado. Para ello, se computa para cada módulo i su grado de calibración como

$$GC_i = \frac{NumNC_i}{NumN_i}$$

donde $NumN_i$, conocido de antemano, es el número de niveles de degradación disponibles en el módulo i y $NumNC_i$ es el número de dichos niveles que han sido calibrados.

El grado de calibración del sistema completo puede obtenerse a partir de las medidas GC_i tomando el mínimo o el valor medio, por ejemplo. Pueden utilizarse medidas más complejas que tengan en cuenta no sólo el número de niveles sino su distribución, pero para los propósitos del sistema que nos ocupa es suficiente con esta aproximación.

La alternativa a estos mecanismos de autocalibración en tiempo de ejecución, son los tests exhaustivos de calibración de los diferentes módulos sobre el hardware de destino. Esta opción proporciona una descripción completa del comportamiento de los módulos previa a su ejecución, aunque no siempre es posible realizarla por el coste que representa en tiempo y consumo de recursos. Además, hay que tener en cuenta las dificultades que puede entrañar la obtención de una caracterización realmente útil. En muchos casos, sobre todo en aquéllos de cierta complejidad, se requeriría una reconstrucción bastante cercana de las condiciones finales de funcionamiento. Esto hace que un simple estudio de los módulos de forma aislada sea insuficiente.

4.5.1. Integración de las políticas de control

En muchos casos nos encontraremos en un estado transitorio entre el inicio de la ejecución y la caracterización completa del sistema. Algunas partes del sistema se encontrarán totalmente calibradas al haberse podido ejecutar en diferentes configuraciones, mientras que otras se han mantenido prácticamente constantes. En esta situación intermedia, el sistema debe decidir cómo aplicar las políticas de control, siendo posibles múltiples aproximaciones al problema. Un enfoque conservador intentará, siempre que sea posible, concentrar las acciones de control en módulos que ya hayan sido probados, lo que aumenta el margen de seguridad a costa, probablemente, de descartar acciones más efectivas. Por el contrario, un esquema más arriesgado tratará de desviar las acciones hacia módulos pobremente caracterizados, lo que permitirá mejorar la calidad de la calibración del sistema. Como aspecto negativo de este comportamiento tentativo o exploratorio está el riesgo de emplear acciones que hagan que el comportamiento del sistema sea más brusco, pudiendo provocar oscilaciones. El coste de la compilación también debe ser tenido en cuenta, pues se trata de la ejecución de algoritmos de optimización computacionalmente costosos que pueden tener un efecto negativo en el rendimiento del sistema. En el sistema en transición podemos hablar entonces de tres niveles de control dependiendo del grado de calibración del sistema:

Control tentativo: Se ensayan acciones exploratorias que no han sido calibradas.

Control calibrado: Se ejecutan órdenes de control aisladas hacia niveles ya calibrados.

Control compilado: Se lanzan órdenes de control coordinadas con previsión del nivel de calidad y la carga resultantes.

La figura 4.24 muestra las relaciones que existen entre los elementos de calibración y control.

Los escenarios de trabajo planteables pueden ser, de una parte, utilizar la compilación como configuración inicial a partir de la cual el sistema evoluciona, o de otra, emplear la compilación siempre que se requiera generar una nueva señal de control. En el primer caso, el impacto en el rendimiento producido por el coste computacional de la compilación es menor, mientras que en el segundo puede ser crítico. En la sección 4.7 se volverá a considerar este aspecto.

A medida que el sistema mejora su nivel de calibración, las calidades que pueden alcanzarse en las salidas son mayores. Las gráficas de la figura 4.25 ilustran de forma

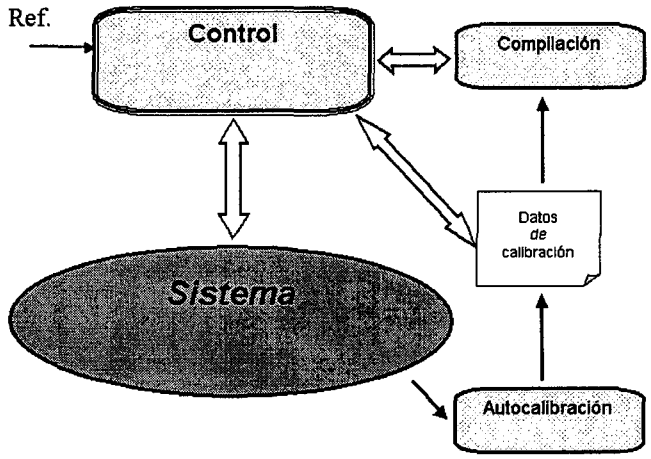

Figura 4.24: Estructura del control incluyendo calibración.

cualitativa cómo evolucionaría la máxima calidad alcanzable para cada nivel de carga a medida que mejorase la calibración del sistema. Las curvas hacen referencia a la calidad de todo el sistema en su conjunto, combinando las calidades de módulos y tareas.

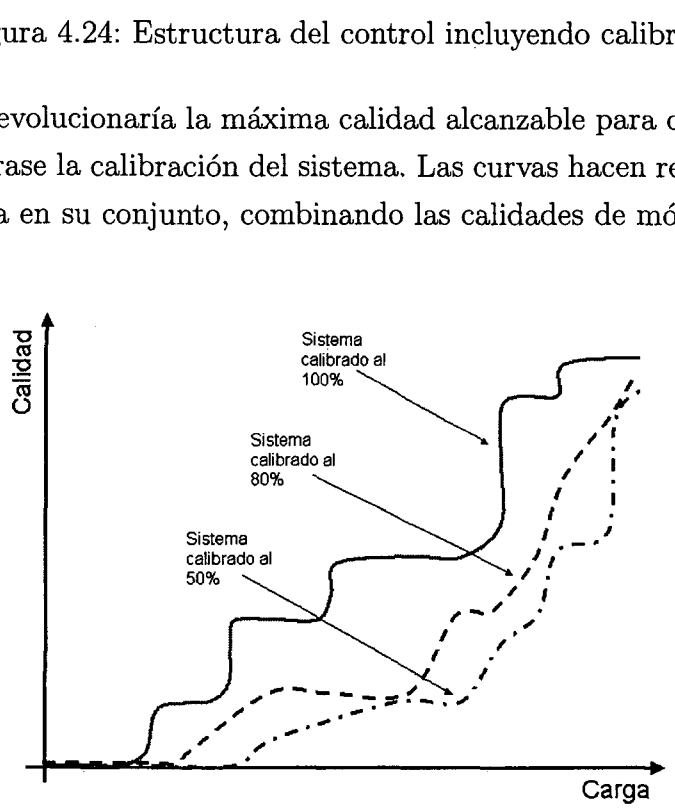

Figura 4.25: Ejemplo de evolución de la calidad con el nivel de calibración.

Los factores que deben ponderarse a la hora de seleccionar un control basado en la calibración o un control de tipo exploratorio son los siguientes:

- Calidad de las salidas.
- Nivel de calibración.

- Error de ajuste.
- Promoción o degradación.
- Nivel de carga del sistema.

Desde el punto de vista de la integridad del sistema, las situaciones más delicadas son aquéllas en las que el sistema está colapsado y es necesario degradar con rapidez, y cuando se intenta promocionar en niveles de carga elevados. La degradación urgente requiere la selección de un módulo calibrado que garantice una degradación eficaz o, si esto no es posible, ensayar acciones de degradación no calibradas, potencialmente efectivas. En el caso de la promoción es conveniente emplear acciones calibradas o, en su defecto, acciones no calibradas de impacto previsiblemente reducido.

Para las situaciones no extremas, el factor fundamental a considerar es el nivel de calibración, indicativo de la profundidad del conocimiento relativo al comportamiento del sistema. Ante un nivel de calibración bajo, los resultados de baja calidad o la imposibilidad de alcanzar un nivel de carga determinado, esto es, un error de ajuste significativo, hacen recomendable la activación de acciones no ensayadas previamente. Con ello, aumentan las configuraciones posibles permitiendo en general un control más preciso y una mayor calidad.

La selección del control exploratorio puede estar a su vez condicionada por otros factores. Así, en entornos donde el sistema pueda estar sometido a cambios bruscos puede ser interesante favorecer el ensayo de configuraciones alejadas de la actual. Con esto mejora el rango de actuación del sistema de control, que puede desplazarse sobre recorridos mayores en el eje del nivel de carga. En cambio, cuando el sistema se encuentre en regiones de funcionamiento estrechas, prima la precisión y la suavidad en la evolución del sistema, lo que hace que resulte más interesante probar niveles de degradación próximos al actual.

4.6. Multiprocesamiento

La propuesta que se recoge en esta tesis se centra en la adaptación de la carga dentro de una máquina monoprocesador. En cualquier caso estableceremos algunas consideraciones sobre los entornos con multiprocesamiento, referidos a los multiprocesadores y los sistemas distribuidos.

En el caso de sistemas multiprocesador, son posibles dos aproximaciones. En la más directa, se asume que el scheduler del sistema realiza un balanceo de carga óptimo

entre los procesadores disponibles. Con esta premisa, es posible tratar el sistema como un recurso computacional único, escalando simplemente las estimaciones de carga para tener en cuenta el número de procesadores disponibles. Una alternativa diferente consiste en aplicar políticas de planificación activas que hagan uso de las capacidades de bajo nivel ofrecidas por el sistema multiprocesador para asignar explícitamente los diferentes módulos a los procesadores disponibles. Las características específicas de cada aplicación en cuanto a configuraciones topológicas, frecuencias de operación o prioridades guiarán esta asignación. Son aplicables aquí algunos de los aspectos considerados para el caso de sistemas distribuidos, considerando que los costes de comunicación y reubicación son ahora mucho menores.

En el caso de sistemas distribuidos se dispone de múltiples máquinas para repartir el cómputo asociado a las tareas en ejecución. Normalmente, estas máquinas son heterogéneas, es decir, poseen características diferenciadas, lo que condiciona dicho reparto. La adaptación computacional puede tener aquí múltiples objetivos, entre los que cabe citar los siguientes:

- Reducir el tiempo de ejecución.
- Balancear la carga.
- Incrementar la tolerancia a fallos.

En el contexto de nuestro sistema son aplicables diferentes estrategias de distribución del procesamiento, todas ellas compatibles con los objetivos mencionados anteriormente.

Distribución por módulos: Se asignan módulos a las diferentes máquinas para que se encarguen de su ejecución.

Distribución por flujo de procesamiento: Se detectan líneas de procesamiento *independientes y se asignan a máquinas diferentes.*

Distribución basada en los datos: Todas las máquinas ejecutan el mismo código, siendo los datos los que se particionan y distribuyen.

En este trabajo se han analizado dos alternativas: la distribución por módulos y la distribución basada en los datos. Para el primer caso, la discusión se centrará en los criterios de distribución y en la integración con las políticas de control ya descritas en este capítulo. Para el segundo caso, se presenta el resultado de una primera fase de

este trabajo de tesis relativa a la distribución de cómputo sobre redes de estaciones de trabajo heterogéneas.

4.6.1. Distribución de módulos

La adaptación en un entorno distribuido añade un grado de libertad adicional a las políticas de control analizadas, puesto que además del nivel de degradación de las tareas es posible elegir la máquina sobre la que deben ejecutarse. Esta característica puede aprovecharse siguiendo dos esquemas básicos, en función de que se permita o no la reubicación de las tareas en tiempo de ejecución. Sea cual sea el esquema utilizado, existen una serie de factores significativos a considerar:

Características del entorno: Puede tratarse de un conjunto de máquinas para uso exclusivo de las aplicaciones del sistema percepto/efector o que se compartan con otros usuarios y aplicaciones. Aún en el caso de uso exclusivo, pueden existir diferencias en la predecibilidad o fiabilidad del entorno.

Diferencias en la potencia de cómputo: Las máquinas pueden ser homogéneas o bien existir diferencias significativas en la potencia de cómputo, memoria, velocidad del procesador, etc.

Hardware especializado: Determinados equipos pueden estar especialmente diseñados para funciones especiales de almacenamiento, procesamiento o visualización.

Ubicación física de sensores y actuadores: Los sensores y actuadores estarán conectados de forma directa a determinadas máquinas dentro del conjunto.

Características de la red de interconexión: Las velocidades de transferencia y los perfiles de carga pueden variar de unos sistemas a otros.

Junto a estos factores, pueden imponerse limitaciones adicionales, como es el establecimiento de niveles de carga máxima admisibles para las diferentes máquinas.

Desde la perspectiva de la ubicación estática, la decisión de la distribución adecuada de las tareas debe analizar los factores anteriormente citados de forma conjunta. Al tratarse de una decisión que se va a prolongar en el tiempo, tiene más relevancia el comportamiento histórico de un factor que su valor concreto en un momento determinado. Cuando se considera la reubicación, sin embargo, la perspectiva temporal se acorta y cobra más importancia el análisis de la situación puntual.

La ubicación adecuada de una tarea o de sus módulos debe considerar en primer lugar las limitaciones de carga impuestas. A continuación debe estudiarse la conveniencia de asociar los módulos dependientes de sensores y actuadores físicos lo más cerca posible de las máquinas asociadas, lo que es extensible para otras dependencias del hardware que puedan aprovecharse para mejorar el rendimiento. Junto a estos factores, deben imponerse restricciones para evitar que unas máquinas se vean significativamente más cargadas que otras. A igualdad de carga, lógicamente, siempre será preferible asignar tareas a las máquinas más potentes.

La reubicación de las tareas consiste en migrar uno o varios módulos de un equipo a otro cuando la situación de la ejecución así lo recomiende. En trabajos previos [Hernández-Sosa y Cabrera-Gámez, 1997*a*], hemos comprobado que en la mayoría de los casos el coste que supone esta operación en un sistema distribuido rara vez es compensado por la ganancia que se deriva de un mejor aprovechamiento de los recursos, por lo que no es una estrategia de control que pueda aplicarse con garantías en intervalos cortos de tiempo. De todas formas, y como estrategia de simplificación previa, conviene separar los módulos en dos grupos, reubicables o no reubicables, restringiendo así el ámbito de las acciones de migración.

Las especiales características de un entorno distribuido, con máquinas heterogéneas en la mayoría de los casos, hacen complejo pensar en mecanismos de calibración de los distintos módulos que puedan aprovecharse para refinar la adaptación. No obstante, otros objetivos de caracterización menos ambiciosos, como es la calibración de las potencias relativas de las máquinas sí parecen abordables y fácilmente integrables dentro de los mecanismos de control.

4.6.2. Distribución basada en los datos

La distribución basada en los datos sigue un esquema SIMD (*Single Instruction Multiple Data*) de paralelización. En este caso, se realiza la partición de los datos de entrada a los módulos y su asignación a las diferentes máquinas sobre las que corren los algoritmos de procesamiento. Esta idea es especialmente adecuada en los casos en que se desea aprovechar los recursos de un sistema computacional distribuido para reducir el tiempo de ejecución.

Aplicada a nuestro sistema, la distribución basada en los datos sólo tiene sentido en el caso de módulos sensores que generen volúmenes de datos importantes y módulos de diagnóstico cuya carga computacional y tiempo de procesamiento así lo justifiquen. La reducción en el tiempo de procesamiento que se alcance debe compensar los costes

debidos a las ineficiencias en la partición de los datos, su distribución y los inconvenientes derivados de la necesidad de mantener copias de código redundantes.

Para efectuar un análisis del problema comparativo, se realizaron ensayos sobre la distribución del procesamiento de diagnósticos de imágenes sobre una red de estaciones de trabajo heterogénea [Hernández-Sosa y Cabrera-Gámez, 1997*b*]. En este caso, para generar una carga computacional suficientemente elevada se recurrió a diagnósticos basados en la convolución de una imagen de entrada con una máscara determinada (suavizados, cálculo de gradientes, eliminación de ruido, etc). Los tiempos de ejecución típicos están en torno al minuto sobre estaciones de trabajo HP serie 700.

En esta ejemplificación se diseñaron estrategias de balanceo de carga que tenían como propósito la reducción del tiempo de ejecución aprovechando los recursos disponibles, y se analizaron los resultados obtenidos con esquemas estáticos y dinámicos [Hernández-Sosa y Cabrera-Gámez, 1996], tanto en entornos de carga externa nula como con carga variable.

En los esquemas estáticos, el reparto de la carga se realiza antes de comenzar la ejecución, tomando como referencia parámetros como la potencia computacional de las máquinas disponibles o su nivel de carga. La ejecución finaliza cuando todas las máquinas han terminado su procesamiento, por lo que el objetivo es igualar los tiempos de ejecución asignando más trabajo a las máquinas más potentes y menos cargadas, y menos a las máquinas menos potentes o más cargadas. En los esquemas dinámicos, en cambio, se fraccionan los datos en un número de bloques superior al número de equipos disponibles, asignándose trabajo a las diferentes máquinas a medida que éstas van quedando ociosas.

La partición de los datos en este tipo de aplicaciones de procesamiento de imágenes basadas en convolución con máscaras es fuente de ineficiencias, puesto que es necesario transferir los datos con un cierto solapamiento para evitar la aparición de zonas muertas (no procesadas). Es fácil demostrar que la forma de minimizar esta ineficiencia es tratar de partir la imagen en bloques cuadrados.

Modelado

Para este estudio se plantearon modelos de predicción del tiempo de ejecución en cada uno de los esquemas de balanceo de carga. El modelo para el tiempo de ejecución empleando un esquema de balanceo de carga estático está compuesto por dos términos, uno asociado al sensor y otro al diagnóstico replicado en cada máquina y que denominaremos como procesos secundarios. Un modelo simple para el tiempo consumido por

un proceso secundario sobre la máquina correspondiente debería contemplar las fases de retardo inicial, recepción de los datos, procesamiento y envío de resultados. Una posible expresión para este modelo es

$$te_i = rpos_i + nbt_i(tprev_1 + tt_i + tpost_i) + nbp_i tproc_i + nbr_i(tprev_i + tt_i + tpost_1)$$

donde i es el índice correspondiente a la máquina ($i = 1$ para la máquina local) en la que se ejecuta el proceso secundario. Las cantidades restantes están referidas a la máquina indicada por el subíndice, y representan:

- $rpos_i$: es el retardo debido a la posición ocupada por la máquina dentro de la lista de distribución, y puede expresarse como

$$rpos_i = \sum_{j=1}^{i-1} nbt_j(tprev_1 + tt_j)$$

- nbt_i: es el número de bytes transmitidos desde el sensor.
- $tprev_i(tpost_i)$: es el tiempo de preparación (lectura) por byte de un mensaje.
- $tproc_i(tt_i)$: es el tiempo de procesamiento (transferencia) de un byte.
- nbp_i: es el número de bytes procesados por el proceso secundario.
- nbr_i: es el número de bytes generados como resultado.

El tiempo total de ejecución viene dado por el tiempo de inicialización más el máximo de los tiempos de ejecución de los procesos secundarios en cada máquina.

$$te = tin(n) + Max_{i=1...n}\{te_i\}$$

donde n es el número de procesos secundarios, que normalmente coincide con el número de máquinas que intervienen en el cómputo, y tin es el tiempo de inicialización común, que depende del número de procesos secundarios.

El modelo presentado puede simplificarse a la vista de las magnitudes relativas de los diferentes tiempos involucrados. Así, puede considerarse el siguiente modelo reducido:

$$te = Max_{i=1...n}\{rpos_i + 2nbt_i\ tt_i + nbp_i\ tproc_i\} + tin(n) + O(\overline{tt}\ \cdot \overline{nbp}) \qquad (4.3)$$

donde se han despreciado los tiempos de preparación y lectura de mensajes, y se ha supuesto que un número equivalente de bytes son transmitidos y recibidos para simplificar los cálculos realizados a continuación. Asimismo, y a fin de obtener una expresión más manejable, no se ha detallado la expresión para el número de bytes procesados. El error cometido es del orden del producto del tiempo medio de transferencia ($\overline{tt}$) por el número medio de bytes procesados ($\overline{nbp}$).

El tiempo de inicialización está constituido esencialmente por el tiempo de lectura de la imagen, y la inicialización y la activación de los procesos secundarios. Tras numerosas pruebas, se comprobó que se puede aproximar el tiempo de inicialización por una constante ($tin(n) = tin$).

En el cálculo de la potencia de cómputo nominal de cada máquina se realizaron pruebas de tiempo de ejecución de un fragmento de código representativo del tipo de procesamiento a realizar, siempre con la carga de la máquina lo más próxima a cero posible. Un valor inversamente proporcional al resultado obtenido en promedio se almacena como medida de la potencia relativa de cada máquina.

Así, considerando el tiempo de procesamiento por byte en una cierta máquina i ($tproc_i$) y su potencia relativa (pr_i), y los valores correspondientes a otra máquina j ($tproc_j$ y pr_j), debe existir una relación de proporcionalidad inversa. Asumiendo un crecimiento lineal del tiempo de procesamiento con el número de bytes procesados en cada máquina, se tiene $tproc_j = tproc_i\ pr_i/pr_j$. Sustituyendo en la expresión 4.3, simplificada al considerar el mismo tiempo de transferencia entre las diferentes máquinas ($tt_i = tt, i = 1...n$), y tomando como máquina de referencia la 1 resulta

$$te = Max_{i=1...n}\left\{rpos_i + nbt_i 2tt + nbp_i\ tproc_1 \frac{pr_1}{pr_i}\right\} + tin + O(\overline{tt}\ \cdot \overline{nbp}) \qquad (4.4)$$

expresión donde sólo es necesario medir el tiempo de procesamiento en la máquina local. Con estos valores de potencia relativa es posible determinar cómo va a modificarse el tiempo de ejecución en función del número de máquinas que participen en el cómputo. La expresión del tiempo de ejecución mínimo suponiendo condiciones ideales para k máquinas es

$$te(k) = \frac{[te(1) - tin]pr_1}{\sum_{i=1}^{k} pr_i} + tin \tag{4.5}$$

El modelo para el esquema de distribución dinámico, puede ser el mismo que el utilizado para el esquema estático (ecuación 4.3), modificado para considerar el procesamiento de m bloques de tamaño fijo y despreciando los tiempos de retardo (por el reducido tamaño de las regiones a procesar). Sin embargo, en este caso hay que considerar tiempos de latencia tanto en el procesamiento como en la transferencia. El modelo utilizado, una vez simplificado, es de la forma

$$te = Max_{i=1...n}\left\{nbloq_i\left[tambt2tt + (tint_1 + tint_i + tinp_1 + tambp\ tproc_1)\frac{pr_1}{pr_i}\right]\right\} + \\ + tin + O(\overline{tt}\ \cdot \overline{nbp}) \tag{4.6}$$

donde:

- $nbloq_i$: es el número de bloques asignados al nodo i.

$$\sum_{i=1}^{n} nbloq_i = m$$

- $tproc_i(tt)$: es el tiempo de procesamiento (transferencia) de un byte.
- $tambp(tambt)$: es el número de bytes procesados (transferidos) por bloque.
- tin: es el tiempo de inicialización común.
- n: es el número de nodos que intervienen.

Experimentos con carga nula

En situaciones de carga cercana a cero, el esquema de balanceo estático basado en la estimación de la potencia de cómputo es el que ofrece mejores resultados.

En la figura 4.26 se muestra una gráfica comparativa de los resultados obtenidos para una aplicación que utiliza tamaño de máscara 7x7 por una distribución estática basada en la potencia y una distribución dinámica con 81 bloques (9x9) sobre un conjunto de 8 estaciones de trabajo. En el eje de ordenadas se muestra el tiempo en segundos y

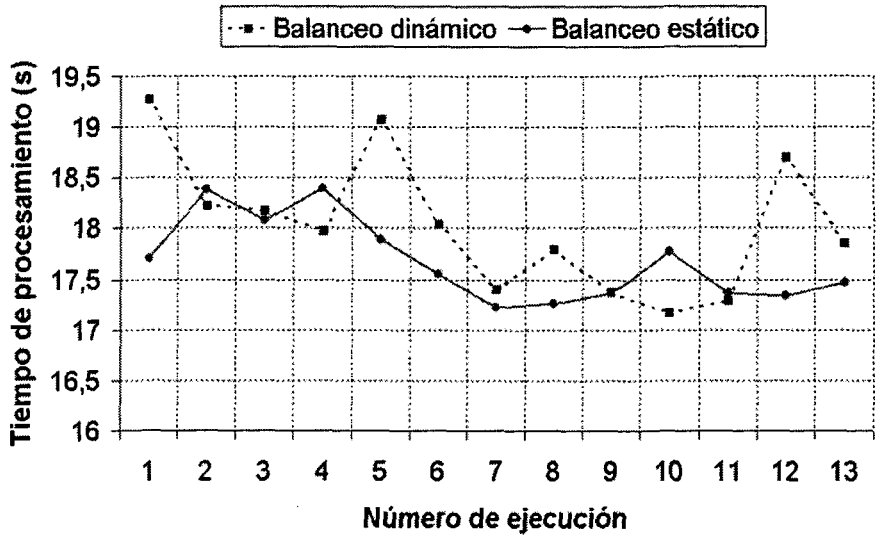

Figura 4.26: Comparación de esquemas de balanceo en un entorno de carga nula.

en el de abcisas las ejecuciones alternadas de cada esquema (cada 15 segundos aproximadamente). Los tiempos se han medido realizando varias ejecuciones y seleccionando el valor menor, que es el que más se acerca a la respuesta con carga nula.

Los resultados del esquema estático son más estables puesto que únicamente se ven afectados por las pequeñas variaciones de carga que se puedan producir en los equipos, mientras que el esquema dinámico se ve afectado además por las fluctuaciones de la carga en la red. Los tiempos de ejecución no varían significativamente de uno a otro esquema siempre que no se incremente en exceso el número de bloques en el esquema dinámico.

La tabla 4.8 muestra los resultados obtenidos con carga cercana a cero para una aplicación que utiliza una ventana de 3x3. Se incluyen junto a los valores reales de tiempo medidos y las estimaciones teóricas obtenidas para el caso en que los recursos computacionales disponibles se aprovechasen al 100 %. Para el esquema dinámico se proporciona además el número de paquetes distribuidos (aquéllos para los que resulta un tiempo de ejecución menor).

La discrepancia entre el valor del tiempo de ejecución real frente al valor teórico ideal en el caso correspondiente a 8 máquinas es del 8.4 % en el caso estático y del 7.6 % en el dinámico, observándose además que la introducción de nuevas máquinas no mejoraba significativamente el tiempo de ejecución (menos de 1 segundo).

N.Nodos	Teórico ideal (Ec. 4.5)	Estático		Dinámico		
		Real	Teórico (Ec. 4.4)	Real	N.Bloques	Teórico (Ec. 4.6)
1	64.74	65	64.94	-	-	-
2	30.33	30.6	30.42	31.1	3x3	30.81
3	21.87	22.2	21.93	22.3	3x3	22.2
4	17.32	17.7	17.37	18.2	4x4	17.75
5	14.59	15.1	14.63	15.3	5x5	15.11
6	12.47	13.2	12.51	13.5	7x7	13.31
7	11.27	12.4	11.3	12.3	8x8	12.24
8	10.44	11.4	10.47	11.3	8x8	11.32

Tabla 4.8: Tiempos de ejecución de la aplicación (seg.) para una ventana de 3x3.

N.Nodos	Teórico ideal (Ec. 4.5)	Estático		Dinámico		
		Real	Teórico (Ec. 4.4)	Real	N.Bloques	Teórico (Ec. 4.6)
1	97.71	98	97.97	-	-	-
2	45.37	45.6	45.48	45.8	3x3	45.87
3	32.49	33.3	32.57	33.4	4x4	33.05
4	25.57	26.5	25.63	27.6	5x5	26.22
5	21.41	22.5	21.46	22.6	5x5	21.95
6	18.2	19.7	18.24	19	5x5	18.65
7	16.37	17.8	16.41	17.8	7x7	17.12
8	15.1	16.4	15.14	17	7x7	15.78

Tabla 4.9: Tiempos de ejecución de la aplicación (seg.) para una ventana de 11x11.

Hay que hacer notar que el esquema dinámico ofrece un buen comportamiento, a pesar de que las condiciones son claramente favorables a un esquema estático. En algunas ocasiones incluso el esquema dinámico ofrece mejores resultados que el esquema estático. Esto es debido a la presencia de pequeñas fluctuaciones en la carga de la red y de los equipos (más probables cuando el número de máquinas crece).

La tabla 4.9 muestra los valores resultantes para una ventana de 11x11. La discrepancia entre el comportamiento real y el ideal en este caso es del 7.9 % para el esquema estático y del 11.2 % para el dinámico. La transferencia de bytes extra es mayor, pero también se incrementa el procesamiento, con lo que los resultados siguen siendo aceptables.

Desde el punto de vista de la eficiencia, los resultados siguen siendo satisfactorios, presentándose evoluciones similares a la comentada para la prueba de la ventana de 3x3

con valores superiores al 80 % en todos los casos. Esto es debido a que el tiempo secuencial supone un porcentaje muy pequeño dentro del tiempo total de ejecución de la aplicación (inferior al 5 %).

Las figuras 4.27 y 4.28 comparan los valores proporcionados por el modelo con los tiempos obtenidos en la práctica con un número creciente de máquinas y para diferentes aplicaciones. La relación *tproc/tt* correspondiente toma un valor en torno a 350 para la aplicación 1 y 200 para la aplicación 2 de la figura 4.27, y 10 para la aplicación 3 de la figura 4.28. En los tres casos el modelo se comporta aceptablemente, con un error de predicción medio situado en torno al 5 % para las dos primeras aplicaciones y en torno al 10 % para la tercera. En este último caso la fiabilidad del modelo se ve afectada por la reducción en la escala de tiempos y la dificultad en la estimación de los parámetros.

Los casos más desfavorables desde el punto de vista de la eficiencia de la distribución del cómputo son aquellos en los que se reduce el factor tiempo de transferencia frente al tiempo de procesamiento (por ejemplo cuando el tamaño máximo de ventana sólo es utilizado por unos pocos procedimientos, siendo para el resto mucho menor).

Experimentos con carga variable

En este apartado se realiza un estudio del comportamiento de los esquemas de distribución de procesamiento en un entorno de carga variable. Dicho estudio se inicia con una comprobación del efecto que las variaciones de carga en los equipos y en la red tienen sobre el tiempo de ejecución de las aplicaciones consideradas. Los datos que se presentan provienen de pruebas realizadas sobre los equipos en horario de uso normal, con varios usuarios trabajando simultáneamente y realizando tareas interactivas con bajo coste computacional (edición de texto, consulta del correo, visualización de directorios, etc). Los resultados seleccionados corresponden a muestras que reflejan los comportamientos típicos detectados.

La velocidad de variación de la carga en los equipos puede apreciarse en la figura 4.29, que considera los tiempos de procesamiento por byte en microsegundos (eje de ordenadas) para un bloque de tamaño fijo en cuatro máquinas y en sucesivas ejecuciones (eje de abcisas) a lo largo de 10 segundos aproximadamente. El tiempo transcurrido entre muestra y muestra está en torno a un segundo. Una variación de 10 microsegundos/byte en la carga de un equipo (aproximadamente una división vertical), representa una alteración sobre el tiempo de ejecución de la aplicación utilizada en las pruebas de 0,5 segundos.

A la vista de esta gráfica queda patente el hecho de que un esquema dinámico es

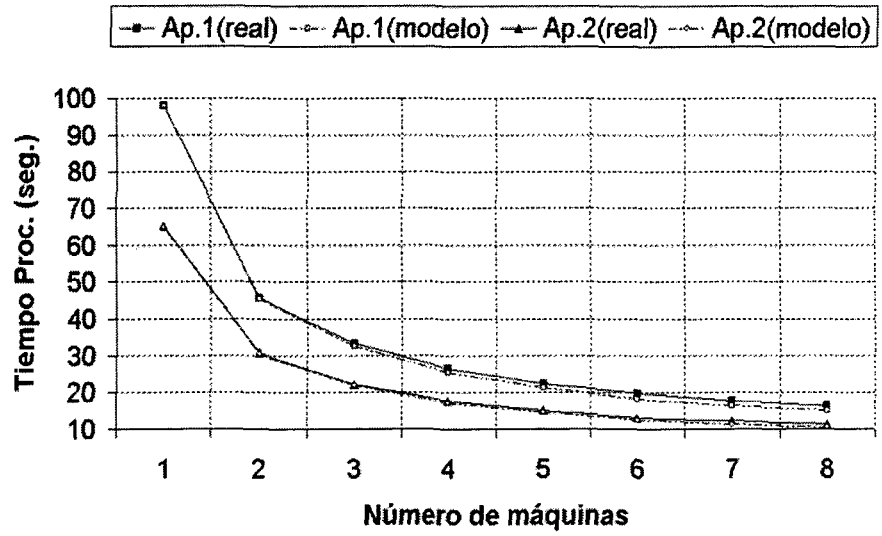

Figura 4.27: Predicción del modelo frente a tiempos reales (aplicaciones 1 y 2).

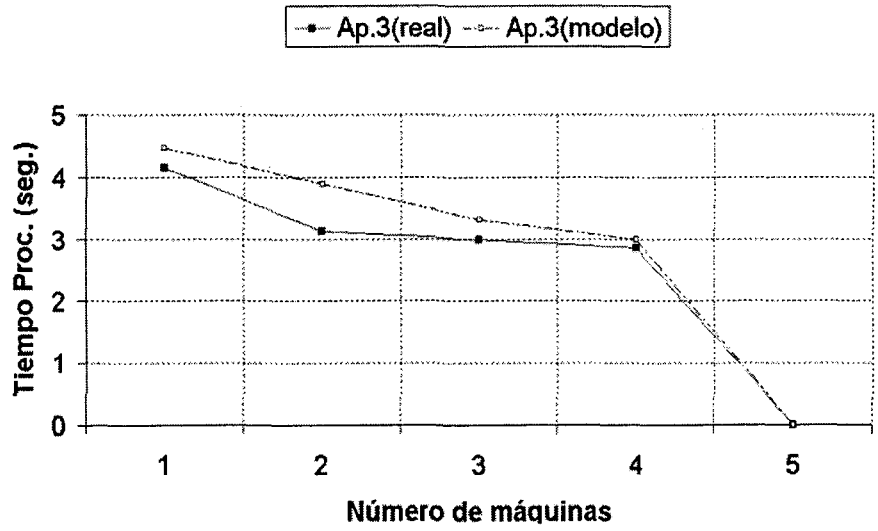

Figura 4.28: Predicción del modelo frente a tiempos reales (aplicación 3).

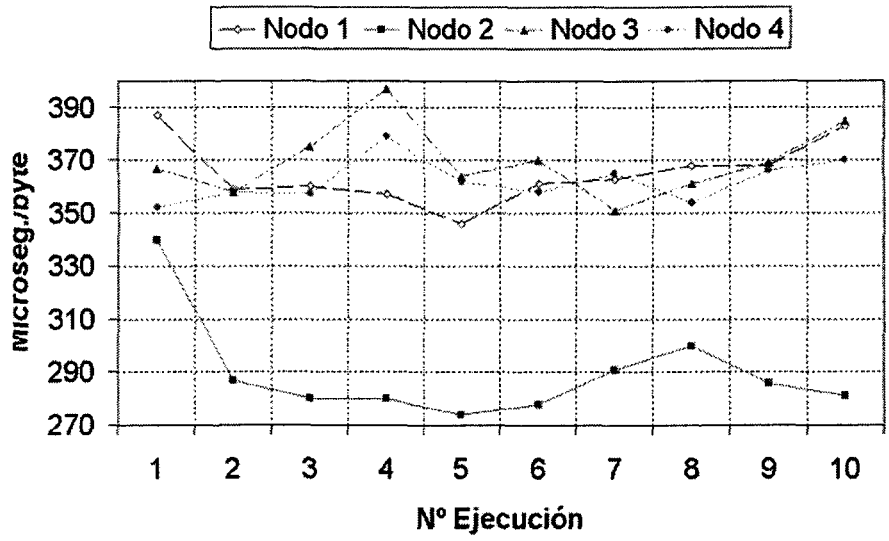

Figura 4.29: Comportamiento del tiempo de procesamiento por byte en un entorno de carga variable.

mucho más adecuado dada la velocidad de variación de la carga. Una estimación estática sólo tendría validez a muy corto plazo (menos de un segundo) cuando las mediciones de carga están próximas a la realidad, o bien a muy largo plazo empleando valores promedio. Estos resultados se verían reforzados si se considerasen para los usuarios del entorno tareas más costosas.

Experimentos con secuencias.

Los tiempos de ejecución en el caso de las secuencias, esto es, con tareas periódicas, siguen las mismas pautas que las indicadas para el caso del procesamiento de una sola imagen. En situaciones de carga nula se obtienen los mejores rendimientos, puesto que el número de bloques puede reducirse al mínimo y, por otra parte, puede solaparse el procesamiento de una imagen con la lectura de la siguiente. En los experimentos realizados, los valores de discrepancia del comportamiento teórico ideal frente al real fueron considerables debido tanto a los reducidos tiempos de ejecución por imagen como a la baja relación tiempo de procesamiento/tiempo de transferencia (en torno a 40). Así, para 4 máquinas se alcanzan valores de discrepancia cercanos al 40 %.

En entornos de carga variable, el rendimiento que se obtiene está en función de la relación entre las mejoras en el balanceo debidas a la reubicación y los retardos

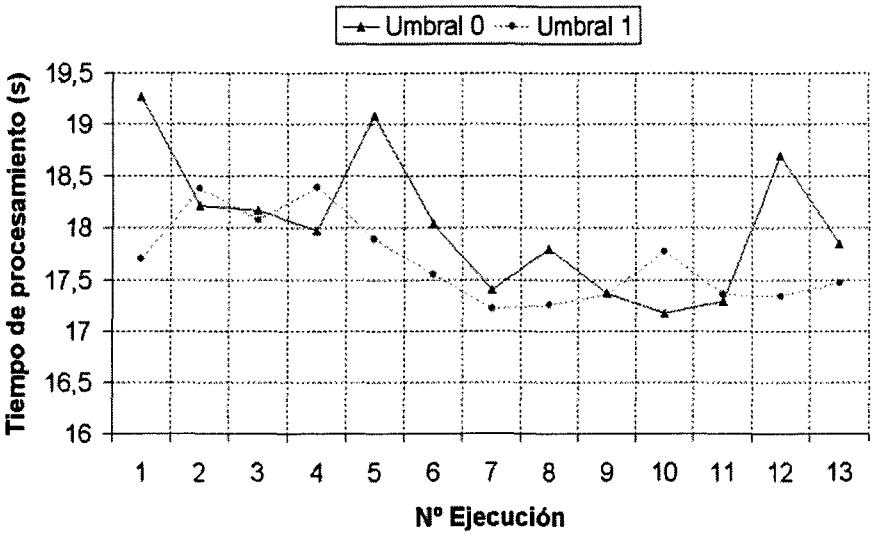

Figura 4.30: Efecto del umbral de reubicación (carga variable).

introducidos para llevarla a cabo.

En general, es conveniente fijar un umbral mayor que uno en la condición de reubicación para evitar que ésta se produzca con excesiva frecuencia. La figura 4.30 compara los tiempos de ejecución obtenidos con umbrales de reubicación 0 y 1 en una situación de carga variable, para cuatro máquinas con división de la imagen en 16 bloques. En ambos casos se presentan fluctuaciones importantes en los tiempos de ejecución, si bien resultan más acusadas cuando no se utiliza umbral.

4.7. Aprendizaje de Situaciones

El aprendizaje incorporado a los mecanismos de adaptación puede reducir considerablemente los tiempos de acomodación. El objetivo es aumentar la robustez del sistema y su nivel de autonomía haciendo uso de la información histórica [Kortenkamp, 2001].

En el caso de sistemas sin calibrar el aprendizaje permite ahorrar tiempo en el proceso de ensayo/error, suministrando una configuración de forma directa. En los sistemas calibrados el aprendizaje se presenta como alternativa a la ejecución de los costosos algoritmos de compilación, lo que de nuevo representa una economía de recursos. En este análisis hablaremos por simplicidad de tareas independientes, a fin de no complicar

innecesariamente la explicación con referencias a la asignación de módulos a tareas.

En nuestro contexto, un esquema de razonamiento basado en casos [Kolodner, 1993] parece ser una solución adecuada. Esta técnica ha sido empleada con éxito en aplicaciones de robótica móvil, en áreas como la planificación de trayectorias, la selección de comportamientos [Likhachev y Arkin, 2001] o el ajuste de parámetros [Lee et al., 2002]. El mecanismo a emplear en nuestro caso está integrado por los siguientes elementos:

Captura de datos:

- Identificación de situaciones estables.
- Almacenamiento de casos.

Control:

- Reconocimiento de casos pasados.
- Generación de las acciones de control.

El objetivo de la etapa de captura de datos es registrar configuraciones del sistema estables junto con la carga que representan. Esta información se utilizará, posteriormente, en la etapa de control. Cuando se comande un nivel de carga deseado para una configuración de tareas determinada deberá comprobarse primero si existe alguna situación similar registrada. Si es así, no es necesario comenzar el proceso de adaptación desde cero, ya que resulta mucho más razonable partir de la configuración conocida almacenada o alguna variante cercana a la misma.

La identificación de situaciones estables se basa en la obtención de medidas de carga real del sistema sobre un intervalo de tiempo suficientemente largo. Nuevamente, debe alcanzarse un compromiso entre filtrado de transitorios y rapidez de respuesta, como ocurría con la estimación de la carga (apartado 4.2.2). En este caso, sin embargo, este factor no es tan crítico como en las etapas de control de violaciones temporales o detección de sobrecargas.

4.7.1. Casos y episodios

Las propiedades que caracterizan una configuración determinada, además del nivel de carga, son las tareas en ejecución, sus periodos de funcionamiento y las prioridades asignadas. Planteado de esta forma, el número de grados de libertad puede invitar a pensar en la escasa utilidad de estos esquemas. Sin embargo, en la mayoría de las ocasiones

la misma configuración de tareas suele estar asociada con las mismas prioridades, mientras que las frecuencias de funcionamiento son tanto una referencia de entrada como un parámetro de ajuste en la adaptación. Esto hace que en muchas situaciones en las que exista un alto grado de coincidencia de tareas el caso almacenado sea utilizable.

Puede definirse entonces el caso a partir del conjunto de tareas activas. Los episodios resultan de parametrizaciones diferentes sobre el mismo conjunto de tareas (caso). Lo más frecuente será que los episodios se diferencien en el nivel de carga fijado y, por tanto, éste será el índice principal. Las prioridades también podrán definir diferentes episodios.

4.7.2. Selección de casos

Si la correspondencia entre el caso almacenado y la situación actual no es exacta, pero sí cercana, pueden plantearse algunas soluciones que intenten aprovechar la información disponible. Suponiendo que las prioridades coinciden, pueden darse las siguientes tipologías de situaciones y las correspondientes estrategias de reparación:

Coinciden las tareas pero no el nivel de carga: En este caso puede realizarse una interpolación sobre los niveles de degradación para obtener una configuración de partida.

Menor número de tareas: La proporción de carga sin asignar, correspondiente a las tareas del caso almacenado no presentes en la configuración actual, contribuye a incrementar los niveles de rendimiento.

Mayor número de tareas: Asumiendo que las tareas extra presentan unas características promedio, puede estimarse el porcentaje de exceso de carga y utilizarlo para incrementar los niveles de degradación de forma proporcional.

Igual número de tareas pero no coinciden todas: Las tareas actuales que no encajan se consideran similares a las registradas en cuanto a carga y se les asigna un nivel de degradación equivalente.

Cuando lo que no coincide son las prioridades se intenta un reajuste del caso almacenado teniendo en cuenta que una mayor prioridad debe ir asociada con un menor nivel de degradación, o lo que es lo mismo, un mayor consumo de recursos.

Pueden emplearse diferentes métricas en la selección de casos o episodios más cercanos. Por ejemplo, para la búsqueda del caso más cercano cuando sólo existe una tarea de diferencia puede utilizarse una medida como la siguiente:

Algoritmo 12 Algoritmo de selección de casos.

if Existe coincidencia entre el caso problema y el caso almacenado **then**
  Seleccionar episodio más cercano (ecuación 4.8).
else
  Seleccionar caso más cercano:
  Seleccionar como casos próximos aquellos que difieran en una sola tarea.
  if Hay casos próximos **then**
    Seleccionar el caso más cercano (ecuación 4.7).
    Seleccionar episodio más cercano (ecuación 4.8).
  else
    Aplicar estrategias de control por defecto.
  end if
end if

$$\alpha|C\_CPU - C\_CPU'| + \beta Dist(A, B) \tag{4.7}$$

donde C_CPU y C_CPU' representan el nivel de carga promedio del caso almacenado y del caso problema respectivamente, A es la tarea no coincidente en el caso almacenado, B es la tarea no coincidente en el caso problema, y $Dist$ es una función que evalúa la similitud entre las tareas no coincidentes a partir, por ejemplo, de los módulos que las integran o sus perfiles de rendimiento. Las variables α y β son factores de ponderación.

Para la selección del episodio más cercano puede considerarse una combinación de las diferencias en nivel de carga y prioridades. Por ejemplo

$$\alpha|C\_CPU - C\_CPU'| + \beta \sum_{i=1}^{n} |P(T_i) - P(T_i')| \tag{4.8}$$

donde C_CPU y C_CPU' representan el nivel de carga del episodio almacenado y del caso problema respectivamente, y $P(T_i)$ y $P(T_i')$ son las prioridades de las tareas en el episodio almacenado y en el caso problema. Las variables α y β son nuevamente factores de ponderación.

El algoritmo 12 resume la estrategia de selección de casos a emplear.

Como variante de las soluciones propuestas siempre será posible trasladar el caso almacenado al real de la forma más directa posible, ignorando las diferencias, para que sean los mecanismos de ajuste los que terminen de alcanzar la solución estable definitiva.

El establecimiento de configuraciones de carga para una tarea determinada es directo cuando ésta se ha calibrado. En caso contrario, cabe plantearse diferentes alternativas que varían en función de los riesgos de sobrecarga asumidos. Una primera

posibilidad conservadora es tomar como nivel de carga a pleno rendimiento el máximo de los que posean las tareas calibradas, y como nivel en máxima degradación también el más alto de los medidos para las tareas calibradas en iguales circunstancias. Otra opción más arriesgada es tomar como niveles de carga los correspondientes al promedio de todas las tareas calibradas.

4.7.3. Integración del aprendizaje, la calibración y el control

En sistemas no calibrados debe coordinarse el aprendizaje con el control. El problema presenta dos vertientes: el almacenamiento de casos no óptimos y los resultados de la compilación no verificados. El primero de los aspectos se refiere al hecho de que las configuraciones estables que pudieran irse almacenando para su posterior reutilización deben ser reconsideradas cada vez que el sistema mejora su nivel de calibración. Esto es así puesto que se trata de configuraciones a las que puede haberse llegado mediante acciones tentativas, que no tienen porqué conducir a resultados óptimos. En el otro extremo se sitúan las configuraciones determinadas por la compilación, que son óptimas desde el punto de vista teórico, pero que no han sido contrastados en la práctica. Aunque en líneas generales, los resultados deben ser válidos, sí pueden aparecer determinados aspectos no modelados que distorsionen el rendimiento final alcanzado. La decisión que debe tomar el control se sitúa entre la seguridad de las configuraciones ya ensayadas frente a la mayor incertidumbre pero mejor rendimiento esperado de las configuraciones teóricas.

Ante el coste computacional que representan los algoritmos de compilación, como ya se ha apuntado anteriormente, puede pensarse en diferentes estrategias de solución para planificar su ejecución. Se pretende con ello evitar que este consumo de recursos acabe afectando al rendimiento del sistema, lo que va en contra del propio objetivo de la compilación. Algunas estrategias planteables serían las siguientes:

- Ejecución en paralelo en máquinas diferentes.
- Ejecución concurrente con baja prioridad.
- Ejecución off-line, sólo cuando el sistema se encuentra ocioso.

En todos los casos, el diagrama de bloques resultante sería el que se muestra en la figura 4.31, donde el gestor de casos es el elemento encargado de mantener la consistencia entre los resultados suministrados por la compilación y los casos estables detectados. En general, los resultados de la compilación se utilizan para orientar las

acciones de control siempre que las condiciones de ejecución lo permitan. De esta forma y a largo plazo, el control basado en casos tiende a mejorar progresivamente la calidad de las configuraciones almacenadas, lo que permite que sea utilizado con garantías en el esquema de control.

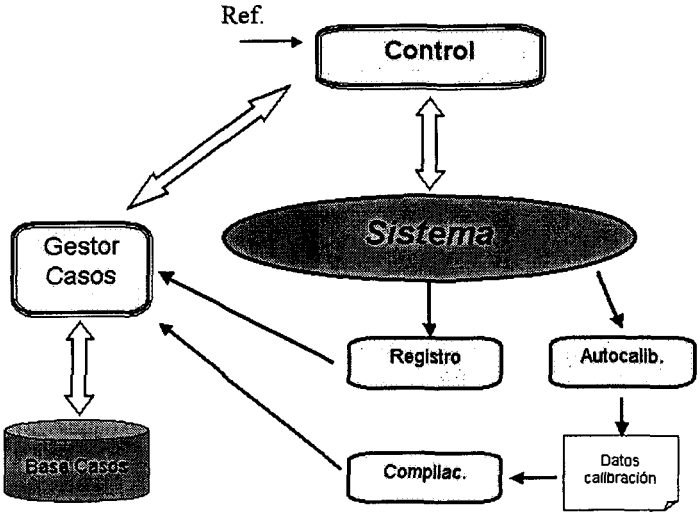

Figura 4.31: Estructura del control con aprendizaje.

Capítulo 5

Experimentos

En este capítulo se intentará demostrar la eficacia de los diferentes mecanismos de adaptación explicados en los capítulos anteriores. Se estudia además la influencia de diferentes factores en los resultados obtenidos. Como apartado final se describe una aplicación real desarrollada en base al sistema propuesto.

5.1. Introducción

Los experimentos que se han realizado tratan de ilustrar diferentes aspectos del sistema, tanto arquitectónicos como relativos a los mecanismos de adaptación computacional. El conjunto de pruebas efectuadas está estructurado en las siguientes secciones y apartados:

Medida de la carga: Pruebas sobre diferentes medidas para la carga del sistema.

Bucles de control: Comportamiento de los diferentes lazos de control planteables.

- Distribución temporal.
- Violaciones temporales.
- Nivel de carga.

Análisis de factores: Conjunto de tests para comprobar la influencia de diferentes factores sobre el comportamiento de la adaptación.

- Resolución.

- Prioridad.
- Topología.
- Hardware.

Sistemas calibrados y no calibrados: Comparación de la evolución en cada caso.

Aplicación real: Implementación de una aplicación de seguimiento.

Puesto que nos encontramos en un entorno de tiempo real blando, no se han utilizado mecanismos de precisión elevada en la obtención de las medidas. El tipo de tareas considerado hace que resulte suficiente para alcanzar los objetivos propuestos la resolución de 10 milisegundos ofrecida por los timers de los sistemas operativos NT/2000.

5.2. Medida de la Carga

Las medidas de la carga del sistema son la referencia sobre la que se aplican las estrategias de control. Por consiguiente, una buena determinación de la carga resulta fundamental para el buen desempeño de la adaptación. En nuestro caso hemos optado por una medida global para estimar la carga a la que se encuentra sometido el sistema en un momento determinado.

Una comparación entre la estimación local y la medida global pone de manifiesto la dificultad que representaría tratar de aplicar una estrategia puramente local en este problema. La figura 5.1 presenta la evolución de los tiempos de ejecución (ver apartado 4.2.2), medido como el tiempo transcurrido desde que se inicia la ejecución hasta que ésta finaliza, para una sola tarea. En este entorno la única carga sobre el sistema, aparte de las tareas propias del sistema operativo, viene representada por dicha tarea. El periodo utilizado es de 2 segundos, y el ciclo de trabajo está en torno al 35 %. Los valores obtenidos son relativamente estables, con una desviación máxima del 0.75 % (debe notarse la influencia de la precisión del timer de NT/2000, 10 ms.). Por lo tanto, en este contexto, podría emplearse una medida local para reflejar la carga global del sistema.

Si el periodo se reduce a 500 milisegundos, de forma que ahora el ciclo de trabajo aumenta hacia el 70 %, se mantiene la estabilidad de las medidas. La variación en los tiempos crece al acercarnos al límite de precisión de los timers utilizados en NT/2000, de forma que la desviación máxima es ahora del 1.45 %. La gráfica 5.2 permite observar el resultado que se obtiene en este caso.

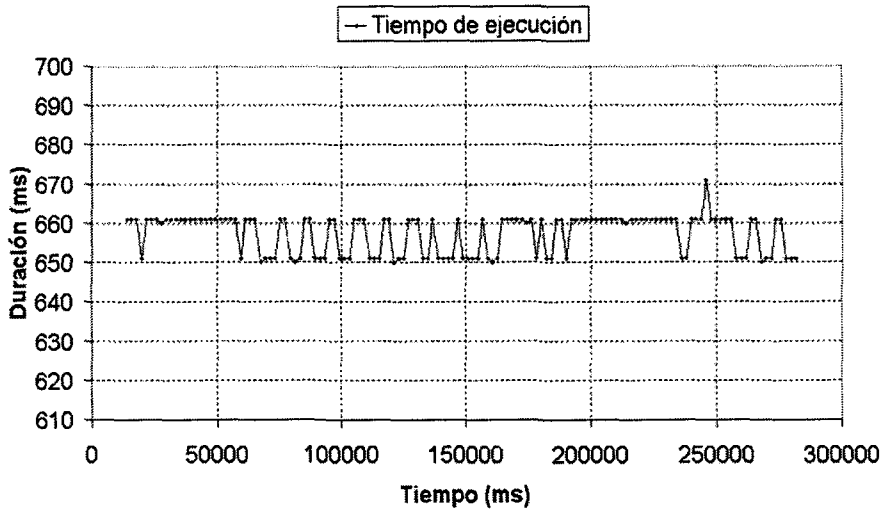

Figura 5.1: Tiempos de ejecución de una tarea (T=2000 ms).

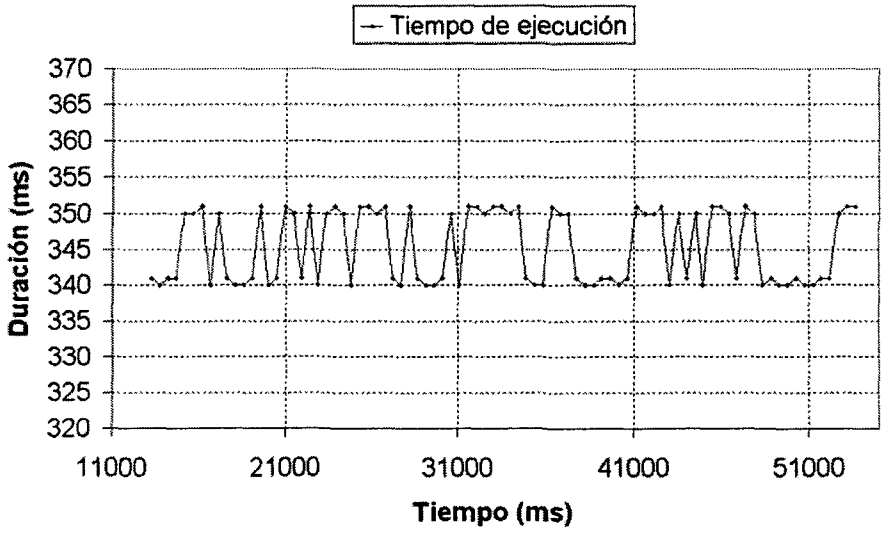

Figura 5.2: Tiempos de ejecución de una tarea (T=500 ms).

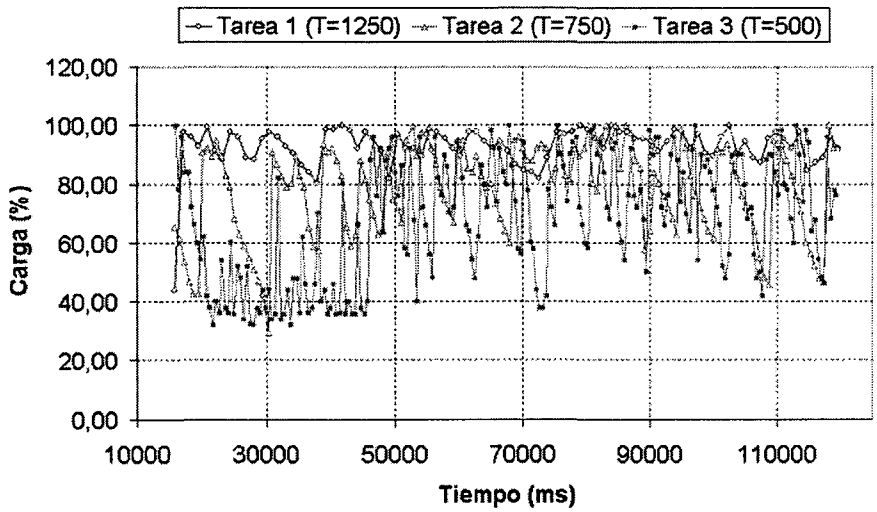

Figura 5.3: Estimaciones locales de carga tomando como intervalo de medida el periodo correspondiente a cada tarea.

En un entorno con múltiples tareas y periodos de funcionamiento diferentes, sin embargo, la variabilidad de las medidas crece. Por ejemplo, en una ejecución de tres tareas con periodos 500, 750 y 1250 milisegundos, las estimaciones locales de carga obtenidas en cada ciclo de ejecución varían tanto de unas tareas a otras como dentro de la misma tarea. Esto puede observarse en las gráficas de la figura 5.3.

La medida global ofrece un comportamiento mucho más estable. La figura 5.4 muestra las estimaciones obtenidas globalmente, y correspondientes al mismo intervalo de ejecución que en el caso anterior. Como ya se ha expuesto anteriormente en este trabajo, el resultado depende del intervalo de análisis de la carga que se tome, que en este caso está en torno a 1200 milisegundos. Cuanto mayor sea el intervalo de análisis más estable será la medida, aunque también disminuirá la capacidad para detectar picos de carga de pequeña duración. Si el intervalo de análisis se incrementa al doble, por ejemplo, se obtienen las medidas que se ilustran en la gráfica 5.5. Este resultado es equivalente a realizar un promediado sobre las estimaciones de carga.

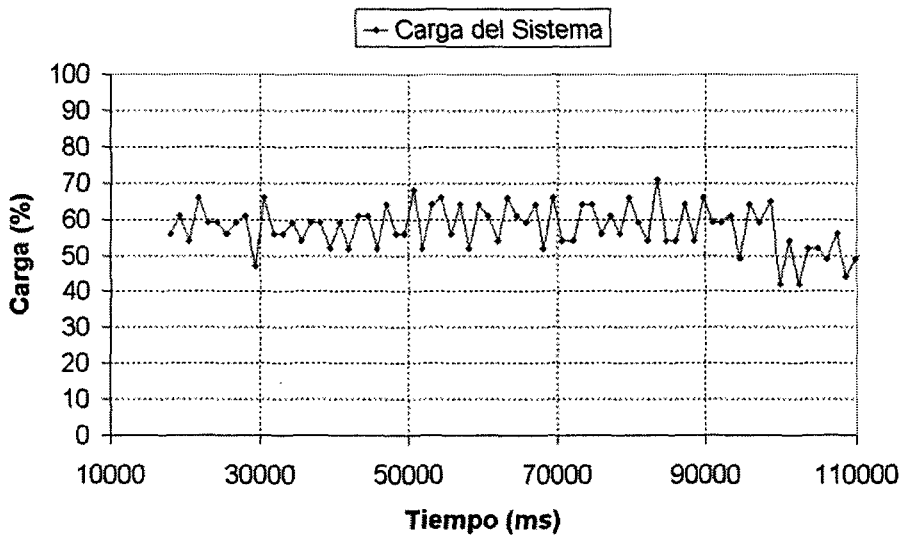

Figura 5.4: Estimaciones globales de carga (intervalo de análisis 1200 ms.).

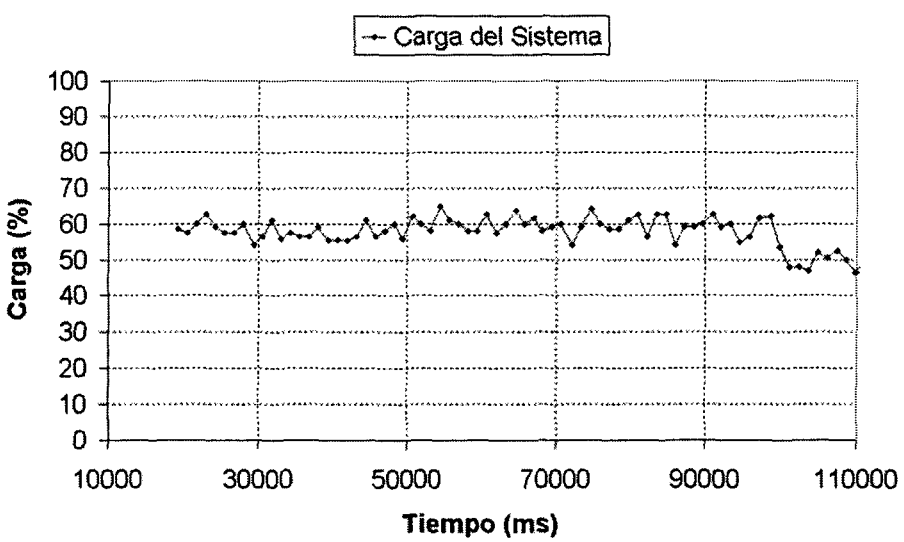

Figura 5.5: Estimaciones globales de carga con intervalo de análisis incrementado (2400 ms.).

5.3. Bucles de Control

En esta sección se analizan los diferentes lazos de control planteados para la adaptación: distribución en el tiempo, violaciones temporales y nivel de carga.

5.3.1. Control de la distribución temporal

El objetivo del control de la distribución de las tareas en el tiempo es tratar de homogeneizar la evolución del sistema, evitando que zonas de congestión coexistan con intervalos de carga nula.

Se realizaron pruebas para comprobar la efectividad del control de homogeneización de la distribución temporal. Debido al reducido contexto de aplicación de este control, se diseñó un entorno simplificado consistente en tres tareas idénticas que se ejecutan con el mismo periodo de funcionamiento, de 500 milisegundos. Las tareas son lanzadas en el mismo instante de tiempo para forzar su ejecución concurrente. En la prueba sin control, las tareas se solapan en el tiempo de manera que la CPU realiza continuos cambios de contexto entre las tres hasta finalizar la ejecución. En la prueba con el control del offset habilitado, las tareas se van desplazando en base al algoritmo y a las medidas locales planteadas.

La gráfica 5.6 muestra el resultado de la ejecución sin control, mientras que en la gráfica 5.7 se aprecia el efecto del control del desplazamiento sobre las tareas en ejecución.

5.3.2. Control de las violaciones temporales

El control de las violaciones temporales tiene por objeto que todas las tareas en ejecución dentro del sistema respeten los periodos de funcionamiento establecidos por el usuario.

La política de control de violaciones temporales se basa en una estrategia jerárquica. Las detecciones se realizan a nivel local, tras lo cual se intenta resolver el problema a ese nivel. Si esto no es posible, se traslada el problema a los niveles superiores.

Consideremos el caso de tres tareas simples e independientes. Cada una de ellas posee un sensor y un diagnóstico cuyas implementaciones admiten dos niveles de degradación en calidad. Esto hace que cada tarea precise cuatro órdenes de degradación sucesivas para pasar de un funcionamiento a máximo rendimiento a operar con el mayor

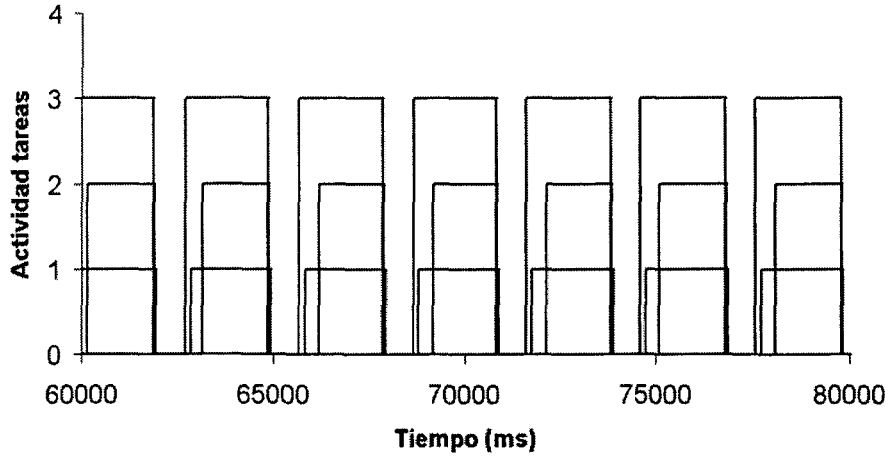

Figura 5.6: Gráfica de la actividad de las tareas sin control de desplazamiento.

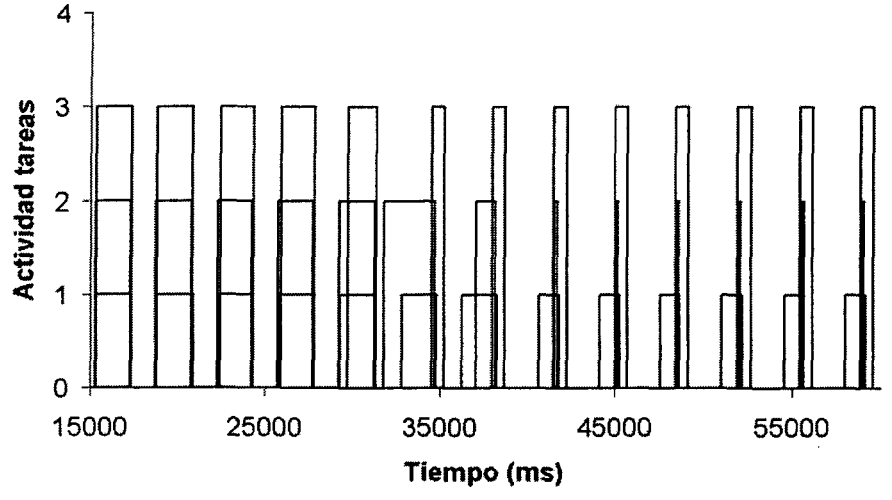

Figura 5.7: Gráfica de la actividad de las tareas con control de desplazamiento (nótese que la resolución del eje de tiempos en esta figura es la mitad de la anterior).

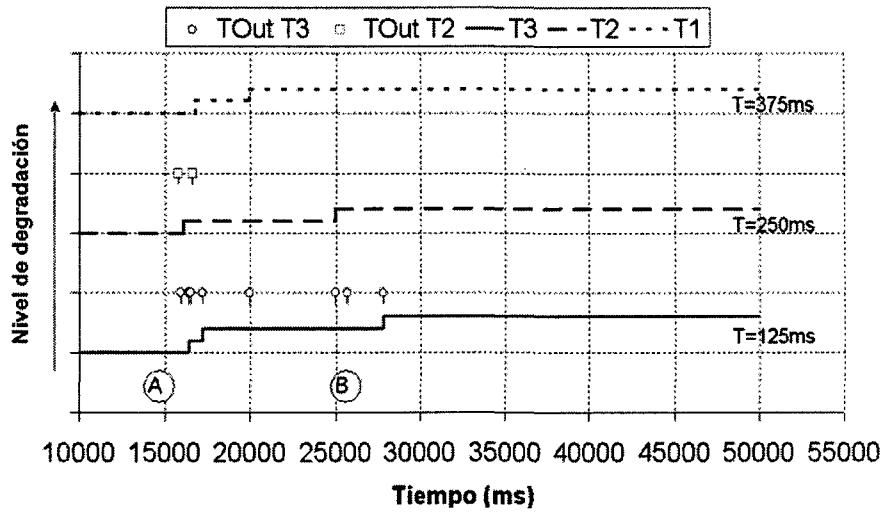

Figura 5.8: Detección de las violaciones temporales y evolución de la degradación.

nivel de degradación posible. Los periodos de funcionamiento fijados son de 125, 250 y 375 milisegundos, inferiores al tiempo de procesamiento requerido por cada tarea a pleno rendimiento a fin de garantizar la aparición de los timeouts. La figura 5.8 muestra la detección de las violaciones temporales y la evolución de los niveles de degradación. Las violaciones se indican mediante marcas sobre la línea asociada a la tarea que las detecta, mientras que las degradaciones vienen representadas por cambios en el eje de ordenadas en la línea representativa de cada tarea, que arranca en el nivel de degradación cero y se va incrementando a medida que se producen nuevas degradaciones. A la derecha se muestran los periodos de funcionamiento de cada tarea. Dos etiquetas, A y B, señalan puntos de interés por la acumulación de timeouts a partir del segundo 15 de ejecución, que es cuando se activa la detección de las violaciones temporales y se habilitan las políticas de control.

En esta gráfica puede comprobarse el efecto de la interacción de las políticas de control locales y globales. La tarea 3 es la que sufre un mayor número de timeouts, lo que hace que los recursos locales de adaptación se agoten y se notifique el error al nivel superior. Desde este nivel se produce la difusión de la señal de control a nivel global para acabar afectando a la tarea 1 (etiqueta A), que no llega a sufrir directamente violaciones temporales.

La carga del sistema se ve afectada por las diferentes acciones de control activadas

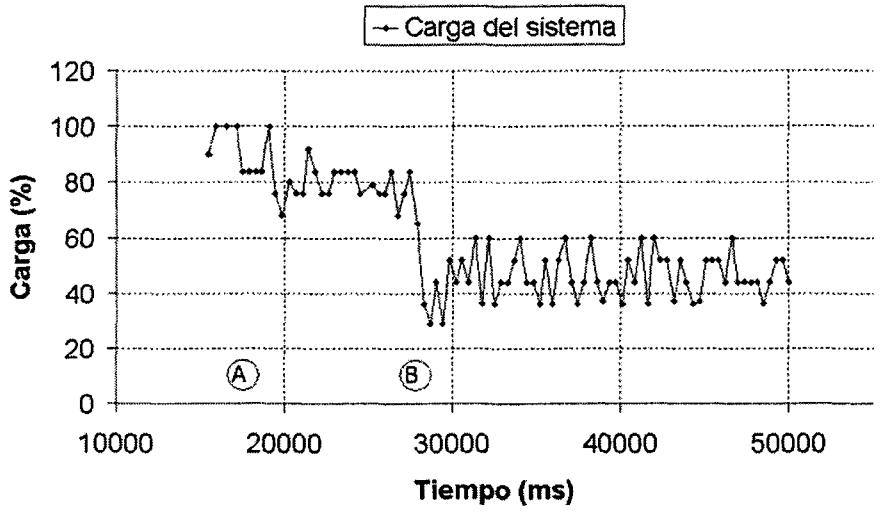

Figura 5.9: Evolución de la carga (intervalo de análisis 500 ms.) del sistema durante la ejecución de las tareas con control de violaciones temporales.

tal y como se muestra en la figura 5.9, donde se emplea un intervalo de análisis de la carga de 0.5 segundos. Se produce una primera reducción de la carga hasta el segundo 20 de ejecución (etiqueta A) debida a las acciones en respuesta a las violaciones detectadas. La carga se estabiliza temporalmente hasta que vuelven a detectarse violaciones a partir del segundo 25 de ejecución (etiqueta B) que hacen que el sistema degrade nuevamente. A partir de ese punto ya no vuelven a detectarse timeouts y no hay nuevas degradaciones.

5.3.3. Control del nivel de carga

No siempre resulta interesante que la totalidad de los recursos del sistema sean consumidos por las tareas en ejecución. En determinadas circunstancias puede ser necesario liberar recursos, por ejemplo, antes de lanzar una nueva tarea. En el control del nivel de carga se establece un nivel de referencia que los mecanismos de promoción/degradación deben tratar de mantener, lo que permite reservar un porcentaje de la capacidad total del sistema para diferentes propósitos. En la gráfica 5.10 se presenta la evolución del sistema a medida que se van indicando diferentes niveles de referencia, con algunos puntos de interés etiquetados. Se muestran medidas de carga puntual y carga media evaluada sobre un intervalo de 2 segundos.

Al sistema se le fija un nivel de carga inicial del 50 %. Posteriormente, en torno

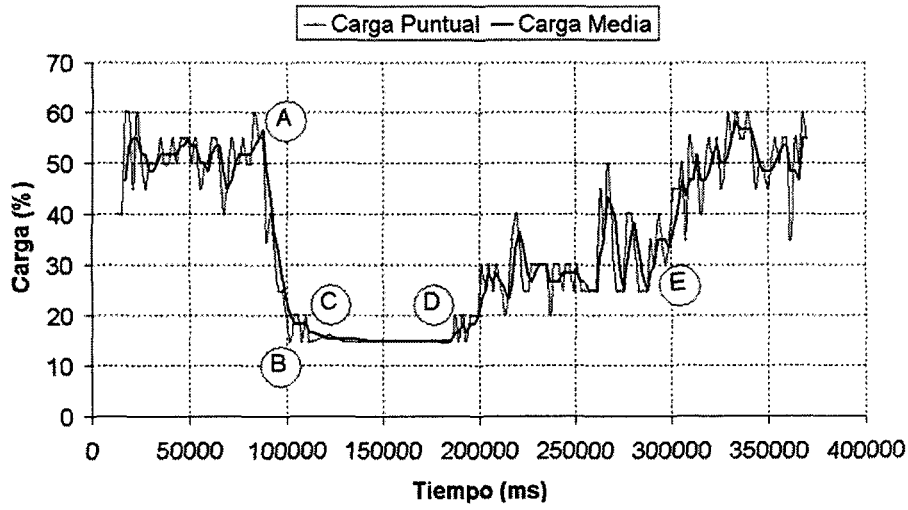

Figura 5.10: Evolución de la carga del sistema controlada.

a los 80 segundos de ejecución (etiqueta A), se le comanda como objetivo el nivel del 10 %. La política de control utilizada consiste en aplicar primero degradación en calidad y, posteriormente, degradación en frecuencia. Los recursos de degradación en calidad se agotan hacia el instante de tiempo $t = 100$ segundos (B). Puesto que aún no se ha alcanzado el nivel de referencia comandado, el sistema aplica control en frecuencia. Sobre el segundo 125 de ejecución (C) se han agotado todos los recursos de degradación sin que se haya alcanzado el objetivo, por lo que simplemente se devuelven señales de degradación fallida al supervisor y el sistema se estabiliza.

A partir del segundo 175 (D) se cambia la referencia de nivel de carga para el sistema al 30 %. El sistema promociona hasta ese nivel aplicando en primer lugar promoción en frecuencia y posteriormente promoción en calidad. Finalmente en torno a los 300 segundos de ejecución (E) se vuelve a la referencia inicial de 50 % de carga.

En la gráfica se observan oscilaciones de diferente amplitud. Las oscilaciones de pequeña amplitud, por ejemplo entre 40 y 50 segundos o entre 200 y 210 segundos, son debidas a fluctuaciones de la carga real del sistema. Las oscilaciones de mayor amplitud, como las presentes entre 250 y 300 segundos, obedecen a la aplicación alternativa de acciones de promoción y degradación. Esto ocurre cuando ninguna configuración de las ensayadas por los algoritmos de control pueden fijar la carga en el nivel objetivo debido a que el número de niveles de adaptación es, como en este caso, reducido. En cualquier

caso, se admite siempre una cierta banda de fluctuación para evitar que el control esté actuando constantemente.

5.4. Análisis de Factores

Se estudian en esta sección los efectos que sobre la adaptación tienen diferentes factores como son el número de niveles de degradación disponibles, la prioridad, la topología o la potencia del hardware de soporte. Los experimentos se particularizan para el caso del control del nivel de carga.

5.4.1. Influencia del número de niveles de degradación

Como se ha indicado, la precisión del sistema en régimen permanente depende de la densidad de niveles disponible. Si en el experimento ilustrado por la figura 5.10 se mantiene la misma configuración de tareas e intervalo de análisis y se incrementa el número de niveles, añadiendo un nivel extra a cada diagnóstico en ejecución, se obtiene el resultado mostrado por la gráfica 5.11. Se han comandado una serie de niveles de carga, incluyendo la referencia del 30 % para una mejor comparación.

Puede apreciarse un cambio sustancial en el comportamiento del sistema en régimen estacionario. Centrándonos en el nivel de referencia del 30 %, el sistema con menor número de niveles presenta oscilaciones de $\pm 10\,\%$, mientras que con el incremento de niveles dichas oscilaciones se limitan a menos de la mitad.

5.4.2. Influencia de la prioridad

Para comprobar el comportamiento de las estrategias de control en función de la prioridad se parte de una configuración de tareas con la misma prioridad y se fuerza la activación de las políticas de control. Posteriormente se repite el ensayo en las mismas condiciones pero con distintos valores de prioridad.

La gráfica 5.12 corresponde a la evolución de los niveles de degradación en cuatro tareas que se ejecutan con la misma prioridad en un sistema sobrecargado. Las tareas utilizadas poseen todas el mismo periodo de funcionamiento (1 segundo) y diferentes capacidades de adaptación. La tarea 1 puede suministrar resultados con dos niveles de calidad, las tareas 2 y 3 admiten cinco niveles de degradación, y la tarea 4 puede operar en tres niveles de calidad distintos.

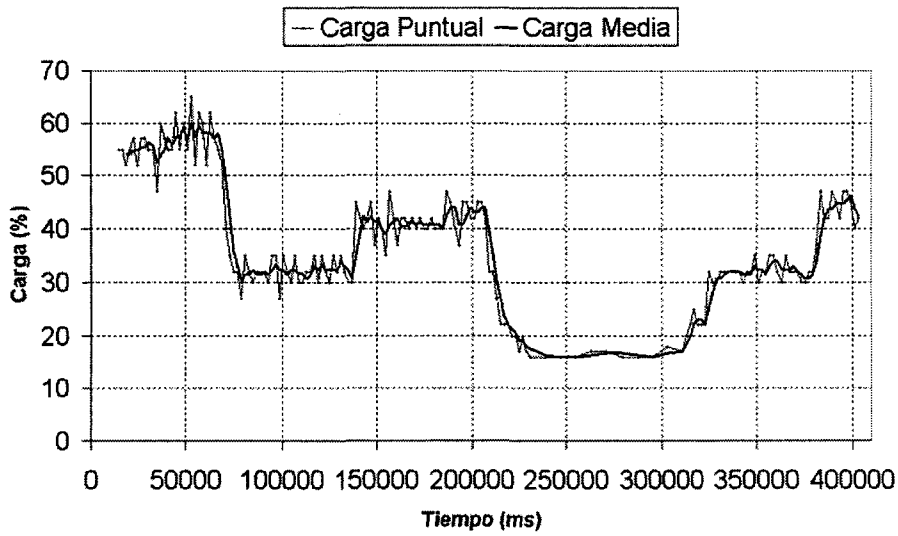

Figura 5.11: Evolución de la carga del sistema con un mayor número de niveles de degradación disponibles.

En la gráfica se muestra el comportamiento de cada tarea cuando se fija como nivel de referencia para la carga un valor del 25 %. En ella se puede apreciar cómo las tareas van degradándose de forma ordenada en función de sus niveles de calidad hasta que se alcanza la referencia de carga marcada.

En la gráfica 5.13 se ejecutan las mismas tareas, pero asignándole a la tarea 4 un valor de prioridad superior. Como consecuencia, las políticas de control se concentran en las tareas menos prioritarias, que son las que sufren las degradaciones hasta alcanzar nuevamente el nivel de carga fijado.

5.4.3. Influencia de la topología

La configuración de los módulos dentro del sistema determina la magnitud de las acciones de control que pueden activarse. En el siguiente ejemplo se ensayan tres configuraciones distintas sobre cuatro tareas en ejecución, comandando un nivel de referencia de carga bajo para activar las políticas de control. Para ilustrar mejor el efecto, la secuencia de las acciones de control se intenta aplicar ordenadamente comenzando por los sensores y finalizando en los actuadores. Asimismo, se ha utilizado un intervalo de análisis corto en relación con los periodos de las tareas para reflejar rápidamente los cambios en la carga global del sistema, aunque a costa de obtener una medida más

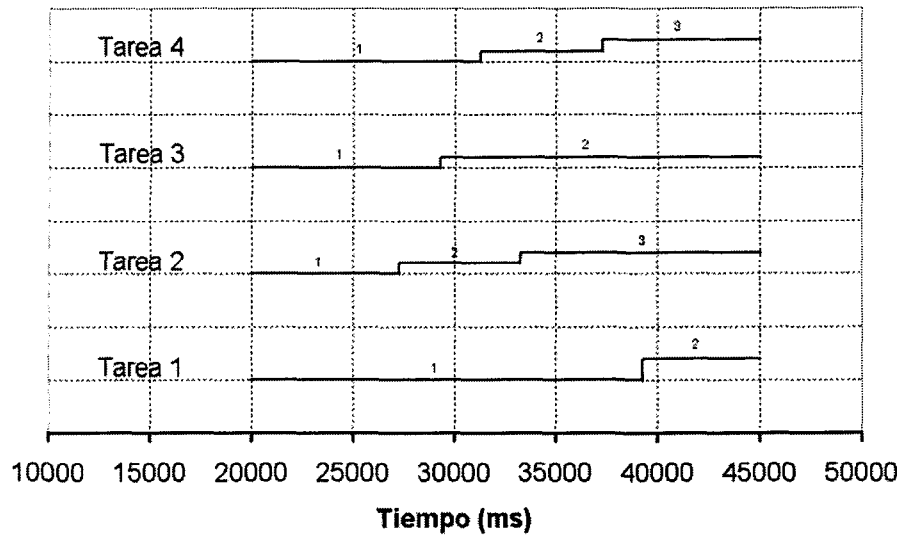

Figura 5.12: Evolución de la degradación en tareas con igual prioridad.

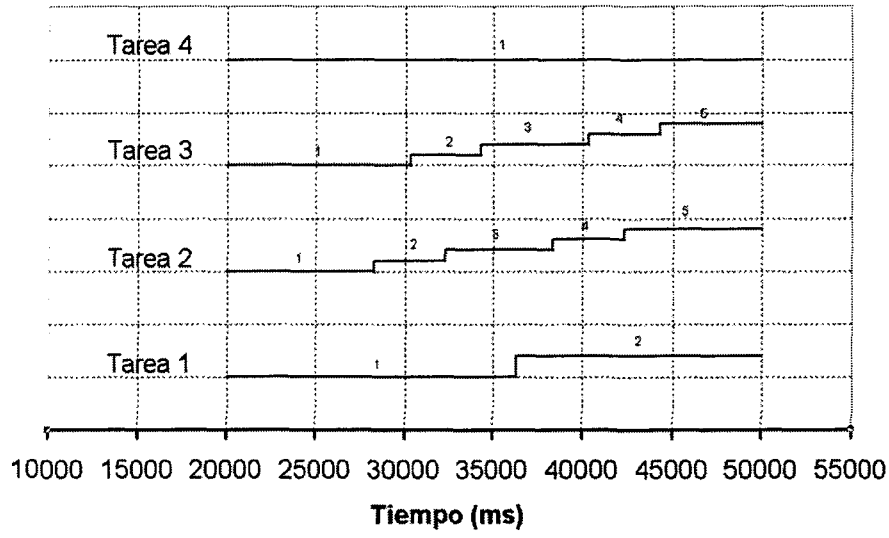

Figura 5.13: Evolución de la degradación en tareas con distinta prioridad.

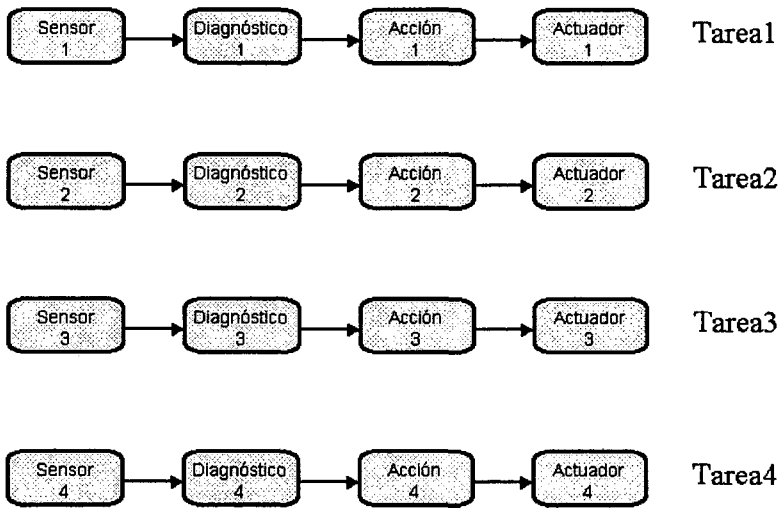

Figura 5.14: Topología de tareas independientes.

ruidosa. Las tareas admiten entre 5 y 10 niveles de degradación cada una de ellas y tienen un periodo de funcionamiento de 1 segundo.

La primera prueba se realiza con tareas independientes que no comparten ninguno de sus módulos productores (ver gráfica 5.14). La gráfica 5.15 muestra el comportamiento de la carga del sistema desde los valores iniciales que rondan el 65 %. A partir del segundo 60 de ejecución se comanda como nivel de referencia para la carga el 25 %, valor que se alcanza sobre los 85 segundos de tiempo de ejecución.

En la segunda prueba se considera una configuración con un sensor compartido por cada dos tareas (figura 5.16). El resultado de la ejecución se ilustra en la figura 5.17, en la que se aprecia un retardo entre la fijación del nivel de referencia (en torno al segundo 32) y la finalización de la secuencia de control (hacia el segundo 42) de unos 10 segundos, frente a los 25 del caso anterior.

Por último, en la tercera prueba se considera que existe un único sensor compartido por todas las tareas (figura 5.18). El efecto sobre el comportamiento del sistema se traduce en una transición entre los dos niveles de carga que tiene ahora una duración de unos 5 segundos aproximadamente, tal y como se muestra en la figura 5.19.

Estos resultados confirman la intuición de que el grado de dependencia debe ser analizado como indicativo de la magnitud de las acciones de control que se activan sobre los diferentes módulos en ejecución.

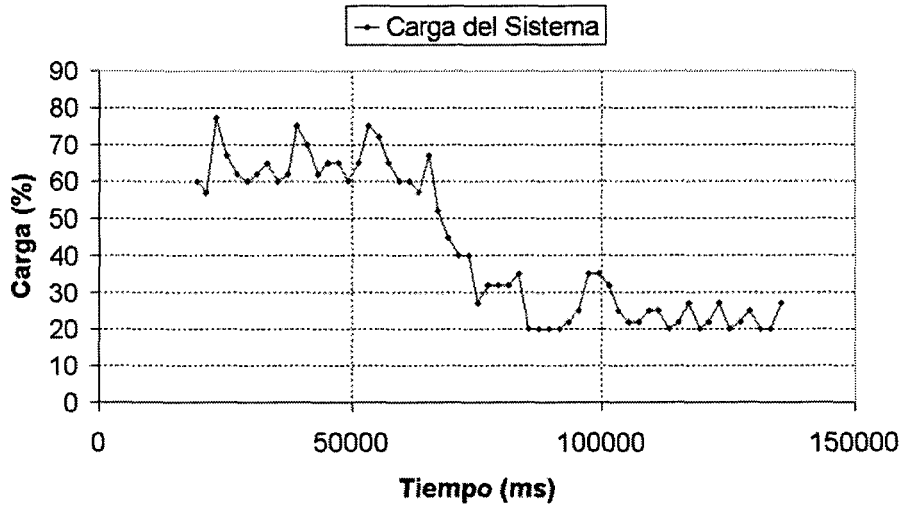

Figura 5.15: Evolución de la carga del sistema con tareas independientes.

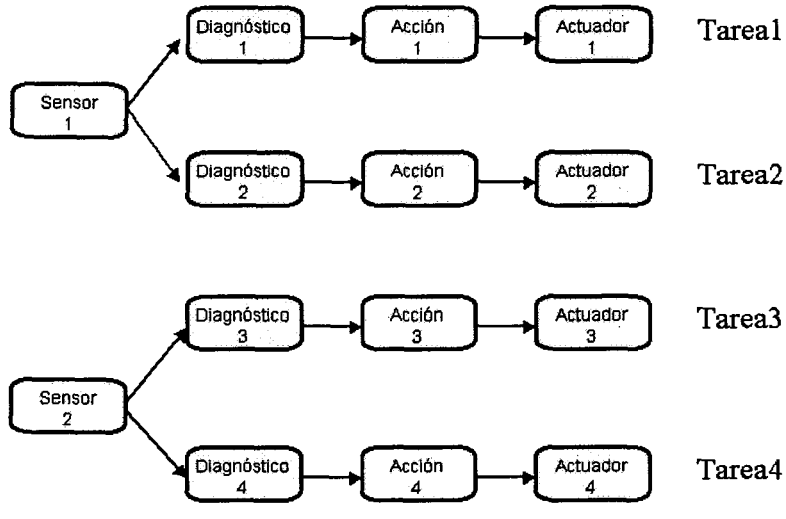

Figura 5.16: Topología de tareas dependientes.

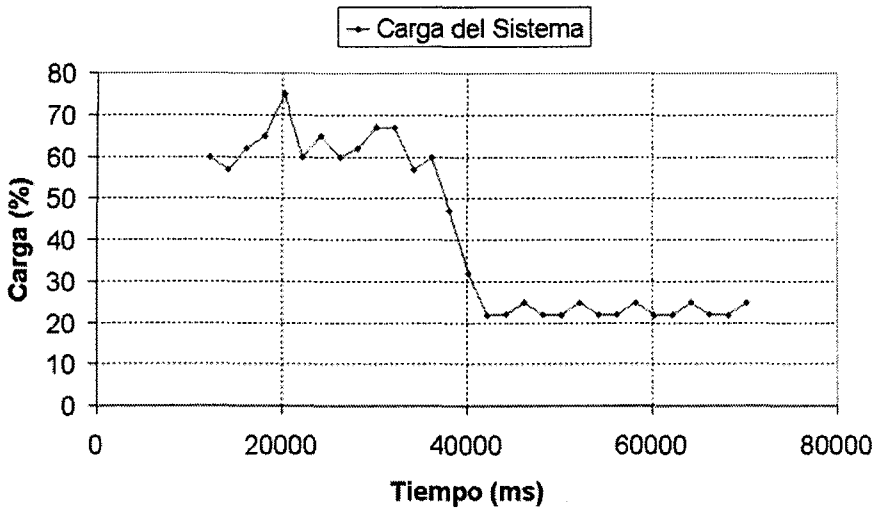

Figura 5.17: Evolución de la carga del sistema con tareas que comparten algunos módulos.

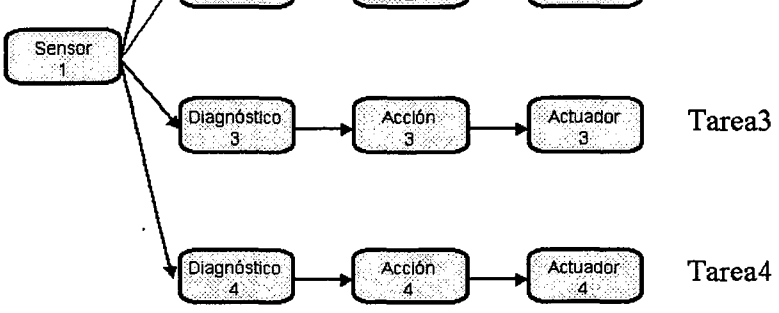

Figura 5.18: Topología de tareas altamente dependientes.

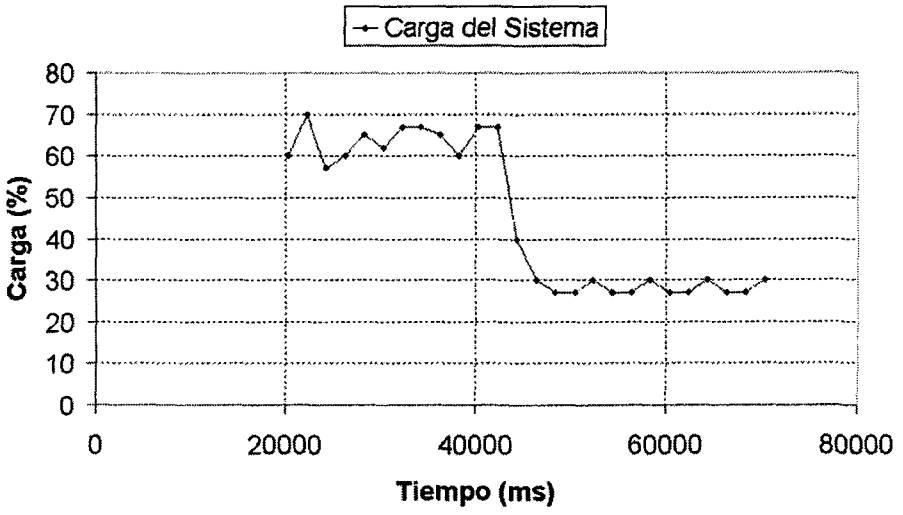

Figura 5.19: Evolución de la carga del sistema con tareas altamente dependientes.

5.4.4. Ensayos sobre distintas máquinas

En esta prueba se pretende verificar la capacidad del sistema para adaptarse a la potencia de cómputo disponible en diferentes máquinas, lo que permite que el usuario se concentre en la definición del conjunto de tareas a ejecutar con independencia de los recursos disponibles. Para ello se define una configuración de tareas fija y se analiza el resultado de las estrategias de control cuando dichas tareas se ejecutan sobre máquinas distintas.

La configuración utilizada consta de cuatro tareas, compartiendo dos de ellas un sensor y siendo las dos restantes independientes (figura 5.20). Las posibilidades de degradación se sitúan en los sensores, con dos niveles de calidad (0 y 2), y los diagnósticos, con cinco niveles de calidad (0, 1, 2, 3 y 4). Además existe un nivel de degradación en frecuencia adicional por tarea.

El problema planteado consiste en partir de una ejecución a pleno rendimiento, comandar un nivel de carga del 30 %, y posteriormente incrementarlo hasta el 50 %. Se muestran los resultados obtenidos en dos máquinas: un Pentium III a 1 GHz y un Pentium III a 600 MHz. La figura 5.21 ilustra el comportamiento de la máquina más potente, que parte de un nivel de carga del 70 % para luego alcanzar los niveles comandados. Para llegar al 30 % el sistema degrada a todos los sensores y a un diagnóstico, mientras que para recuperarse hasta el 50 % se promociona el diagnóstico degradado y

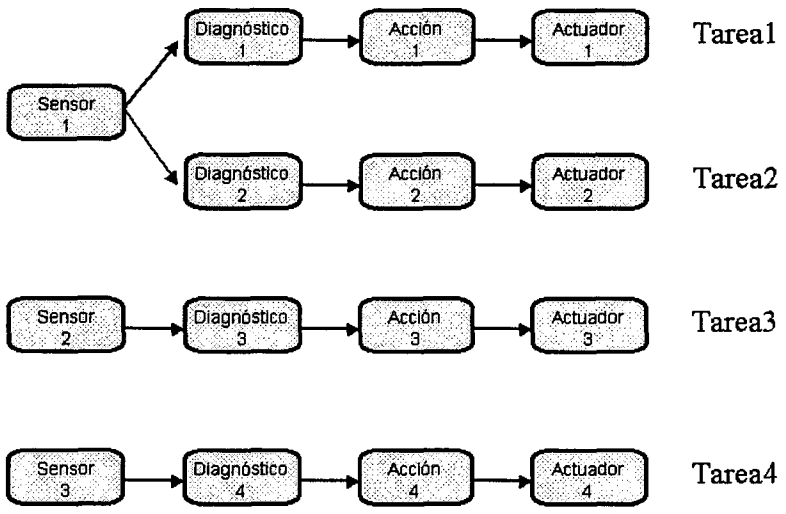

Figura 5.20: Topología de tareas para los ensayos en distintas máquinas.

uno de los sensores.

En la máquina menos potente, tal y como muestra la figura 5.22, no se alcanza el nivel comandado del 30 % de carga a pesar de que se agotan los recursos de degradación existentes. El nivel del 50 % sí es alcanzable desde la situación de degradación total promocionando algunos módulos.

La tabla 5.1 presenta las configuraciones de niveles de degradación para los módulos sensores y de diagnósticos en las que se estabiliza cada máquina para un nivel de referencia del 50 %. Como cabe esperar, los niveles de degradación alcanzados son mayores en la máquina menos potente (máquina 2).

Módulo	Sens.1	Sens.2	Diag.1	Diag.2	Diag.3	Diag.4
Máquina 1	2	2	0	0	0	0
Máquina 2	2	2	1	2	4	1

Tabla 5.1: Configuración final en cada máquina para un nivel de referencia del 50 %.

5.5. Sistemas Calibrados y no Calibrados

Los sistemas no calibrados carecen de información para orientar sus acciones de control hacia los módulos adecuados. Simplemente basan la estrategia de control en la

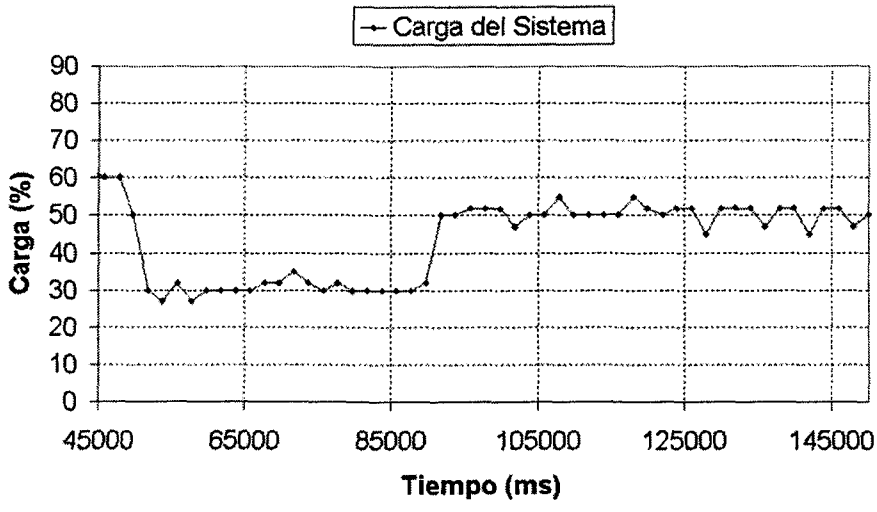

Figura 5.21: Adaptación máquina 1 (PIII, 1GHz).

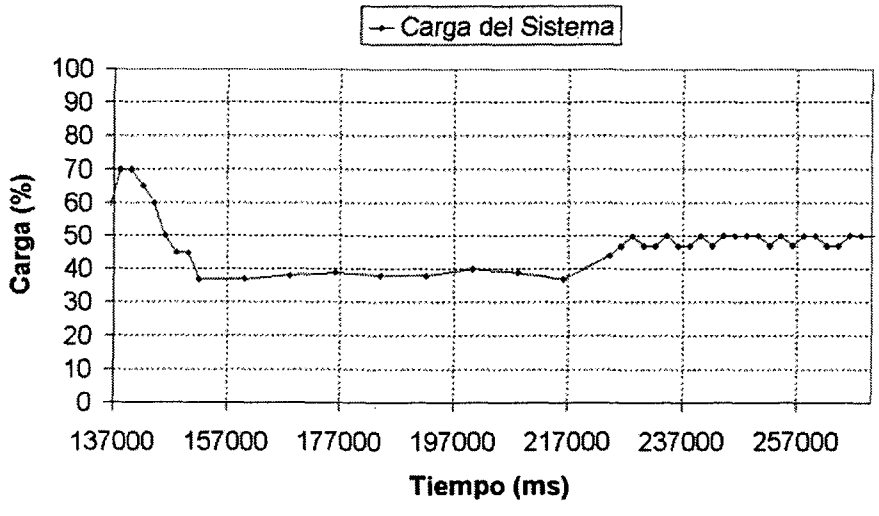

Figura 5.22: Adaptación máquina 2 (PIII, 600MHz).

información de los niveles de calidad para forzar una degradación/promoción paulatina y homogénea dentro del sistema. Desde el punto de vista del control clásico, se trata de un controlador no lineal en bucle cerrado asimilable a los del tipo encendido/apagado con histéresis. El error en régimen permanente es nulo dentro de la precisión y rango utilizados, y la ganancia debe mantenerse baja para evitar desestabilizar al sistema.

En los sistemas calibrados se aplica un control inicial en bucle abierto en base a la información previamente recogida del sistema. Si los perfiles de rendimiento son similares en todos los módulos del sistema, tanto los sistemas calibrados como los no calibrados alcanzan estados finales cercanos. Las diferencias se centran en los tiempos de respuesta, puesto que la compilación permite comandar simultáneamente múltiples módulos, mientras que en las acciones no calibradas se requiere una estrategia más conservadora y progresiva. Cuando las condiciones del sistema difieren de las estimadas durante el proceso de calibración, sin embargo, el control en bucle cerrado utilizado en los sistemas no calibrados completa la secuencia de control permitiendo alcanzar el nivel de referencia establecido. Este último es el caso de la aparición de carga no modelada en el sistema o la presencia de situaciones de error que afecten a las capacidades de cómputo del sistema.

Las diferencias de comportamiento son mayores en el caso de que los perfiles de rendimiento difieran claramente de unos módulos a otros. Un sistema calibrado concentrará sus acciones prioritariamente sobre módulos con perfiles de rendimiento relativamente planos, en los que puede alterarse significativamente la carga sin afectar tanto a la calidad, dejando de lado los perfiles con pendientes elevadas. En un sistema no calibrado, en cambio, pueden verse afectados un número de módulos innecesariamente elevado para cubrir los mismos objetivos.

En la figura 5.23 vemos tres perfiles de rendimiento (P1, P2 y P3) que comportan variaciones de carga significativamente distintas para los mismos niveles de calidad. De esta forma, una reducción de un nivel de calidad en P3 supone una variación de carga que requeriría un salto de dos niveles en P2 y que no podría alcanzarse aún abarcando todos los niveles en P1.

Para comprobar este efecto se realizó una prueba con cuatro tareas simples constituidas por un sensor y un diagnóstico cada una de ellas. Se definieron dos tipos de perfiles de rendimiento para los ocho módulos en ejecución, uno tipo "carga-variable" u horizontal y otro tipo "carga-fija" o vertical. El perfil de carga variable presenta una baja pendiente, cinco niveles de calidad entre 0.1 y 1 y variaciones significativas de carga entre niveles. El perfil de carga fija, por el contrario, tiene una pendiente elevada, con dos niveles de calidad entre 0.1 y 1, y carga prácticamente constante. En un primer test

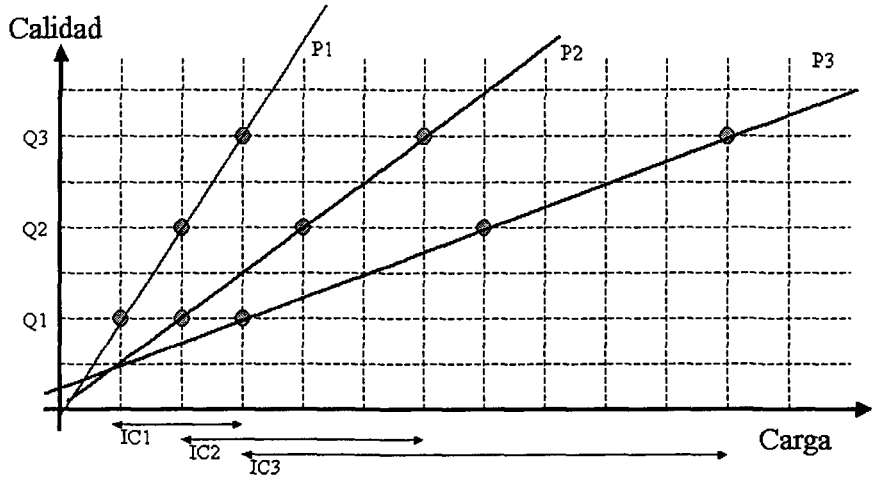

Figura 5.23: Tres perfiles de rendimiento.

se asignó a todos los módulos el perfil tipo horizontal y se comandó un valor de nivel de carga bajo. El resultado fue similar en el caso no calibrado y en el calibrado, produciéndose una degradación homogénea en todos los módulos. En un segundo test se asignó a todos los módulos el perfil tipo vertical, con la excepción de dos diagnósticos, a los que se les asignó un perfil tipo horizontal. El sistema no calibrado comandó degradaciones a siete de los ocho módulos, mientras que el sistema calibrado concentró sus acciones únicamente en los módulos de diagnóstico con perfil horizontal.

5.5.1. Control proporcional

Una forma de intentar reducir los tiempos de establecimiento en el comando de niveles de carga para sistemas no calibrados es añadir acción proporcional al bucle de control, lo que daría lugar a una configuración similar a un controlador PI en control clásico. De esta forma, el salto en los niveles de promoción/degradación de las acciones de control dependerá de la magnitud del error entre el nivel de carga actual y el nivel deseado. El peligro que se corre, como siempre ocurre en estos casos es que el sistema puede desestabilizarse o presentar oscilaciones si se emplean ganancias excesivamente elevadas.

En el siguiente ejemplo se comparan los resultados obtenidos en un sistema con

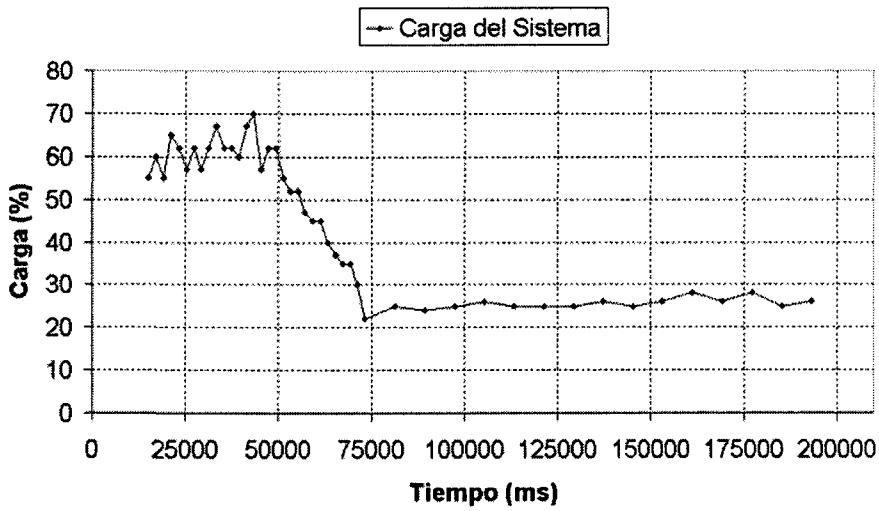

Figura 5.24: Sistema sin control proporcional.

y sin control proporcional en una prueba con cuatro tareas en la que se lleva al sistema desde el valor de carga inicial hasta un valor final de 25 %. En la figura 5.24 aparece el resultado de la ejecución sin control proporcional, de forma que la corrección del error es independiente de su magnitud. En este caso, la acción de control aplicada sobre un módulo determinado supone siempre un salto desde el nivel de degradación actual a uno contiguo.

La figura 5.25 muestra la evolución de la carga cuando se incluye control proporcional con una pequeña ganancia. En ella puede apreciarse una leve mejoría en el tiempo de respuesta, aumentando la pendiente en la zona de transición.

Finalmente, en la figura 5.26 se ilustra el comportamiento con un incremento adicional en la ganancia del controlador proporcional. El tiempo de respuesta vuelve a reducirse, si bien comienzan a apreciarse pequeñas señales de oscilación que se intensificarán si se emplean ganancias mayores.

5.6. Una Aplicación Real

Como experimento final, se presenta una aplicación real de seguimiento desarrollada sobre el sistema. El objetivo era construir una aplicación con múltiples estados en los que se ejecutaran concurrentemente tareas de diferente nivel de prioridad con capa-

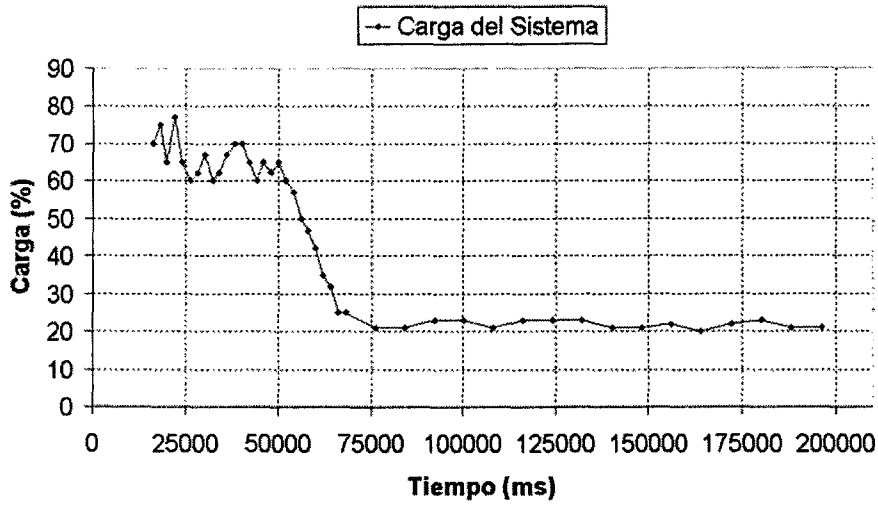

Figura 5.25: Sistema con control proporcional y ganancia pequeña.

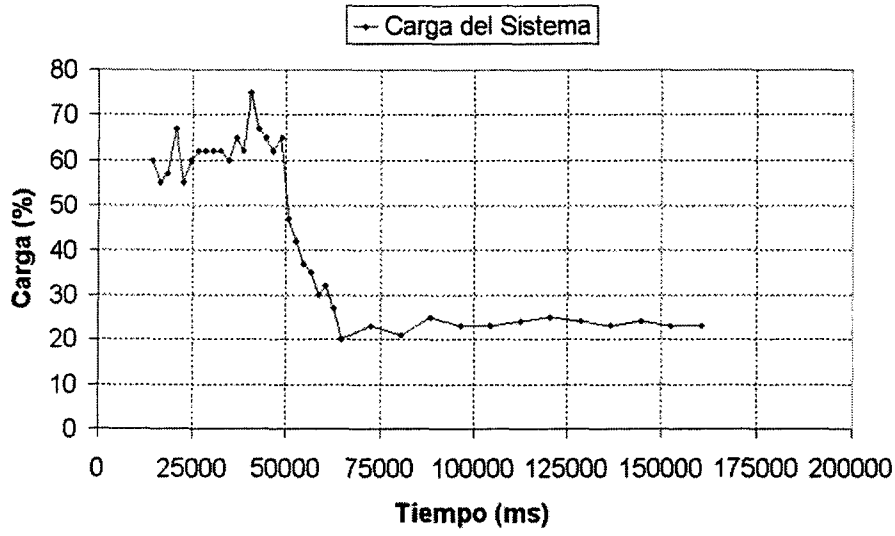

Figura 5.26: Sistema con control proporcional y ganancia mayor.

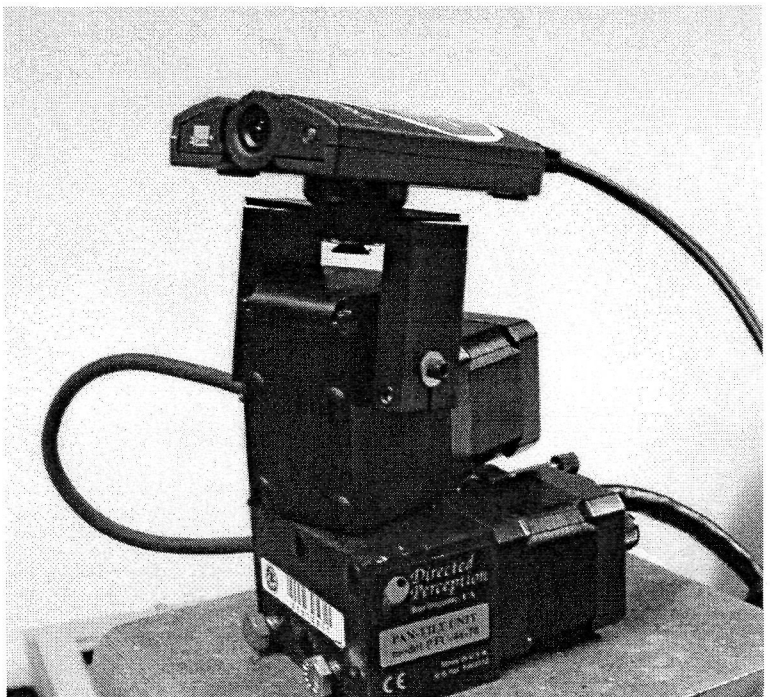

Figura 5.27: Cámara instalada en la unidad pan-tilt.

cidad de adaptación y mostrar cómo se puede definir una aplicación real en términos de los objetos propuestos por la arquitectura.

5.6.1. Planteamiento del problema

El sistema debe ser capaz de seguir visualmente un objeto al tiempo que realiza cómputos complementarios de extracción de características del entorno con una prioridad menor. Deben incorporarse comportamientos de recuperación en caso de pérdida del objetivo y mecanismos de adaptación. Se dispone de un sistema de visión activa compuesto por una cámara orientable con dos grados de libertad (ver figura 5.27).

5.6.2. Estados, tareas y módulos

La aplicación consta de tres estados: "seguimiento", "búsqueda activa" y "búsqueda pasiva" (figura 5.28). En el estado de "seguimiento" se ejecuta la tarea de seguimiento visual basado en correlación de patrones [Guerra-Artal, 2002] junto con dos tareas auxiliares de menor prioridad que se encargan de intentar caracterizar el patrón mediante la extracción de características de color y de varianza sobre la zona de la imagen centrada

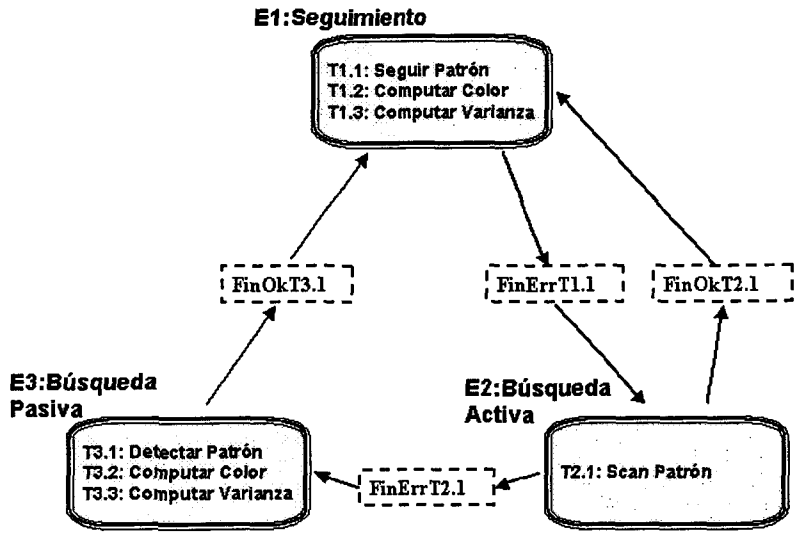

Figura 5.28: Estados y transiciones de la aplicación real de seguimiento.

en el patrón. Cuando el seguimiento falla, se transita al estado "búsqueda activa", en el que se lanza una única tarea que realiza una inspección del entorno en busca del patrón perdido. Esta inspección efectúa pequeños desplazamientos en torno a la última posición en que fue detectado el objetivo, analizando en cada una el resultado de la correlación para comprobar si es posible reanudar el seguimiento.

Del estado de búsqueda activa se puede salir con éxito si se recupera el patrón, en cuyo caso se retorna al estado de seguimiento, o con fracaso, si se completa el barrido sin detección positiva. En este último caso, el sistema transita al estado "búsqueda pasiva", en el que se ejecuta una tarea de detección sobre las imágenes servidas por la cámara en posición fija. Junto a esta tarea, y simplemente a efectos de analizar el funcionamiento de las políticas de adaptación, se ejecutan las tareas menos prioritarias de cómputo de propiedades. El sistema permanece en este estado hasta que se realiza una detección positiva, momento en el que se reinicia el seguimiento. La figura 5.28 muestra el diagrama de estados, tareas y transiciones resultante.

La aplicación consta de un total de cinco tareas distintas y 11 módulos: un sensor, dos actuadores, cinco acciones y tres diagnósticos. El sensor es el módulo encargado de encapsular al sensor físico, que en este caso es una webcam "HomeConnect" de 3Com. Incorpora funciones de inicialización, configuración, captura de imágenes y finalización. El módulo actuador principal controla a un efector físico consistente en una unidad PTU-

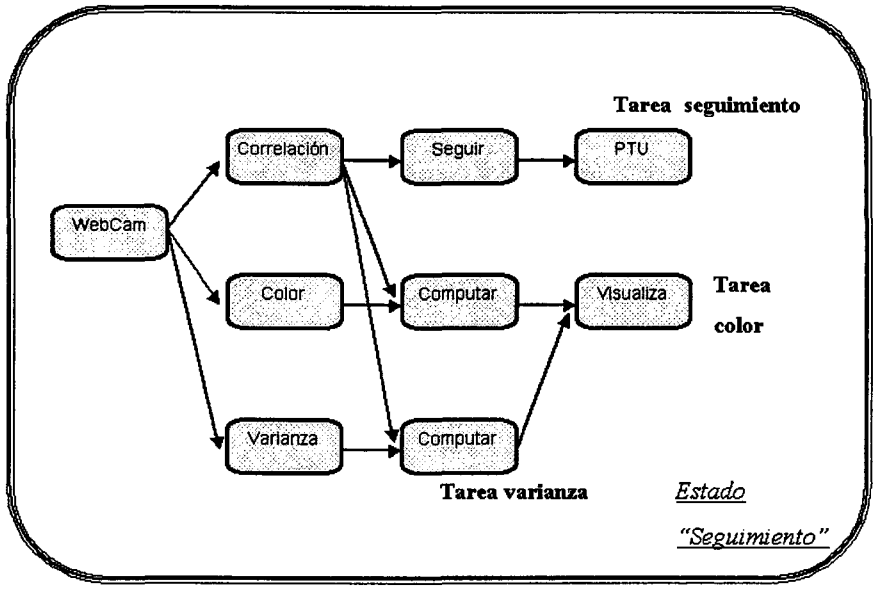

Figura 5.29: Topología de módulos en el estado de seguimiento.

46-70 de Directed Perception, sobre la que va montada la cámara. Posee, además de las funciones de inicialización, configuración y finalización, comandos en modo posición, velocidad y parada de emergencia. Como actuador adicional virtual está un visualizador de imágenes en el que volcar el resultado del procesamiento.

Los diagnósticos implementados son un procedimiento de detección de patrones por correlación, extracción de color y cómputo de la varianza, todos ellos aplicados sobre las imágenes servidas por el sensor de cámara. Las acciones disponibles son el seguimiento, la búsqueda, la detección y la monitorización o cómputo. El seguimiento y la búsqueda envían comandos al efector pan/tilt, mientras que el resto se limita a volcar los resultados sobre el efector de visualización.

Todas las transiciones se activan a partir de la finalización con éxito o con error de diferentes tareas. Este control se basa en el funcionamiento del módulo de diagnóstico de detección de patrones, que suministra una medida de la calidad de la detección.

La figura 5.29 muestra la topología de los módulos que forman parte de las tareas del estado de seguimiento. La configuración correspondiente a los estados de búsqueda se ilustra en la figura 5.30.

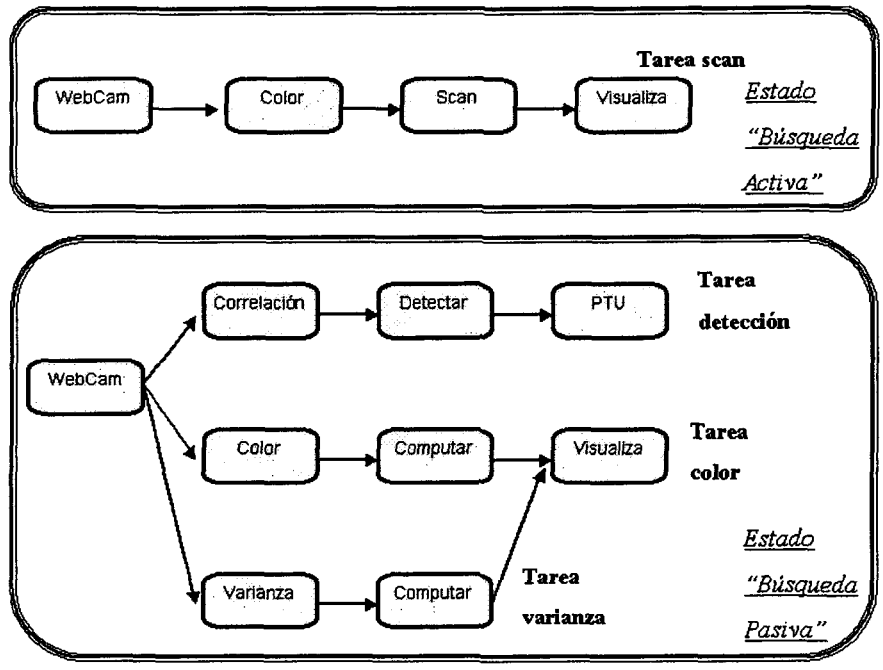

Figura 5.30: Topología de módulos en los estados de búsqueda.

5.6.3. Adaptación computacional

Los módulos implementados incorporan diferentes mecanismos de modificación de la carga computacional. El sensor de cámara puede servir imágenes a los módulos de diagnóstico en cuatro resoluciones diferentes. El cálculo de la correlación puede reducir su demanda computacional mediante dos métodos distintos: limitando el área de búsqueda o reduciendo el tamaño del patrón a localizar. El resto de los diagnósticos pueden asimismo computarse submuestreando la imagen de entrada para acortar el tiempo de procesamiento, configurando tres niveles de degradación.

Como grado de libertad adicional está la modificación de la frecuencia de operación de las diferentes tareas. Inicialmente se toman como periodos 120 milisegundos para la tarea de seguimiento y 500 milisegundos para las restantes.

La mayor carga computacional se produce en el estado de seguimiento, donde las tareas de menor prioridad deben degradarse para verificar la frecuencia impuesta. Cuando se transita a los estados de búsqueda pasiva, la demanda de recursos es menor, y las tareas secundarias pueden recuperarse. En el estado de búsqueda activa, todos los esfuerzos se centran en la tarea de recuperación, eliminándose las secundarias.

Las siguientes gráficas ilustran los mecanismos de adaptación aplicados a la tarea de seguimiento. En la figura 5.31 se muestra la evolución del tiempo de procesamiento

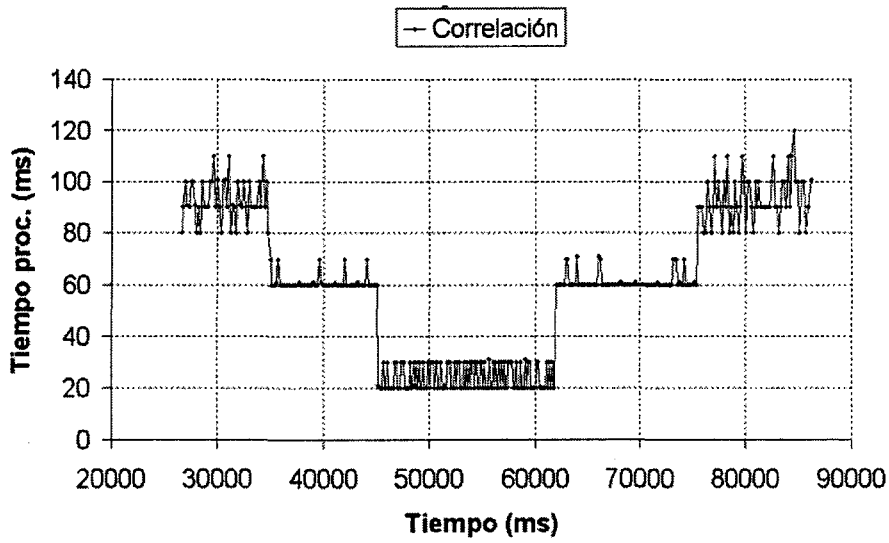

Figura 5.31: Evolución del tiempo de procesamiento en distintos niveles de degradación.

en el diagnóstico de correlación cuando se fuerza una reducción de los recursos disponibles. Inicialmente la tarea de seguimiento opera sobre imágenes a resolución completa (320x240), pasando posteriormente en el primer nivel de degradación a una resolución de 296x216, y en el segundo nivel a 240x180. Posteriormente se recupera hasta volver a trabajar con imágenes a resolución máxima.

La gráfica 5.32 presenta la contribución a la carga global del sistema en los diferentes niveles de degradación, evaluada sobre un intervalo de 500 milisegundos.

5.6.4. Ejemplos de ejecución

A continuación se presentan dos ejemplos del sistema en ejecución. En el primero, el sistema transita entre los estados de seguimiento y búsqueda activa. Cuando el seguimiento falla se pasa al estado de búsqueda activa, en el que se realiza un barrido por el entorno con vistas a recuperar el objetivo. En este ejemplo el sistema siempre recupera el objetivo antes de completar el barrido. La figura 5.33 muestra un cronograma que indica qué estados se encuentran activos en cada instante. En la figura 5.34 se muestra la evolución de la carga del sistema a medida que se van produciendo las transiciones.

En el segundo ejemplo se ilustra el caso en que el sistema, una vez perdido el objetivo, completa la búsqueda activa sin éxito, por lo que se transita al estado

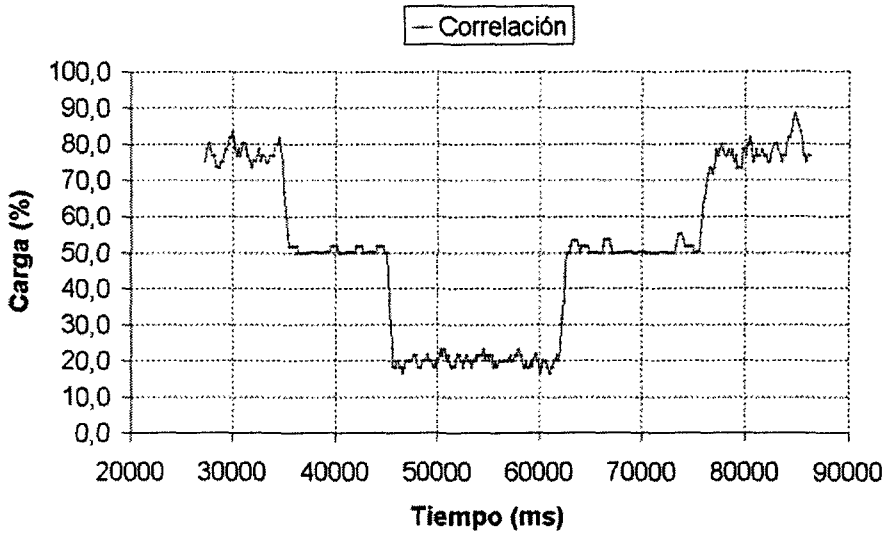

Figura 5.32: Evolución de la contribución a la carga global de la tarea de seguimento.

de búsqueda pasiva. El ejemplo termina con la recuperación del objetivo y la vuelta al estado de seguimiento. La gráfica 5.35 presenta el cronograma de actividad de los diferentes estados, mientras que en la figura 5.36 se muestra la evolución de la carga del sistema a lo largo de la ejecución.

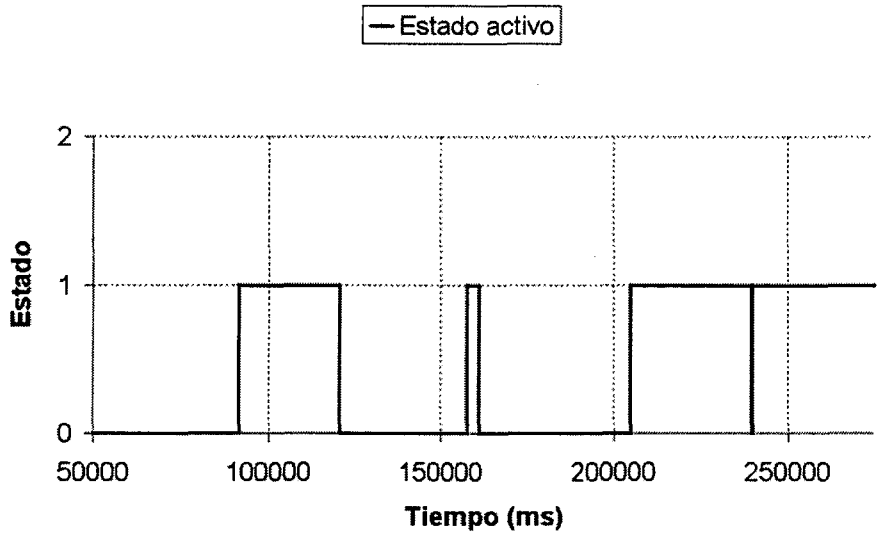

Figura 5.33: Estados activos en función del tiempo para el ejemplo 1.

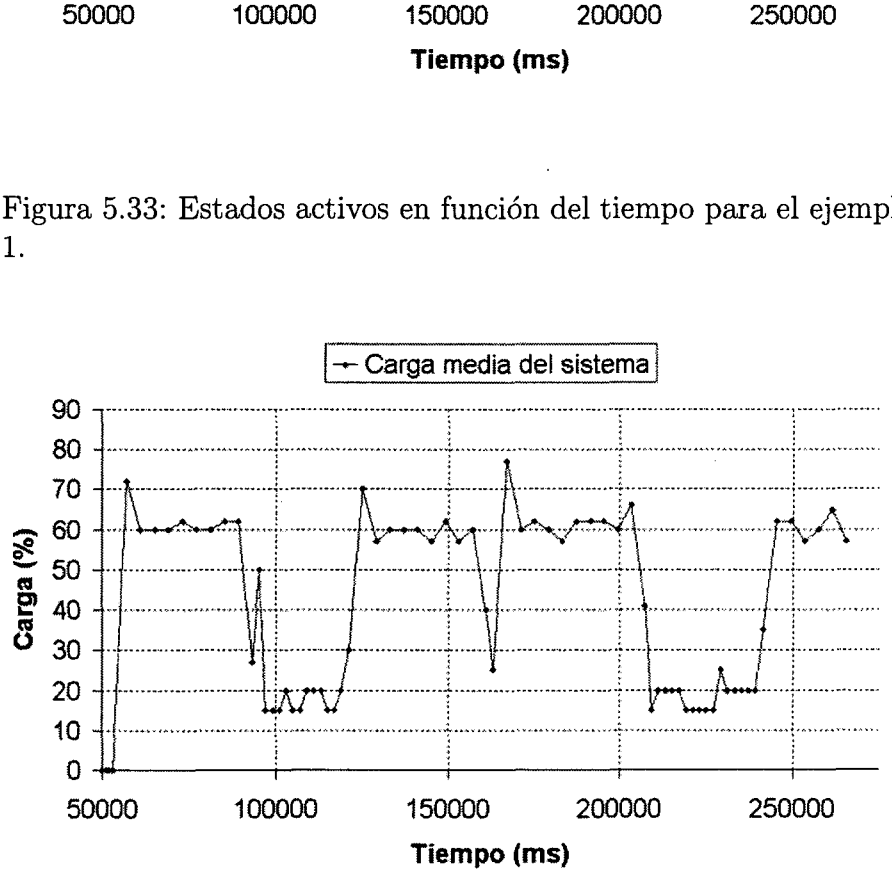

Figura 5.34: Evolución de la carga del sistema para el ejemplo 1.

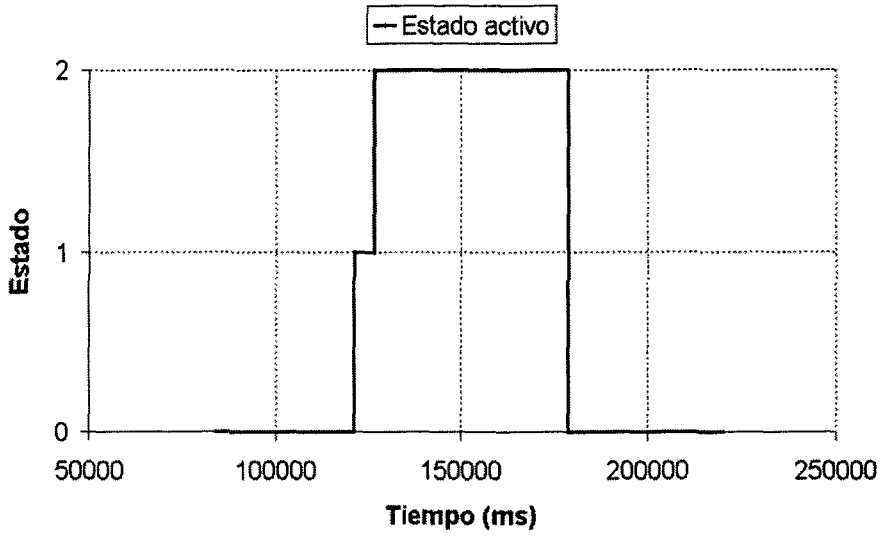

Figura 5.35: Estados activos en función del tiempo para el ejemplo 2.

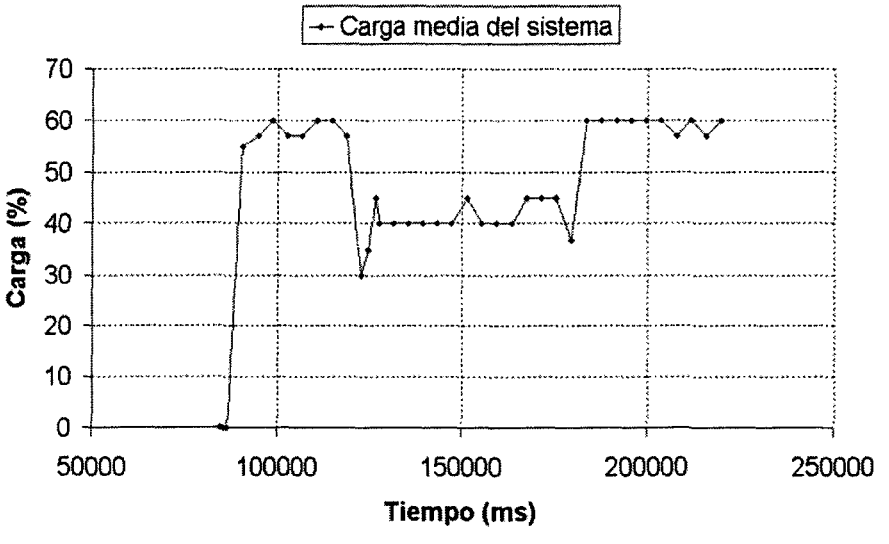

Figura 5.36: Evolución de la carga del sistema para el ejemplo 2.

Capítulo 6

Conclusiones y Perspectivas de Desarrollo

En este capítulo se presentan las conclusiones de este trabajo de tesis, así como las posibles líneas de desarrollo que pueden explorarse en el futuro.

6.1. Conclusiones

6.1.1. Motivación

La tarea de diseñar e implementar sistemas percepto-efectores robustos requiere contribuciones procedentes de las diferentes disciplinas que confluyen en este área: Sistemas Operativos, Control Automático, Inteligencia Artificial. Aunque a corto plazo puedan alcanzarse rendimientos significativos con aplicaciones ad hoc, el éxito en el largo plazo pasa por el uso de metodologías de desarrollo que permitan la construcción de sistemas robustos, capaces y adaptables. En este sentido son deseables aspectos como la promoción del diseño modular y la reutilización del código en aras de acortar los tiempos de puesta en funcionamiento de nuevas aplicaciones y de facilitar el mantenimiento y modificación de las existentes. En la etapa de diseño es importante disponer de un modelo de referencia claro que, sin ser excesivamente restrictivo, defina un conjunto de elementos modulares que permitan estructurar las aplicaciones y facilitar su desarrollo. Durante la implementación son deseables mecanismos que simplifiquen la programación de nuevos elementos y la integración.

En tiempo de ejecución, un sistema percepto-efector está sometido generalmente a fuertes restricciones temporales y de recursos, las cuales deben ser controladas ade-

cuadamente para permitir que el sistema se adapte dinámicamente a los cambios del entorno. En caso contrario, la ausencia de control puede llegar a provocar una degradación catastrófica que limite seriamente la funcionalidad del sistema. Esto es especialmente relevante si se tiene en cuenta la naturaleza de los contextos de ejecución en que operan normalmente este tipo de sistemas. Se trata de sistemas de tiempo real blando con múltiples objetivos, los cuales cambian rápidamente y dan lugar a la activación de diferentes estados y tareas. Deben buscarse, pues, mecanismos de control que garanticen la estabilidad y reduzcan al mínimo la duración de las situaciones transitorias que se presenten. Incluso dentro de la misma configuración de estados y tareas, deben proveerse estrategias de protección de las tareas más prioritarias y un trato homogéneo para las que compartan el mismo nivel de prioridad; en suma, se persigue alcanzar una adaptación suave y ordenada del sistema.

La adaptación de bajo nivel es un aspecto frecuentemente ignorado en el diseño de arquitecturas de control de este tipo de sistemas. En muchas ocasiones, los aspectos adaptativos se abordan una vez concluidos los desarrollos arquitectónicos, con lo que se corre el riesgo de incluir inadvertidamente limitaciones que trunquen determinadas soluciones o, en el mejor de los casos, condicionen su eficiencia. En otros casos, ni siquiera se consideran las limitaciones hardware desde el punto de vista de los recursos computacionales, con lo que puede comprometerse la seguridad o estabilidad del sistema si se llega de forma no anticipada a situaciones de falta de recursos.

6.1.2. Sistema propuesto

En esta tesis se propone una organización modular para el diseño de sistemas percepto-efectores. Esta estructura permite definir diferentes arquitecturas multinivel distribuidas basadas en un conjunto reducido de objetos: sensores para la captación de los datos, diagnósticos para su procesamiento, acciones para evaluar los resultados del cómputo y generar comandos como respuesta y actuadores para hacer operativos los comandos. La configuración interna es común a todos los módulos, y está constituida por la interconexión de tres unidades funcionales: la unidad BU encargada del procesamiento, la unidad TD para el control y la unidad COM que gestiona las comunicaciones. Se dispone de un lenguaje de programación simple, basado en un conjunto reducido de instrucciones, para la implementación de cada tipo de módulo.

Se presenta una organización del cómputo basada en estados y tareas para el control de la misión a más alto nivel. Las tareas se componen mediante una combinación de acciones y diagnósticos, siendo posible la introducción de valores de frecuencia de

funcionamiento y prioridad a nivel de tarea. Esto permite caracterizar la operación de las tareas en tiempo de ejecución.

6.1.3. Adaptación computacional

Se presenta un esquema de adaptación de bajo nivel en un sistema percepto-efector que permite ajustar los recursos demandados a las capacidades computacionales existentes. Se analizan los diferentes problemas de control planteables, las acciones de control disponibles y su integración en políticas de control que las coordinen. De esta forma, se consigue que el sistema degrade y promocione en función de los recursos disponibles de manera controlada. Se ha tratado tanto el caso de los sistemas calibrados como los no calibrados.

Se incluyen modelos para la estimación de la carga del sistema, a nivel local y global. En el caso de distribución del cómputo se definen modelos para estimar el tiempo de procesamiento en los diferentes esquemas de balanceo de carga. Se proponen además medidas para la caracterización de aspectos del sistema como su nivel de calibración o las capacidades de adaptación.

Dentro de la estrategia de control, se arbitran políticas tanto locales como globales, las cuales se ofrecen como una utilidad al diseñador. En las definiciones de las tareas, pueden añadirse valores de tolerancia a las violaciones temporales, con el propósito de que sean utilizados en las políticas de control. A nivel de módulos es posible incluir información sobre los recursos de degradación disponibles, en caso de existir.

Se propone la inclusión de mecanismos de aprendizaje basado en casos y se presenta una configuración de control integrado que combina el aprendizaje con los procesos de autocalibración en el esquema de control global del sistema.

6.1.4. Experimentos

Se han realizado un conjunto de experimentos dentro de los siguientes apartados: simulación de mecanismos y políticas de control, sistemas calibrados y no calibrados, pruebas de distribución de la carga computacional, pruebas con diagnósticos e implementación de una aplicación real.

Para las pruebas de simulación se ha desarrollado un entorno que permite cargar un conjunto de estados y tareas, ordenar su ejecución y ensayar las diferentes estrategias de control. Se ha comprobado la eficacia de los bucles de control tanto para la distribución

temporal como en las violaciones de los periodos de funcionamiento a partir de medidas locales de la carga. También se ha demostrado la validez de los esquemas de adaptación a la hora de fijar el nivel de carga del sistema a partir de medidas globales.

Las comparaciones entre los sistemas calibrados y no calibrados se han basado en un simulador de compilación de tareas. Se ha verificado que en situaciones de perfiles de rendimiento homogéneos entre los diferentes módulos, tanto la compilación como el control no calibrado conducen a resultados similares. En presencia de perfiles claramente diferenciados, sin embargo, se hace necesaria la información de calibración para orientar convenientemente las acciones de control.

Para los ensayos relativos a la distribución del cómputo se ha desarrollado una aplicación que permite ensayar diferentes esquemas de balanceo de carga sobre un conjunto de estaciones de trabajo. Se ha demostrado la necesidad de recurrir a esquemas adaptativos como única solución válida en situaciones de niveles de carga cambiantes, y siempre que las exigencias computacionales sean suficientemente elevadas para compensar los overheads producidos. Se ha comprobado asimismo la validez de los modelos de predicción para el tiempo de ejecución propuestos.

Se han diseñado un conjunto de experimentos con el objetivo de probar la validez de la descomposición modular propuesta a nivel de procesamiento en un dominio concreto, como es el de la segmentación de imágenes. Para ello se han implementado una batería de diagnósticos de procesamiento tanto a nivel de pixels como de segmentos. Los resultados obtenidos muestran que es posible aplicar con éxito este enfoque.

Por último, se ha ensayado una implementación real para una aplicación de seguimiento en visión activa. Esta implementación se ha planteado como un primer test para evaluar los beneficios que reporta la organización modular propuesta. Una validación más rigurosa sólo podrá emitirse cuando el número de implementaciones desarrolladas, su modificación y mantenimiento así lo permitan.

6.1.5. Principales aportaciones

Como principales aportaciones de esta tesis cabe destacar las siguientes:

1. **La propuesta de organización modular:** La configuración interna de cada módulo, con sus diferentes unidades funcionales BU, TD y COM, permite identificar claramente los flujos de procesamiento y de control. La implementación de nuevos componentes se ve facilitada por la definición de secciones separadas para las diferentes etapas de inicialización, ciclo de procesamiento y finalización. El

usuario desarrollador simplemente debe completar zonas de código bien delimitadas, siendo los restantes elementos incluidos de forma automática por parte del sistema.

El planteamiento propuesto promueve la reutilización del código gracias a la existencia de interfaces bien definidas. En la base de conocimiento se recogen todos los aspectos relativos a la compatibilidad entre las entradas y las salidas de los diferentes tipos de módulos. De esta manera se garantiza una interconexión segura entre los módulos cuando éstos se combinan para configurar tareas.

2. **El lenguaje de composición de tareas:** La definición de tareas empleando combinaciones de acciones y diagnósticos a partir de la información almacenada de forma jerárquica en la base de conocimiento de objetos y tipos de objetos. Esta organización favorece una interpretación más directa de la misión desempeñada por el sistema.

3. **La propuesta integrada de arquitectura y control:** Desde un principio se considera la presencia de mecanismos de control locales en cada módulo. Ello permite dotar al sistema de un elevado grado de autonomía a bajo nivel, de forma que son implementables de forma directa esquemas de autorregulación y técnicas de recuperación de errores que luego pueden extenderse a nivel global. Las soluciones de adaptación se benefician de este hecho para combinar políticas de control locales que permiten mejorar los tiempos de respuesta y reducir los overheads, con políticas globales que ofrecen una mejor selección de los objetivos del control. La idea que subyace es la de que la robustez de un sistema se ve claramente beneficiada por el hecho de que los elementos que lo integran ya presenten un comportamiento robusto.

   Estos mecanismos de control están parametrizados de forma que el usuario puede seleccionar la configuración deseada para conseguir un comportamiento más robusto de su aplicación. Es posible incluso la activación selectiva de los mecanismos de manera que sólo afecten a un subconjunto de las tareas activadas, permaneciendo el resto inalteradas. Aunque se ofrece una política de control por defecto, la estructuración del código y la integración de los elementos adaptativos en el ciclo de operación básico permite su modificación de una forma simple.

4. **Estudio comparativo de sistemas calibrados y no calibrados:** Se analiza tanto el caso de los sistemas calibrados como los no calibrados. Se presentan alternativas para la elaboración de estrategias de control a medida que el sistema va transitando desde el estado de no calibrado al estado calibrado.

5. **Incorporación del aprendizaje:** La consideración de los mecanismos de aprendizaje basado en casos aporta un doble beneficio: por un lado puede emplearse como medio para reducir los tiempos de estabilización en el control no calibrado, y por otro, a fin de evitar la ejecución de los costosos algoritmos de compilación en el caso de sistemas calibrados.

6. **Resultados experimentales:** Los experimentos realizados constituyen un material valioso como evaluación de las propuestas planteadas a nivel de simulación y sobre sistemas reales, sirviendo al mismo tiempo de guía para posibles mejoras y ampliaciones. El análisis de factores permite orientar las decisiones de control adecuadamente, en función de las características propias de cada entorno. Las pruebas efectuadas para cada política de control han hecho posible verificar su eficacia, así como comprobar los beneficios aportados al facilitar el diseño de esquemas de coordinación. Por último, los ensayos sobre sistemas reales han proporcionado una primera realimentación de los aspectos ligados a la puesta en práctica de las ideas planteadas.

6.2. Líneas Futuras de Desarrollo

Como prolongación de esta tesis pueden plantearse múltiples extensiones. Algunas de las líneas de trabajo más relevantes serían las que se enumeran a continuación:

1. **Demostradores:** Se tiene prevista la construcción de una serie de demostradores completos que permitan validar de forma extensiva tanto la metodología de desarrollo como los mecanismos de adaptación y aprendizaje. Algunos ejemplos serían el diseño e implementación de aplicaciones en robótica móvil para conseguir una navegación robusta o la construcción de un sistema de visión multiobjetivo.

2. **Librería de módulos:** El desarrollo de los demostradores implica un elevado esfuerzo de implementación, pues requerirá que se implanten un amplio número de comportamientos y capacidades, tales como la navegación, evitación de obstáculos, localización, planificación, identificación a nivel de objetos, etc. La disponibilidad de estos módulos permitirá sintetizar diferentes sistemas y servirá para evaluar si la metodología de diseño favorece o no un diseño incremental y la reutilización del software, como apuntan los resultados preliminares que en este sentido se han obtenido en esta tesis.

3. **Aprendizaje:** Otra línea interesante a desarrollar es el estudio en profundidad de las propuestas realizadas en este trabajo sobre adaptación y aprendizaje aplicadas sobre sistemas reales. Deberán recogerse aspectos tales como la integración de políticas o el análisis de la influencia sobre el rendimiento del sistema.

4. **Recuperación de errores:** Posibilitada por el diseño de los módulos basado en una clara separación entre computación, control y comunicaciones, se plantea la investigación de mecanismos de diagnóstico y recuperación de fallos en sistemas modulares. Algunos aspectos a tratar en este apartado serían las estrategias de localización del origen de las excepciones, el control de los "fenómenos de avalancha" debidos a las múltiples detecciones locales o la integración de estrategias de recuperación y continuidad sobre la base de un diseño modular reutilizable.

5. **Adaptación de alto nivel:** Como extensión de los esquemas de control de bajo nivel son planteables estrategias de adaptación de más alto nivel. En este sentido se abordaría la composición dinámica de nuevas tareas a partir de diagnósticos y acciones primitivas en función de los recursos disponibles. Tiene cabida también aquí el desarrollo de planificadores de alto nivel que reconsideren la configuración completa de tareas en ejecución cuando ésta no sea sostenible.

6. **Modificadores lingüísticos:** En la línea de mejora de la capacidad descriptiva del lenguaje está la posibilidad de modular tareas mediante modificadores lingüísticos. Son planteables términos intensificadores o atenuadores para el cómputo de los diagnósticos (cuánto) que afecten a los umbrales de detección, localizadores espaciales (dónde) que focalicen o restrinjan el ámbito del procesamiento, o delimitadores temporales (cuándo) que determinen el inicio y fin de la actividad. Análogamente, se puede pensar en modificadores que alteren el funcionamiento de las acciones en términos de la intensidad promedio de los comandos que dirigen a los actuadores, por ejemplo.

7. **Instrucción basada en lenguaje natural:** Una línea prometedora es la constituida por la exploración de las técnicas de instrucción basada en el lenguaje natural (IBL). Se trataría de aprovechar la composición de tareas en términos próximos al lenguaje natural que se propone en este trabajo para implementar mecanismos de instrucción vocal.

8. **Herramientas:** El progreso adecuado de las líneas propuestas pasa necesariamente por la construcción de herramientas para la monitorización de las tareas y estados en ejecución, incluyendo medidas temporales y registro de eventos; y de

forma conjunta, la generación de utilidades de depuración para detectar errores de programación y acortar los tiempos de desarrollo. También puede ser interesante disponer de aplicaciones para el análisis off-line de los resultados de la ejecución.

Referencias

Alami, R., Aguilar, L., Bullata, H., Fleury, S., Herrb, M., Ingrand, F., Khatib, M. y Robert, F. (1995). 'A general framework for multi-robot cooperation and its implementation on a set of three hilare robots.' En '4th International Symposium on Experimental Robotics (ISER'95),' págs. 26–39. Stanford, CA, USA.

Alami, R., Chatila, R., Fleury, S., Ghallab, M. y Ingrand, F. (1998). 'An architecture for autonomy.' *International Journal of Robotics Research (Special Issue on Integrated Architectures for Robot Control and Programming)*, **17**(4), págs. 315–337.

Arkin, R. C. (1989). 'Motor schema-based mobile robot navigation.' *The International Journal of Robotics Research*, **8**(4), págs. 92–112.

Arkin, R. C. (1992). 'Homeostatic control for a mobile robot: Dynamic replanning in hazardous environments.' *The International Journal of Robotics Research*, **9**(2), págs. 197–214.

Arkin, R. C. (1998). *Behavior-Based Robotics.* The MIT Press.

Arkin, R. C. y Balch, T. R. (1997). 'AuRA: principles and practice in review.' *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, **9**(2-3), págs. 175–189.

Bajscy, R. y Large, E. (1999). 'When and where AI will meet robotics?' *AI Magacine*, **20**(3), págs. 57–65.

Berry, G. (2000). *The Foundations of Esterel.* MIT Press.

Boddy, M. y Dean, T. (1994). 'Decision-theoretic deliberation scheduling for problem solving in time-constrained environments.' *Artificial Intelligence*, **67**(2), págs. 245–286.

Bonasso, R. P., Firby, J., Gat, E., Kortenkamp, D., Miller, D. P. y Slack, M. G. (1997). 'Experiences with an architecture for intelligent, reactive agents.' *Journal of Experimental & Theoretical Artificial Intelligence*, **9**(2/3), págs. 237–256.

Bondavalli, A., Stankovic, J. y Strigini, L. (1993). 'Adaptable fault tolerance for real-time systems.' En 'Proceedings of Predictably Dependable Computing Systems,' págs.–. San Francisco, CA, USA.

Braitenberg, V. (1984). *Vehicles. Experiments in Synthetic Psychology.* The MIT Press.

Brooks, R. (1986). 'A robust layered control system for a mobile robot.' *IEEE Journal of Robotics and Automation*, **2**(1), págs. 14–23.

Brooks, R. (1990). 'The Behavior Language: User's guide.' Informe Técnico AIM-1227, MIT Artificial Intelligence Lab.

Brooks, R. A. (1991). 'Intelligence without representation.' *Artificial Intelligence*, **1**(47), págs. 139–159.

Buttazzo, G. C., Lipari, G. y Abeni, L. (1998). 'Elastic task model for adaptive rate control.' En 'IEEE Real-Time Systems Symposium (RTSS'98),' págs. 286–295. Madrid, España.

Cabrera-Gámez, J. (1994). *Sistema Basado en Conocimiento para Segmentación de Imágenes. Desarrollos y Aplicaciones.* Tesis Doctoral, Dpto.de Informática y Sistemas, Univ. de Las Palmas de Gran Canaria.

Cabrera-Gámez, J., Domínguez-Brito, A. C. y Hernández-Sosa, D. (2001). 'CoolBOT: A component-oriented programming framework for robotics.' *Lecture Notes in Computer Science*, **2238**, págs. 285–305.

Cabrera-Gámez, J., Hernández-Sosa, D., Domínguez-Brito, A., Castrillón-Santana, M., Lorenzo-Navarro, J., Isern-González, J., Guerra-Artal, C., Pérez-Pérez, I., Falcón-Martel, A., Hernández-Tejera, M. y Méndez-Rodríguez, J. (2000). 'Experiences with a museum robot.' En 'Workshop on Edutainment Robots,' Bonn, Alemania.

Cassandras, C. (1993). *Discrete Event Systems. Modelling and Performance Analysis.* Irwin and Aksen.

Christensen, H. I. y Pirjanian, P. (1997). 'Theoretical methods for planning and control in mobile robotics.' En 'International Conference on Conventional and Knowledge Based Intelligent Electronic Systems,' págs. 32–38. Adelaida, Australia.

Clark, R. J., Arkin, R. C. y Ram, A. (1992). 'Learning momentum: On-line performance enhancement for reactive systems.' En 'IEEE International Conference on Robotics and Automation,' págs. 111–116. Niza, Francia.

Coste-Manière, E. y Simmons, R. (2000). 'Architecture, the backbone of robotic systems.' En 'Proceedings of the IEEE International Conference on Robotics & Automation,' págs. 67–72. San Francisco, CA, USA.

Coste-Manière, E. y Turro, N. (1997). 'The MAESTRO language and its environment : Specification, validation and control of robotic missions.' En 'Proceedings of IROS 97, vol. 2,' págs. 836–841. Grenoble, Francia.

D'Ambrosio, B. (1989). 'Resource bounded-agents in an uncertain world.' En 'Proceedings of the Workshop on Real-Time Artificial Intelligence Problems,' Detroit, MI, USA.

Dean, T. y Boddy, M. (1988). 'An analysis of time-dependent planning.' En 'Proceedings of the 7th National Conference on Artificial Intelligence, AAAI,' págs. 49–54. St. Paul, MN, USA.

Domínguez-Brito, A., Cabrera-Gámez, J., Hernández-Sosa, D., Castrillón-Santana, M., Lorenzo-Navarro, J., Isern-González, J., Guerra-Artal, C., Pérez-Pérez, I., Falcón-Martel, A., Hernández-Tejera, M. y Méndez-Rodríguez, J. (2001). 'Eldi: An agent based museum robot.' En 'Proceedings of the European Workshop on Servive and Humanoid Robots, ServiceRob'2001,' Island of Santorini, Greece.

Domínguez-Brito, A., Hernández-Sosa, D. y Cabrera-Gámez, J. (2002). 'Programming with components in robotics.' En 'Proceedings III Workshop sobre Agentes Físicos,' págs. 85–98. Murcia, España.

Domínguez-Brito, A. C., Andersson, M. y Christensen, H. I. (2000). 'A software architecture for programming robotic systems based on the discrete event system paradigm.' Informe Técnico KTH/NA/P–00/13–SE, Centre for Autonomous Systems, KTH (Royal Institute of Technology).

Firby, R. J. (1989). *Adaptive Execution in Complex Dynamic Worlds.* Tesis Doctoral, Yale University.

Firby, R. J., Kahn, R. E., Prokopwicz, P. y Swain, M. J. (1995*a*). 'An architecture for vision and action.' En 'Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95),' págs. 72–81. Montreal, Canada.

Firby, R. J., Prokopwicz, P. y Swain, M. J. (1995*b*). 'Plan representations for picking up trash.' Informe técnico, Department of Computer Science, University of Chicago.

Fleury, S., Herrb, M. y Chatila, R. (1997). 'GenoM: A tool for the specification and the implementation of operating modules in a distributed robot architecture.' En 'Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS),' págs. 842–848. Grenoble, Francia.

Franklin, S. (1995). *Artificial Minds.* MA: MIT Press.

Garvey, A. J. y Lesser, V. (1993). 'Design-to-time real-time scheduling.' *IEEE Transactions on Systems, Man and Cybernetics*, **23**(6), págs. 1491–1502.

Gat, E. (1991). 'ALFA: A language for programming reactive robotic control systems.' En 'Proceedings of the IEEE Conference on Robotics and Automation,' págs. 1116–1121. Sacramento, CA, USA.

Gat, E. (1992). 'Integrating planning and reacting in a heterogenous asynchronous architecture for controlling real-world mobile robots.' En 'Proceedings of the Tenth National Conference on Artificial Intelligence,' págs. 809–815. San Jose, CA, USA.

Gat, E. (1997*a*). 'ESL: A language for supporting robust plan execution in embedded autonomous agents.' En 'Proceedings of the IEEE Aerospace Conference,' págs. 319–324. Aspen, CO, USA.

Gat, E. (1997*b*). 'On three-layer architectures.' *Artificial Intelligence and Mobile Robots. MIT/AAAI*, págs. 195–210.

Georgeff, M. P. y Ingrand, F. F. (1989). 'Monitoring and control of spacecraft systems using procedural reasoning.' En 'Proceedings of the Eleventh Space Operations-Automation and Robotics Workshop,' Houston, TX, USA.

González, O., Shrikumar, H., Stankovic, J. y Ramamritham, K. (1997). 'Adaptive fault-tolerance and graceful degradation under dynamic hard real-time scheduling.' En 'Proceedings of the Eighteenth IEEE Real-Time Systems Symposium,' págs. 79–89. San Francisco, CA, USA.

Guerra-Artal, C. (2002). *Contribuciones al Seguimiento Visual Precategórico.* Tesis Doctoral, Universidad de Las Palmas de Gran Canaria.

Guzzoni, D., Cheyer, A., Julia, L. y Konolige, K. (1997). 'Many robots make short work.' *AI Magazine*, **18**(1), págs. 55–64.

Hayes-Roth, B. (1988). 'Making intelligent systems adaptive.' Informe Técnico STAN-CS-88-1226, Stanford University.

Hayes-Roth, B. (1993). 'Opportunistic control of action in intelligent agents.' *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(6), págs. 1575–1587.

Hayes-Roth, B. (1995). 'An architecture for adaptive intelligent systems.' *Artificial Intelligence*, **72**(1-2), págs. 329–365.

Hayes-Roth, B., Washington, R., Ash, D., Collinot, A., Vina, A. y Seiver, A. (1992). 'Guardian: A prototype intensive-care monitoring agent.' *Artificial Intelligence in Medicine*, **4**, págs. 165–185.

Hernández-Sosa, D. y Cabrera-Gámez, J. (1996). 'Ejecución distribuida de aplicaciones de procesamiento de imágenes en un entorno computacional heterogéneo. métodos de distribución y experimentos.' En 'Actas de las VII Jornadas de Paralelismo, vol. 3,' págs. 19–40. Santiago de Compostela, España.

Hernández-Sosa, D. y Cabrera-Gámez, J. (1997*a*). 'Distribution of image processing application on a heterogeneous workstation network. modeling, load-balancing and experimental results.' En 'SPIE-97 Parallel on Distributed Methods for Image Processing, vol. 3166,' págs. 170–179. San Diego, CA, USA.

Hernández-Sosa, D. y Cabrera-Gámez, J. (1997*b*). 'Parallelization and computation distribution in a kbvs. experimental results.' En 'Preprints of the VII National Symposium on Pattern Recognition and Image Analysis, vol. 1,' págs. 323–328. Barcelona, España.

Hernández-Sosa, D., Cabrera-Gámez, J. y amd M. Hernández-Tejera, A. F.-M. (1995). 'SVEX: A knowledge-based tool for image segmentation.' En 'Proceedings of International Conference on Acoustics, Speech and Signal Processing, vol. 4,' págs. 2555–2558. Detroit, MI, USA.

Hernández-Sosa, D., Lorenzo-Navarro, J., Cabrera-Gámez, J., Hernández-Tejera, M., Méndez-Rodríguez, J. y Falcón-Martel, A. (1999). 'A generic model for perception-action systems. analysis of a knowledge based prototype.' *Lecture Notes in Computer Science*, **1542**, págs. 293–311.

Hernández-Tejera, M., Cabrera-Gámez, J., Castrillón-Santana, M., Dominguez-Brito, A., Guerra-Artal, C., Hernández-Sosa, D. y Isern-González, J. (1999). 'DESEO: An active vision system for detection, tracking and recognition.' *Lecture Notes in Computer Science*, **1542**, págs. 379–391.

Hexmoor, H., Lammens, J., Caicedo, G. y Shapiro, S. (1993). 'Behavior based AI, cognitive processes, and emergent behaviors in autonomous agents.' Informe Técnico 93-15, Dept. of Computer Science, SUNY at Buffalo.

Hexmoor, H., Lammens, J. y Shapiro, S. (1992). 'An autonomous agent architecture for integrating perception and acting with grounded embodied symbolic reasoning.' Informe Técnico 92-21, Dept. of Computer Science, SUNY at Buffalo.

Horvitz, E. J., Cooper, G. F. y Heckerman, D. E. (1989). 'Reflection and action under scarce resources: Theoretical principles and empirical study.' En 'Proceedings of the Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89),' págs. 1121–1127. Detroit, MI, USA.

Iagnemma, K., Shibly, H., Rzepniewski, A. y Dubowsky, S. (2001). 'Planning and control algorithms to enhance rover rough terrain mobility.' En 'Proceedings of I-SAIRAS: 6th International Syposium on Artificial Intelligence, Robotics, and Automation in Space,' Quebec, Canada.

Ingrand, F. F., Chatila, R., Alami, R. y Robert, F. (1995). 'Embedded control of autonomous robots using procedural reasoning.' En 'ICAR-95,' San Feliu de Guixols, España.

Ingrand, F. F. y Coutance, V. (1993). 'Real-time reasoning using procedural reasoning.' Informe Técnico 93104, LAAS.

Ingrand, F. F., Georgeff, M. P. y Rao, A. S. (1992). 'An architecture for real-time reasoning and system control.' *IEEE Expert Magazine*, **7**(6), págs. 33–44.

Jones, S. D. (1997). *Robust Task Achievement.* Tesis Doctoral, Institut National Polytechnique de Grenoble.

Kato, K. y Hirose, S. (2001). 'Development of the quadruped walking robot, TITAN-IX -mechanical design oncept and application for the humanitarian demining robot.' *Advanced Robotics*, **15**(2), págs. 191–204.

Khatib, O. (1986). 'Real time obstacles avoidance for manipulators and mobile robots.' *International Journal of Robotics Research*, **5**(1), págs. 90–98.

Kolodner, J. (1993). *Case-Based Reasoning.* Morgan Kaufmann Publishers.

Konolige, K. (1997). 'COLBERT: A language for reactive control in sapphira.' En 'KI-97: Advances in Artificial Intelligence, 21st Annual German Conference on Artificial Intelligence,' págs. 31–52. Freiburg, Alemania.

Konolige, K., Myers, K. L., Ruspini, E. H. y Saffiotti, A. (1997). 'The Saphira architecture: A design for autonomy.' *Journal of experimental & theoretical artificial intelligence: JETAI*, **9**(1), págs. 215–235.

Kortenkamp, D. (2001). 'The roles of machine learning in robust autonomous systems.' En 'Proceedings of the AAAI Symposium on Robust Autonomy,' págs. 111–113. Stanford, CA, USA.

Kortenkamp, D., Bonasso, R. P. y Murphy, R. (1998). *Artificial Intelligence and Mobile Robots: Case Studies of Sucessful Robot Systems.* The MIT Press.

Kortenkamp, D., Milam, T., Simmons, R. y Fernandez, J. L. (2002). 'A suite of tools for debugging distributed autonomous systems.' En 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA-2002),' Washington D.C., USA.

Kortenkamp, D., Milam, T., Simmons, R. y Lopez-Fernandez, J. (2001). 'Collecting and analyzing data from distributed control programs.' *Electronic Notes in Theoretical Computer Science*, **55**(2).

Kosecka, J. y Bajcsy, R. (1993). 'Cooperation of visually guided behaviors.' En 'Proceedings of the Fourth International Conference on Computer Vision (ICCV'93),' págs. 502–506. Berlin, Alemania.

Kosecka, J., Christensen, H. y Bajcsy, R. (1995). 'Discrete event modeling of navigation and gaze control.' *International Journal of Computer Vision (Special Issue on Qualitative Vision)*, **14**(2), págs. 179–191.

Langer, D., Rosenblatt, J. K. y Herbert, M. (1994). 'A behavior-based system for off-road navigation.' *IEEE Journal of Robotics and Automation*, **10**(6), págs. 776–782.

Lee, J. B., Likhachev, M. y Arkin, R. C. (2002). 'Selection of behavioral parameters: Integration of discontinuous switching via case-based reasoning with continuous adaptation via learning momentum.' En 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA),' tomo 2, págs. 1275–1281. Washington D.C., USA.

Likhachev, M. y Arkin, R. C. (2001). 'Spatio-temporal case-based reasoning for behavioral selection.' En 'Proceedings of the IEEE International Conference on Robotics and Automation (ICRA),' tomo 2, págs. 1627–1634. Seul, Corea.

Liu, J., Lin, K., Bettati, R., Hull, D. y Yu, A. (1994). *Use of Imprecise Computation to Enhance Dependability of Real-Time Systems*, págs. 157–182. Kluwer Academic Publishers.

Lyons, D. (1990). 'A process-based approach to task-plan representation.' En 'IEEE Int. Conf. on Robotics and Automation,' págs. 2142–2147. Cincinatti, OH, USA.

Maes, P. (1995). 'Artificial life meets entertainment: Life like autonomous agents.' *Communications of the ACM*, **38**(11), págs. 108–114.

McDermott, D. (1991). 'A reactive plan language.' Informe Técnico CSD-RR-864, Computer Science Dept, Yale University.

Miller, D. P., Desai, R., Gat, E., Ivlev, R. y Loch, J. (1992). 'Reactive navigation through rough terrain: Experimental results.' En 'Proceedings of 10th National Conference on Artificial Intelligence (AAAI),' págs. 823–828. San Jose, CA, USA.

Moravec, H. (1990). 'The Stanford Cart and the CMU rover.' En I. J. Cox y G. T. Wilfong, editores, 'Autonomous Robot Vehicles,' págs. 407–441. Springer-Verlag.

Musliner, D. J. (2001). 'Imposing real-time constraints on self-adaptive controller synthesis.' *Lecture Notes in Computer Science*, **1936**, págs. 143–160.

Musliner, D. J., Durfee, E. H. y Shin, K. G. (1993). 'CIRCA: A cooperative intelligent real time control architecture.' *IEEE Transactions on Systems, Man, and Cybernetics*, **23**(6), págs. 1561–1574.

Musliner, D. J., Durfee, E. H. y Shin, K. G. (1995). 'A framework for analyzing resource/quality tradeoffs in real-time AI.'

Musliner, D. J., Goldman, R. P., Pelican, M. J. y Krebsbach, K. D. (1999). 'Self adaptive software for hard real-time environments.' *IEEE Intelligent Systems*, **14**(4), págs. 23–29.

Nilsson, N. J. (1980). *Principles of Artificial Intelligence.* Tioga Press.

Paulus, D., Drexler, C., Reinhold, M., Zobel, M. y Denzler, J. (2000). 'Active Computer Vision System.' En V. Cantoni y C. Guerra, editores, 'Computer Architectures for Machine Perception,' págs. 18–27. IEEE Computer Society, Los Alamitos, CA, USA.

Pirjanian, P. (1998). *Multiple Objective Action Selection and Behavior Fusion Using Voting.* Tesis Doctoral, Dpt. of Medical Informatics and Image Analysis, Aalborg University.

Prassler, E., Ritter, A., Schaeffer, C. y Fiorini, P. (2000). 'A short history of cleaning robots.' *Autonomous Robots*, **9**(3), págs. 211–226.

Ram, A., Arkin, R. C., Boone, G. y Pearce, M. (1994). 'Using genetic algorithms to learn reactive control parameters for autonomous robotic navigation.' *Journal of Adaptive Behavior*, **2**(3), págs. 277–305.

Ram, A., Arkin, R. C., Clark, R. J. y Moorman, K. (1992). 'Case-based reactive navigation: A case-based method for on-line selection and adaptation of reactive control parameters in autonomous robotic systems.' Informe técnico, Georgia Tech.

Ramamritham, K. y Stankovic, J. (1991). 'Scheduling strategies adopted in Spring: A overview.' En A. van Tilborg y G. Koob, editores, 'Foundations of Real-Time Computing: Scheduling and Resource Allocation,' págs. 277–307. Kluwer Academic Publishers.

Rosenblatt, J. K. (1995). 'DAMN: A distributed architecture for mobile navigation.' En 'Proceedings of the AAAI Spring Symp. on Lessons Learned from Implememted Software Architectures for Physical Agents,' Stanford, CA, USA.

Russell, S. J. y Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Saffiotti, A., Ruspini, E. H. y Konolige, K. (1997). 'Using fuzzy logic for mobile robot control.' En H. P. D. Dubois y H. J. Zimmermann, editores, 'International Handbook of Fuzzy Sets and Possibility Theory,' tomo 5. Kluwer Academic Publishers Group, Norwell, MA, USA, and Dordrecht, The Netherlands.

Schoner, G., Dose, M. y Engels, C. (1996). 'Dynamics of behavior: Theory and applications for autonomous robot architectures.' *Robotics and Autonomous Systems*, **16**(4), págs. 213–246.

Seto, D., Lehoczky, J. P., Sha, L. y Shin, K. G. (1996). 'On task schedulability in real-time control system.' En 'Proceedings of the IEEE Real-Time Systems Symposium,' págs. 13–21.

Simmons, R. (1992). 'Concurrent planning and execution for autonomous robots.' *IEEE Control Systems*, **12**(1), págs. 46–50.

Simmons, R. y Apfelbaum, D. (1998). 'A task description language for robot control.' En 'Proceedings of Conference on Intelligent Robotics and Systems,' Vancouver, Canada.

Simon, D., Espiau, B., Kapellos, K. y Pissard-Gibollet, R. (1997). 'ORCCAD: software engineering for real-time robotics. a technical insight.' *Robotica*, **15**(1), págs. 111–115.

Simoncelli, M., Zunino, G., Christensen, H. I. y Lange, K. (2000). 'Autonomous pool cleaning: Self localization and autonomous navigation for cleaning.' *Autonomous Robots*, **9**(3), págs. 261–270.

Stewart, D. B. (1994). *Real-Time Software Design and Analysis of Reconfigurable Multi-Sensor Based Systems.* Tesis Doctoral, ECE Dept., Carnegie Mellon University.

Stewart, D. B. y Khosla, P. K. (1997). 'Mechanisms for detecting and handling timing errors.' *Communications of the ACM*, **40**(1), págs. 87–93.

Thrun, S., Bennewitz, M., Burgard, W., Cremers, A. B., Dellaert, F., Fox, D., Hahnel, D., Rosenberg, C. R., Roy, N., Schulte, J. y Schulz, D. (1999). 'MINERVA: A tour-guide robot that learns.' En 'KI - Kunstliche Intelligenz,' págs. 14–26.

Tsotsos, J. K. (1995). 'Behaviorist intelligence and the scaling problem.' *Artificial Intelligence*, **75**(4), págs. 135–160.

Tsotsos, J. K. (1997). 'Intelligent control for perceptually attentive agents: The S* proposal.' *Robotics and Autonomous Systems*, **21**(1), págs. 5–21.

Wagner, T., Garvey, A. y Lesser, V. (1998). 'Criteria directed task scheduling.' *Journal for Approximate Reasoning–Special Scheduling Issue*, **19**(1-2), págs. 91–118.

Zilberstein, S. (1996). 'Using anytime algorithms in intelligent systems.' *AI Magazine*, **17**(3), págs. 73–83.

Zilberstein, S., Charpillet, F. y Chassaing, P. (1999). 'Real-time problem-solving with contract algorithms.' En 'Proceedings of the 16th International Joint Conference on Artificial Intelligence,' págs. 1008–1015. Stockholm, Sweden.

Zilberstein, S. y Russell, S. J. (1996). 'Optimal composition of real-time systems.' *Artificial Intelligence*, **82**(1-2), págs. 181–213.