UNIVERSIDAD CARLOS III DE MADRID

DEPARTAMENTO DE INGENIERÍA TELEMÁTICA

TESIS DOCTORAL

TÉCNICAS DE SOPORTE A LA FLEXIBILIDAD FUNCIONAL EN SISTEMAS EMBARCADOS DISTRIBUIDOS DE TIEMPO REAL

Autora: **Iria Manuela Estévez Ayres**
Ingeniera de Telecomunicación

Directores: **María de la Soledad García Valls**
Doctora Ingeniera de Informática

Luís Miguel Pinho de Almeida
Doutor em Engenharia Electrotécnica

Leganés, Junio de 2007

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día __ de _____ de _____.

Presidente

Vocal

Vocal

Vocal

Secretario

Realizado el acto de defensa y lectura de la Tesis el día __ de _____ de _____ en _____.

Calificación:

EL PRESIDENTE EL SECRETARIO LOS VOCALES

A mis padres
y a mi hermano.

Nada se construye sobre piedra; todo se construye sobre arena, pero debemos construir como si la arena fuese piedra.

Jorge Luis Borges

Agradecimientos

Con la presentación de esta tesis se cierra un capítulo, una etapa en el camino. Como en todas las buenas lecturas, hubo párrafos arduos, donde seguir se hacía cada vez más difícil y una se pregunta por qué ha escogido precisamente este libro, por qué no devolverlo directamente a la biblioteca, a la librería, exigiendo, además, una compensación moral por toda la desazón sentida. Hubo diálogos imposibles, monólogos delirantes y escenas solitarias, que requerían silencio y recogimiento para poder ser completadas. Pero también hubo pasajes que te gustaría releer una y otra vez: escenas de compañerismo, solidaridad, generosidad, de crecimiento personal y de enriquecimiento mutuo. Ahora, a punto de pasar la página, llegando a la posada desde la que se retomará el camino, se hace parada y balance, llegando a la conclusión de que todo valió la pena gracias a las personas valiosísimas que me acompañaron y que encontré durante este peregrinaje.

Gracias, en primer lugar, a Marisol y a Luís, mis directores de tesis. No sólo por ser, a nivel profesional y técnico, los mejores mentores que podría haber deseado, sino, sobre todo, por su talla humana. Sin su apoyo, sin sus ánimos en todos los momentos en los que estuve a punto de tirar la toalla, de ir por la vía fácil y coger el bus escoba, no estaría ahora escribiendo estos agradecimientos.

A todos los miembros del Departamento de Ingeniería Telemática. El nivel de implicación personal, la profesionalidad, el compañerismo, la solidaridad presentes a todos los niveles hacen que sienta que Fortuna me sonrió al entrar a trabajar aquí. A mis compañeros de docencia y de investigación durante estos años. A Simon, mi compañero de despacho, por los tés y las conversaciones alternativas. También a los que ya no trabajan con nosotros: Rosa, Guille, Luis, Jesús Villamor... MariCarmen: gracias por ser tú.

A todos los compañeros del Laboratório de Sistemas Electrónicos da Universidade de Aveiro y a sus familias, que me acogieron e hicieron que mi estancia en Aveiro sea uno de los recuerdos más felices y que guardo con más cariño del desarrollo de esta tesis. Cada vez que vuelvo a Aveiro, nada más llegar, me siento en casa.

Gracias, en particular, a Paulo Pedreiras y a Ricardo Marau, que, además de todas las discusiones técnicas que enriquecieron este trabajo, me ofrecieron su amistad y compañerismo. También a Javi, su mujer y sus niños, que hicieron, con su amistad, más agradable, si cabe, mi estancia.

A todos los amigos que hice durante esta etapa, y a mis compañeros en la larga y oscura senda del doctorado: a Raquel, JJ, Antonio y Julio, por las largas conversaciones sobre la vida; a Sergut, Pablo Serrano, Rosa, Guille y Luis; a Baricco por Javi y a Javi por Baricco; a Caro por toda esa comida mexicana que me acompañó durante los primeros años, por las largas sobremesas y noches hablando, por tu dulzura, por ser mi amiga en la distancia y pese a ella.

Al resto del sector gallego: Norbi y Pablo, por el apoyo recibido en los malos momentos, por soportar el pequeño caos que soy, por las conversaciones técnicas y no técnicas, por, siempre, estar dispuestos a arreglar el mundo.

A mis amigos de siempre: David, Aldara y Dani. Por miles de razones que no cabrían aquí ni aunque tuviera todo el libro para explicarlas. La principal de ellas es que construisteis vuestras casas en mi corazón y que no os vais ni con agua caliente. Gracias por ser mis amigos, pese a la distancia, en los buenos y, sobre todo, en los malos momentos. Dentro de muchos años espero poder explicaros el resto.

A Raquel, por ser mi amiga, mi hermana, un trocito de mí con piernas, por el cobalto y las rosas amarillas, por arrancar las tachuelas, por, siempre, batir palmas por mí.

Gracias ó *meu* reiki positivo, por poner la nota color esperanza. Por todos los ánimos, por las largas conversaciones, y por, pese a todas las horas y días que te escatimé debido a esta tesis roba tiempo, musas y pensamientos, ser capaz de inventar sólo para nosotros el *Tempo*, uniendo islas en este mar.

Finalmente, gracias a las personas más importantes de mi vida: mi familia.

Gracias David, por todas esas cosas inefables que suponen que ser tu hermana sea el mayor de los orgullos, el mayor de los regalos. Gracias por la luz que sigue brillando.

Gracias a mis padres. Todo lo bueno que hay en mí me lo enseñasteis vosotros. Me enseñasteis la valentía, la generosidad, el valor del sacrificio, cómo nunca rendirse ante las adversidades. Siempre me instasteis y me animasteis a volar sola, a encontrar mi propio camino. Aún así, el sacrificio más duro realizado durante esta etapa fue estar lejos de vosotros: os echo de menos todos y cada uno de los días. Gracias por amarme tanto.

Por todo y a todos, muchísimas gracias,

Iria

Resumen

Durante la última década, gracias a los avances en diversos campos tecnológicos, se han diversificado el conjunto de entornos en los que es necesario desarrollar sistemas que ofrezcan garantías temporales. Muchos de estos son entornos dinámicos, donde las características de la carga computacional no siempre pueden ser predecibles, y donde ya no es aplicable la aproximación clásica de diseño, habitualmente pesimista, que asegura garantías temporales pero que puede implicar en entornos dinámicos un gasto de recursos prohibitivo.

Así, se plantea el reto de adaptar las metodologías empleadas en diferentes niveles arquitecturales a estos nuevos entornos, y explorar nuevas vías y paradigmas que permitan conjugar flexibilidad funcional y dinamismo con predictibilidad temporal.

Esta tesis aborda este reto mediante la exploración de la aplicabilidad a sistemas de tiempo real de conceptos propios del paradigma de orientación a servicios, con el fin de ofrecer flexibilidad, y, al mismo tiempo, beneficiarse de algunas de las ventajas que éste ofrece. Para conseguir tal fin, se propone un modelo concreto de sistema basado en una aproximación holística al diseño y configuración, donde las aplicaciones están gobernadas por tiempo. Fijado el modelo de sistema, se propone un modelo de aplicación basada en servicios y se analizan desde diferentes perspectivas las entidades y procesos que estarán presentes en una arquitectura que le dé soporte, diferenciando dos posibles aproximaciones a la composición que influirán en el diseño de dicha arquitectura: estática, que una vez realizada no admite reconfiguraciones, o dinámica, en la cual una aplicación puede reconfigurarse en tiempo de ejecución. Se proponen, además, algoritmos para la composición de aplicaciones, tanto exhaustivos, aplicables en la aproximación estática, como mejorados, con un tiempo de ejecución acotado, apropiados para su empleo en tiempo de ejecución. Finalmente, se realiza la validación del modelo y de las ideas propuestas mediante la implementación de un prototipo sobre un protocolo concreto de comunicaciones de tiempo real, al que se le realizaron pequeñas adaptaciones y sobre el cual se definió una arquitectura adecuada.

Abstract

During the last decade, due to the advances in several technology fields, the application domains where the development of systems with temporal guarantees is needed has increased. The majority of such domains are dynamic; the characteristics of their computational load cannot always be predicted in advance. Although the classical design approach provides temporal guarantees, it is no longer applicable since it is too pessimistic, and it implies prohibitive resource consumption.

Thus, new challenges raise. On one side, it is required to adapt the current methodologies used in different architectural levels to these new environments. Also, it is needed to explore new directions and paradigms that allow combining functional flexibility and dynamism with temporal predictability.

The current work addresses these new challenges through the exploration of the applicability of concepts from the service oriented paradigm to distributed real-time systems. The introduction of some of the characteristics of the service oriented paradigm will allow to provide support for dynamic flexibility. Therefore, the current work proposes a concrete system model based on a holistic time-triggered-based approach for design and configuration. Based on this system model, a service-based application model is proposed. Also, it analyses the architectural entities and processes from different points of view, distinguishing between two different architectural design approaches: static and dynamic. The former applies when no reconfigurations at run-time are admitted, and the latter where these reconfigurations are possible. Several application composition algorithms have been proposed: (1) an exhaustive algorithm, applicable to the static approach and (2) an improved algorithm, with bounded execution times, suitable for its usage at run-time. Eventually, to validate the feasibility of the model and the proposed ideas, an architecture has been defined and a prototype of it has been implemented on top of a concrete real-time communications protocol.

Contenido

Lista de acrónimos XIX

1. Planteamiento y Objetivos **1**
1.1. Introducción 1
1.2. Motivación de la Tesis 4
1.3. Objetivos y Retos 6
1.4. Visión general de las contribuciones propuestas 8
1.5. Organización de la memoria 10

2. Estado del Arte **13**
2.1. Sistemas de tiempo real 13
2.1.1. Tareas de tiempo real 14
2.1.2. Planificación de sistemas de tiempo real 15
2.1.3. Planificación centralizada 17
2.1.3.1. Planificación estática: ejecutivos cíclicos 17
2.1.3.2. Planificación dinámica 18
2.1.3.3. Comunicación entre tareas 24
2.1.3.4. Plazos superiores a los períodos 28
2.1.3.5. Tratamiento de eventos aperiódicos 29
2.1.4. Cambios de modo en sistemas de tiempo real 30
2.2. Sistemas de tiempo real distribuidos 30
2.2.1. Planificación distribuida 32
2.2.1.1. Modelos y políticas de planificación para el análisis de sistemas distribuidos 33
2.2.1.2. Análisis de planificabilidad 37
2.2.2. Comunicación de tiempo real 40
2.2.2.1. Propiedades de composición, escalabilidad y tolerancia a fallos 40
2.2.2.2. Arquitecturas gobernadas por eventos vs. gobernadas por tiempo 42
2.2.2.3. Modelos de cooperación 44
2.3. Redes de comunicaciones de tiempo real 45

2.3.1. CAN . . . 46
2.3.2. Protocolo TTP/C . . . 46
2.3.3. FlexRay . . . 47
2.3.4. DeviceNet . . . 48
2.3.5. Profibus . . . 48
2.3.6. Ethernet . . . 49
2.3.7. Ethernet PowerLink . . . 50
2.3.8. Protocolos FTT . . . 51
2.3.9. ARINC 629 . . . 52
2.3.10. AFDX / ARINC 664 . . . 52
2.4. Arquitecturas de Gestores de Calidad de Servicio . . . 54
2.5. Sistemas basados en componentes . . . 55
2.6. Paradigma de orientación a servicios . . . 57
2.6.1. Composición de servicios para la construcción de aplicaciones genéricas . . . 60
2.7. Trabajos relacionados . . . 62
2.7.1. RTSOA (Arizona State University) . . . 63
2.7.1.1. Análisis . . . 64
2.7.2. Tesis doctoral de Xiaohui Gu (University of Illinois) . . . 64
2.7.2.1. Análisis . . . 67
2.7.3. HOLA-QoS (Universidad Politécnica de Madrid y Universidad Carlos III de Madrid) . . . 68
2.7.3.1. Análisis . . . 70
2.7.4. Tesis de Tešanović (Linköping Universitet) . . . 70
2.7.4.1. Análisis . . . 72
2.7.5. Tecnología de componentes para RTRMI (Texas A & M University) . . . 73
2.7.5.1. Análisis . . . 74
2.7.6. Tecnología de componentes de tiempo real (Texas A & M University) . . . 74
2.7.6.1. Análisis . . . 76
2.7.7. QuO (BBN Technology) y CIAO (University of Washington) 77
2.7.7.1. Análisis . . . 79
2.7.8. Otros trabajos relacionados . . . 80
2.7.9. Conclusiones . . . 81

3. Modelo de Aplicaciones Basadas en Servicios 83
3.1. Introducción . . . 83
3.2. Discusión sobre el modelo del sistema . . . 87
3.3. Modelo de servicio . . . 90
3.3.1. Perfiles de servicio . . . 91
3.3.2. Elección de perfiles . . . 93

3.3.2.1. Ejemplo de estudio de compatibilidad de perfil . . 96
3.3.3. Caracterización temporal de un perfil de servicio 101
3.4. Aplicaciones basadas en servicios 102
3.4.1. Clasificación de las aplicaciones 103
3.4.2. Modelo funcional 104
3.4.3. Modelo temporal 107
3.4.3.1. Planificación holística usando flujos de datos . . . 108
3.4.3.2. Planificación holística superpuesta de mensajes y tareas 109
3.4.4. Extensión a la planificación holística 111
3.5. Conclusiones 112

4. Soporte arquitectónico para la composición 115
4.1. Introducción 115
4.2. Descripción funcional de la arquitectura 116
4.2.1. Fases en la composición de servicios 118
4.3. Requisitos arquitectónicos para la composición dinámica 120
4.3.1. Arquitectura propuesta 121
4.4. Requisitos arquitectónicos para la composición estática 124
4.4.1. Arquitectura propuesta 126
4.5. Conclusiones 128

5. Algoritmos para la Composición de Aplicaciones 131
5.1. Introducción 131
5.2. Planteamiento del problema 133
5.3. Algoritmo de Composición 134
5.4. Algoritmo de Composición Mejorado 141
5.4.1. Heurísticos desarrollados 145
5.4.1.1. Primera combinación válida 146
5.4.1.2. Tamaño de bloque fijo 146
5.4.1.3. Uso de la dispersión como medida para realizar o no poda 147
5.4.1.4. Tamaño de bloque variable entre niveles 147
5.5. Figuras de Mérito 147
5.5.1. Figuras de Mérito desarrolladas para el Algoritmo de Composición 148
5.5.1.1. Minimización del tiempo de respuesta extremo a extremo 149
5.5.1.2. Maximización del tiempo de respuesta extremo a extremo 149
5.5.1.3. Minimización/maximización del número de nodos físicos involucrados 150
5.5.1.4. Minimización de la utilización media 150

5.5.1.5. Equilibrado de carga entre los nodos involucrados 153
5.6. Validación experimental 154
5.6.1. Descripción del prototipo 155
5.6.2. Metodología de simulación 158
5.6.3. Análisis y presentación de resultados 161
5.7. Conclusiones 176

6. Composición Dinámica de Servicios 177
6.1. Introducción 177
6.2. Elección del protocolo de comunicaciones 178
6.3. Funcionamiento de FTT-SE 178
6.3.1. Tranmisión de mensajes síncronos 179
6.3.1.1. Construcción de la planificación para cada ciclo elemental 181
6.3.2. Transmisión de mensajes asíncronos 183
6.4. Extensiones propuestas al paradigma FTT 186
6.5. Prototipo del Sistema 189
6.5.1. Cambios introducidos en los nodos esclavos 189
6.5.2. Extensiones realizadas en el maestro 191
6.5.3. Entidades implementadas en el maestro 194
6.5.4. Herramientas de medida implementadas 194
6.5.5. Consideraciones prácticas 195
6.5.5.1. Direccionamiento 195
6.5.5.2. Gasto de recursos por parte de perfiles instanciados pero que no forman parte de una aplicación compuesta 196
6.6. Validación experimental 197
6.6.1. Interferencia introducida por las herramientas de medida . . 197
6.6.2. Coste cambio de la configuración de las aplicaciones 200
6.6.3. Dos implementaciones similares de una tarea productora-consumidora 201
6.6.4. Dos implementaciones de una tarea productora-consumidora que ofrecen diferente calidad de servicio 203
6.7. Conclusiones 208

7. Conclusiones y Líneas Futuras 209
7.1. Conclusiones generales 209
7.2. Resumen de la tesis y resultados destacables 210
7.3. Futuras líneas de trabajo 215
7.3.1. Mejoras al modelo de sistema y a los algoritmos de composición 215
7.3.1.1. Desarrollo y evaluación de nuevas figuras de mérito para la composición 215

7.3.1.2. Evaluación exhaustiva de los algoritmos de composición 216
7.3.1.3. Extensión de los algoritmos de composición para la inclusión de algoritmos heurísticos de asignación de prioridades 216
7.3.1.4. Extensión de los algoritmos de composición para el tratamiento de las características propias de los mensajes 216
7.3.1.5. Extensión del modelo de sistema a sistemas orientados a eventos 216
7.3.1.6. Aplicación de técnicas de gestión de calidad de servicio 217
7.3.2. Mejoras a la arquitectura propuesta sobre FTT-SE 217
7.3.2.1. Implementación completa de la arquitectura propuesta 218
7.3.2.2. Gestión de la tolerancia a fallos a nivel de aplicación en la arquitectura 218
7.3.2.3. Composición inter-dominio en FTT-SE 218
7.3.3. Implementación de la arquitectura sobre tecnologías Java . . 219
7.3.3.1. Implementación de una arquitectura de composición estática sobre RT-RMI 219
7.3.3.2. Implementación de una arquitectura de composición dinámica sobre RT-RMI basada en el paradigma FTT 220
7.3.4. Implementación de herramientas de ayuda al diseñador . . . 220

Referencias **221**

Índice de figuras

1.1. Propuesta de arquitectura 9

2.1. Ejemplo de transacción de tiempo real 34
2.2. Modelo lineal gobernado por eventos 36
2.3. Ciclo elemental en FTT-Ethernet 51
2.4. Estructura RT–UML 57
2.5. Modelo conceptual de una arquitectura orientada a servicios 58
2.6. Protocolo de composición de servicios de *SpiderNet* 65
2.7. Estructura en subsistemas de *SpiderNet* y gestión de la monitorización en *Utility SpiderNet* 66
2.8. Vista general de la arquitectura HOLA-QoS 69
2.9. Modelo RTCOM 70
2.10. Desarrollo de un sistema de tiempo real atendiendo a la metodología ACCORD 71
2.11. Protocolo para la migración de un componente a otra ubicación (figura tomada de [191]) 74
2.12. Aislamiento de los recursos mediante el uso de componentes (figura tomada de [233]) 75
2.13. Arquitectura del sistema (figura tomada de [233]) 76
2.14. Provisión dinámica de QoS en QuO (figura tomada de [232]) 77
2.15. Qosket (figura tomada de [232]) 78
2.16. Elementos de CIAO (figura tomada de [232]) 79
2.17. Componente Qosket usando CIAO (figura tomada de [232]) 79

3.1. Abstracción de una aplicación 84
3.2. Ejemplo de aplicación distribuida 86
3.3. Características del perfil del ejemplo del apartado 3.3.2.1 97
3.4. Conjunto resultante $\vec{\Delta}_{out}^{pet}$ para el ejemplo del apartado 3.3.2.1 98
3.5. Cálculo de $\vec{\Delta}_{out}^{perfil}$ y $\vec{\Delta}_{out}^{pet} \cap \vec{\Delta}_{out}^{perfil}$ del ejemplo del apartado 3.3.2.1 99
3.6. Cálculo de $\vec{\Delta}_{out}^{perfil}$ y $\vec{\Delta}_{out}^{pet} \cap \vec{\Delta}_{out}^{perfil}$ una vez aplicada ζ_{out}^{2} del ejemplo del apartado 3.3.2.1 100
3.7. Caso general invocación 102
3.8. Clasificación aplicaciones 103

3.9. Especificación de una aplicación deseada 105
3.10. Estructura del grafo de una aplicación en ejecución 107
3.11. Tarea productora 110
3.12. Tarea consumidora 110
3.13. Tarea productora-consumidora 111

4.1. Funcionalidad básica ofrecida por la arquitectura 116
4.2. Entidades presentes en la arquitectura 117
4.3. Pasos en la fase de publicación 118
4.4. Pasos en la fases de Especificación, Descubrimiento y Composición 119
4.5. Arquitectura propuesta para la composición dinámica 122
4.6. Arquitectura propuesta para la composición estática 126

5.1. Presentación de la problemática 133
5.2. Aplicación ejemplo y su posible cronograma temporal 135
5.3. Aplicación ejemplo y posibles ubicaciones de perfiles 136
5.4. Módulos implementados en el prototipo 155
5.5. Estructura de la aplicación deseada empleada en los experimentos . 160
5.6. Figura de Mérito Combinada: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo. 165
5.7. Figura de Mérito Combinada: Tiempos de ejecución en función del número de combinaciones 167
5.8. Figura de Mérito R_{e2e}: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo. 169
5.9. Figura de Mérito R_{e2e}: Tiempos de ejecución en función del número de combinaciones 171
5.10. Figura de Mérito Utilización Perfiles: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo. 173
5.11. Figura de Utilización Media Perfiles: Tiempos de ejecución en función del número de combinaciones 175

6.1. Arquitectura interna del maestro en el protocolo FTT-Ethernet original [176] 180
6.2. Causalidad temporal en los enlaces de bajada [150] 182
6.3. Esquema de señalización fuera de banda [150] 183
6.4. Esquema de señalización fuera de banda para FTT-SE [150] 184
6.5. Arquitectura propuesta 187
6.6. Ejemplo de reconfiguración 188
6.7. Cambios realizados en el módulo de recepción y transmisión de los nodos esclavos 190
6.8. Esquema del prototipo implementado en el maestro 193
6.9. Esquema del hilo periódico que realiza el cambio en las aplicaciones compuestas 194

6.10. Interfaces ofrecidas por la herramienta de medida 195
6.11. Configuración de los nodos en los experimentos realizados 197
6.12. Configuración de la aplicación en los experimentos realizados . . . 198
6.13. Interferencia introducida por la herramienta de medida 198
6.14. Interferencia introducida por la herramienta de medida DRQTracer (figura tomada de [25]) 199
6.15. Coste cambio de la configuración de las aplicaciones 200
6.16. Experimento 1: Transmisión de paquetes en la red observada por el programa capturador de tráfico 201
6.17. Experimento 1: Tiempo de respuesta de los nodos productores medido por el programa capturador de tráfico 202
6.18. Experimento 1: Diferencia temporal entre activaciones de la tarea consumidora, medido en el nodo consumidor 202
6.19. Experimento 2: Transmisión de paquetes en la red observada por el programa capturador de tráfico 203
6.20. Experimento 2: Tiempo de respuesta de los nodos productores medido por el programa capturador de tráfico 204
6.21. Experimento 2: Diferencia temporal entre activaciones de la tarea consumidora, medido en el nodo consumidor 204
6.22. Perfiles $2A$ y $2B$ ubicados en el mismo nodo físico ($\phi^m_{2A} = 100\,T_{EC}$, $\phi^m_{2B} = 200\,T_{EC}$) 205
6.23. Perfiles ubicados en nodos físicos distintos ($\phi^m_{2A} = 500\,T_{EC}$, $\phi^m_{2B} = 200\,T_{EC}, T_C = 20$) 206
6.24. Perfiles ubicados en nodos físicos distintos ($\phi^m_{2A} = 300\,T_{EC}$, $\phi^m_{2B} = 80\,T_{EC}, T_C = 20$) 207

Índice de cuadros

2.1. Clasificación algoritmos planificación y ejemplos para planificación centralizada 17
2.2. Estructura original del *Trigger Message* en FTT-Ethernet 51
2.3. Resumen de trabajos relacionados 63

5.1. Ejemplos comparación entre figuras de mérito basadas en utilización 152
5.2. Ejemplos comparación entre figuras de mérito basadas en el uso de la desviación típica entre utilizaciones. 154
5.3. Características hardware de las máquinas donde se realizaron las simulaciones 161
5.4. Figura de Mérito Combinada: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo 164
5.5. Figura de Mérito Combinada: Error medio, máximo y condicionado a error en porcentaje para el mejor caso. 164
5.6. Figura de Mérito Combinada: Error Medio, Máximo y Error Condicionado a Error en Porcentaje con utilización media en el sistema de $\bar{U} = 0{,}2$ 166
5.7. Figura de Mérito Combinada: Comparativa número de combinaciones exploradas para el peor caso 166
5.8. Figura de Mérito R_{e2e}: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo 168
5.9. Figura de Mérito R_{e2e}: Error medio, máximo y condicionado a error en porcentaje para el mejor caso. 168
5.10. Figura de Mérito R_{e2e}: Error medio, máximo y condicionado a error en porcentaje para $\bar{U} = 0{,}20$. 170
5.11. Figura de Mérito R_{e2e}: Comparativa número medio de combinaciones exploradas para el peor caso 170
5.12. Figura de Mérito Utilización Perfiles: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo . . 172
5.13. Figura de Mérito Utilización Perfiles: Error medio, máximo y condicionado a error en porcentaje para el mejor caso. 172
5.14. Figura de Mérito Utilización Perfiles: Error medio, máximo y condicionado a error en porcentaje para $\bar{U} = 0{,}2$. 174

5.15. Figura de Mérito Utilización Perfiles: Comparativa número de combinaciones exploradas para el peor caso 174

6.1. Estructura original del Mensaje de Activación 188

Índice de Algoritmos

2.1. Algoritmo iterativo para la obtención de los tiempos de respuesta de Tindell 39
5.1. Algoritmo de composición exhaustivo 137
5.2. Algoritmo para el cálculo de la figura de mérito y los parámetros de las tareas y mensajes de la aplicación 139
5.3. Fase de preparación previa del algoritmo de composición mejorado . 142
5.4. Algoritmo de composición mejorado 144
5.5. Análisis de planificabilidad de la red implementado [176] 157

Lista de acrónimos

A

$ACCORD$ (Aspectual Component-Based Real-Time System Development) Metodología para el desarrollo de sistemas de tiempo real basados en aspectos y componentes.

$AFDX$ Avionics Full-Duplex Switched Ethernet

B

$B2C$ (Business to Consumer) Aplicaciones empresa-consumidor final

$BDRS$ En el modelo de composición dinámica, Base de Datos de Requisitos del Sistema

$BPEL4WS$ Business Process Execution Language for Web Services

C

CAN (Controller Area Network) Protocolo de red de controlador de área

$CBSD$ (Component Based Software Development) Desarrollo de Software Basado en Componentes

CCM (CORBA Component Model) Modelo de componentes de CORBA

$CIAO$ (Component-Integrated ACE ORB) ORB ACE con soporte para componentes

CNI (Communication Network Interface) En una red TTP/C, interfaz de comunicación de red

$CORBA$ Common Object Request Broker Architecture

$CoSeRT$ (Composition of Service-Based Real-Time Applications) Composición de Aplicaciones de Tiempo Real basadas en Servicios

$CSMA$ (Carrier Sense Multiple Access) Acceso múltiple con detección de portadora

$CSMA-CA$ (Carrier Sense Multiple Access with Collision Avoidance) CSMA con elusión de colisiones

$CSMA-CD$ (Carrier Sense Multiple Access with Collision Detection) CSMA con detección de colisiones

$CSMA-NBA$ (Carrier Sense Multiple Access with Non-destructive Bitwise Arbitration) CSMA con arbitraje de bit no destructivo

D

$DCLC$ Delay-Constrained Least Cost path

DMS (Deadline Monotonic Scheduling) Método de prioridad a la tarea más urgente

$DREQUIEMI$ (Distributed Real-Time System Quality of Service for Embedded Method Invocation)

DSM (Distributed Shared Memory) Memoria compartida distribuida

E

EC (Elementary Cicle) En el paradigma FTT , ciclo elemental

EDF (End Deadline First) Prioridad a la tarea con el plazo más próximo

EJB Enterprise Java Beans

F

FTT (Flexible Time–Triggered paradigm) Paradigma flexible gobernado por tiempo

$FTT-CAN$ (Flexible Time–Triggered paradigm for CAN) Paradigma flexible gobernado por tiempo para redes CAN

$FTT-Ethernet$ (Flexible Time–Triggered paradigm for Ethernet) Paradigma flexible gobernado por tiempo para redes Ethernet

$FTT-SE$ (Flexible Time–Triggered paradigm for Switched Ethernet) Paradigma flexible gobernado por tiempo para redes Switched Ethernet

FTU (Fault Tolerance Unit) En una red TTP/C, unidad tolerante a fallos

H

$HOLA - QoS$ (Homogeneneous and Open Layered Architecture for Quality of Service management) Arquitectura para la gestión de calidad de servicio con niveles

$HOPA$ (Heuristic Optimized Priority Assignment) Algoritmo heurítico optimizado para la asignación de prioridades

I

$IPCP$ (Immediate Priority Ceiling Protocol) Protocolo de Herencia inmediata del techo de Prioridad

L

$LFSO$ (Lock Free Shared Objects)

LLF (Least Laxity First) Método de prioridad a la tarea con la menor holgura

M

MAC (Medium Access Control) Control de acceso al medio

MCP Multiple Constrained Path selection

$MEDL$ (Message Descriptor List) En una red TTP/C, tabla de descriptores de mensajes

P

PCP (Priority Ceiling Protocol) Protocolo de Techo de Prioridad

PIP (Priority Inheritance Protocol) Protocolo de Herencia de Prioridad

Q

QoS (Quality of Service) Calidad de servicio

QuO (Quality Objets) Middleware basado en objetos con calidad de servicio

R

$REALTOR$ (REsource ALlocaTOR) Protocolo de descubrimiento y ubicación de componentes de tiempo real para RMI y RTSJ.

RMI (Remote Method Invocation) En tecnologías Java, mecanismo de invocación remota.

RMS (Rate Monotonic Scheduling) Planificación de Tasa Monótona. Prioridad a la tarea más frecuente.

RPC (Remote Procedure Call) Llamada a procedimiento remoto

$RTCOM$ (Real Time Component Model) Modelo de componentes de tiempo real.

$RT-CORBA$ (Real-Time CORBA) CORBA de tiempo real

$RT-JAVA$ (Real-Time Java) Java de tiempo real

$RT-RMI$ (Real-Time RMI) RMI de tiempo real

$RTSJ$ (Real-Time Specification for Java) Especificación de Java para tiempo real.

$RT-UML$ (Real-Time Unified Modelling Language) UML para tiempo real

S

SLA (Service Level Agreement) Acuerdo a nivel de servicio

SOA (Service Oriented Architectures) Arquitecturas orientadas a servicio

SOC Service Oriented Computing

SRP (Stack Resource Policy)

T

TBS (Total Bandwidth Server) Servidor de ancho de banda total.

$TDMA$ (Time Division Multiple Access) Acceso al medio por división en el tiempo

TM (Trigger Message) En el paradigma FTT, mensaje de disparo

TTA (Time Triggered Architecture) Arquitectura gobernada por tiempo

TTP/C (Time Triggered Protocol for Class C) Protocolo gobernado por tiempo para clase C

U

UML (Unified Modelling Language) Lenguaje de modelado unificado

W

$W3C$ World Wide Web Consortium

$WCDO$ (Worst Case Analysis for Dynamic Offsets) Análisis de Peor Caso para Offsets Dinámicos

$WCDOPS$ (Worst Case Analysis for Dynamic Offsets with Priority Schemes) Análisis de Peor Caso con Offsets Dinámicos y Perfiles de prioridad

$WCET$ (Worst Case Execution Time) Tiempo de ejecución en el peor caso

$WCRT$ (Worst Case Response Time) Tiempo de respuesta en el peor caso.

WFO (Wait Free Objects)

X

$XRTJ$ (eXtensible high-integrity Real–Time Java) Java de Tiempo Real extensible para sistemas críticos

Capítulo 1

Planteamiento y Objetivos

En este capítulo se describe de forma general el contexto de la presente tesis doctoral. En primer lugar, se plantean las motivaciones que han dado lugar a la elaboración de la tesis, presentándose de forma concisa los objetivos de la misma. Posteriormente, se describe el plan de trabajo planteado para su elaboración. Finalmente, se presenta la organización del resto de capítulos de este documento.

1.1. Introducción

Muchos de los sistemas de tiempo real actuales incluyen dentro de sus requisitos la capacidad de trabajar en entornos dinámicos, donde las características de la carga computacional no siempre son predecibles [182]. El diseño de estos sistemas requiere conjugar flexibilidad y dinamismo con predictibilidad temporal, no siendo ya adecuado aplicar la aproximación clásica de diseño de los sistemas de tiempo real, basada en el cálculo de peores tiempos de ejecución y asignación de recursos en el peor caso, que asegura determinismo temporal, pero que podría implicar, en un sistema dinámico, un gasto de recursos prohibitivo. En estos momentos se está realizando un esfuerzo por parte de la comunidad de tiempo real para la adaptación a entornos dinámicos [41], tanto a un nivel bajo de abstracción, donde es necesaria más investigación en algoritmos de gestión de recursos, nuevos modelos de tareas, control de admisión, monitorización y algoritmos de adaptación, como a alto nivel donde para obtener sistemas flexibles se deberán desarrollar protocolos que manejen niveles de calidad, así como arquitecturas que les den soporte.

Por otra parte, los sistemas distribuidos han experimentado una gran evolución desde su nacimiento, con la posibilidad de ejecutar remotamente aplicaciones en otros ordenadores conectados en red [230]. Desde mediados de los años setenta, la capacidad de procesamiento de los ordenadores ha aumentado notablemente y las redes de ordenadores se han popularizado, permitiendo la aparición de nuevos y más sofisticados paradigmas de distribución de procesos que a su vez favorecieron la aparición de nuevos paradigmas de programación, como pueden ser el *Distri-*

buted Shared Memory (DSM) (Memoria compartida distribuida mediante paginado) [161, 215] o la llamada a procedimiento remoto (RPC) [34], implementación del paradigma Cliente-Servidor.

La evolución experimentada por Internet en los últimos 20 años, ha propiciado que deje de ser una simple infraestructura que proporciona conectividad, con garantías *best-effort*, para pasar a ser un proveedor de aplicaciones, tales como sistemas de transmisión multimedia o *servicios Web* [5], que ofrecen y necesitan distintas calidades de servicio (*QoS*). Para garantizar dichas calidades de servicio, se han propuesto numerosas soluciones tanto a nivel de red (con arquitecturas como IntServ [113] y DiffServ [111], o protocolos como RSVP [112]) como de aplicación [158]. Otras soluciones [235, 88] han propuesto establecer el compromiso de provisión de *QoS* a través de un contrato a nivel de servicio (*Service Level Agreement contract*, SLA) [111] cuya violación puede suponer penalizaciones no tolerables. Estos esfuerzos están dirigidos a ofrecerle al usuario un servicio "personalizado", es decir, ajustado a sus necesidades reales y a una serie de requisitos impuestos por el propio usuario. Además, la evolución de Internet no se ha circunscrito a su funcionalidad, sino que también se ha extendido al modelo de distribución de dichos servicios: el modelo centralizado, donde un servidor central proveía los contenidos, se está rápidamente transformando en un modelo descentralizado en el que las aplicaciones finales ofrecidas al usuario hacen uso de servicios y contenidos alojados en distintos proveedores dispersos geográficamente.

Cabe destacar que, gracias a la evolución de la tecnología *hardware*, cada vez hay más dispositivos con capacidad de cómputo y comunicación que nos acompañan en todo momento, ofreciendo más funciones de las que en un principio tenían asociadas [48]: un teléfono móvil ya no sólo es un dispositivo desde el que podemos realizar llamadas, sino que además ofrece la posibilidad de escuchar música, ver vídeos, grabar conversaciones, gestionar una agenda o incluso acceder a Internet. Este nuevo entorno cambiante, saturado de elementos heterogéneos dispersos (desde teléfonos o PDAs a computadores con gran capacidad de cómputo, pasando por sensores y actuadores o dispositivos de función específica, como pueden ser una impresora o un frigorífico), de los que el usuario no es consciente pero de los cuales hace uso, hizo necesaria una evolución del paradigma de Computación distribuida, la Computación Ubicua [197], campo de investigación en el que se están realizando grandes esfuerzos hacia la visión ofrecida por Weiser en su artículo *"Computer for the 21st Century"* [236], en la que los dispositivos estén completamente integrados en el entorno (difuminándose y "desapareciendo" en él) y satisfagan las necesidades del usuario atendiendo no sólo a la funcionalidad y características propias del dispositivo (en algunos casos con fuertes limitaciones de procesamiento, memoria, batería, etc.) sino también al entorno y situación personal de dicho usuario.

Para adaptarse a este nuevo entorno, se está consolidando la utilización del paradigma de computación orientada a servicios (*Service Oriented Computing*, SOC) [103]. Este paradigma está basado en un modelo de computación distribuida

fundamentado en la existencia de proveedores y consumidores de servicios, y registros para la publicación y búsqueda de los mismos. Entendiendo como servicio [124] *"un dominio de control de una envergadura acotada que contiene un conjunto de tareas que cooperan para alcanzar objetivos relacionados"*. A partir de ahora, nos referiremos a servicio como a una entidad *software* autocontenida que proporciona una determinada funcionalidad.

El ámbito de aplicación de la SOC [9] son tanto entornos internos a una organización como entornos de computación colaborativa entre distintas organizaciones. Las principales características que debe cumplir una arquitectura que dé soporte a este paradigma son, según se expone en [103]:

- Acoplamiento débil entre proveedores y consumidores de servicios.
- Independencia con respecto a la implementación (lenguajes de programación, bibliotecas de código, plataformas de ejecución, etc.)
- Configuración flexible y dinámica.
- Robustez y fiabilidad.
- Modelado de interacciones a un nivel de abstracción alto.
- Colaboración.

Dado el planteamiento de los servicios como elementos básicos de este tipo de arquitecturas, una de las extensiones naturales al concepto es el de composición de servicios, que permite crear nuevos servicios y aplicaciones a partir de servicios ya existentes. Así, una entidad *software* puede proveer un servicio compuesto, basado en una combinación adecuada de otros servicios. Es decir, la entidad actúa al mismo tiempo como proveedora de un servicio y consumidora de otros, siendo un medio de añadir valor al entorno en el que se ejecuta. Dicha composición de servicios ya se aplica en numerosos entornos, como pueden ser los portales de información que agregan información; aplicaciones B2C que involucran una agregación de productos para ajustarse a las necesidades de un determinado usuario; empresas virtuales que hacen uso de los servicios que ofrecen otros muchos proveedores (desde servicios de red hasta de aplicación) para ofrecer, a su vez, otros nuevos servicios diferentes, etc.

Así, se puede observar que la utilización de la composición de servicios además de permitir decrementar el tiempo de desarrollo de un proyecto, a través del uso de piezas ya existentes para crear una mayor, permite una sencilla gestión de la *flexibilidad* de los servicios seleccionados, pudiendo *elegir* de entre un conjunto de proveedores aquél que, implementando la misma funcionalidad, sea el que más se ajuste a nuestras necesidades actuales, seleccionando otro proveedor cuando éstas cambien (*selección dinámica de servicios*). Esta flexibilidad puede ser utilizada para proporcionar *tolerancia a fallos* a nivel de aplicación, ya que servicios que ya no

funcionan pueden ser cambiados por otros y, además, este tipo de arquitecturas suele dar facilidades [103] para poder capturar y manipular el estado de una transacción. Al permitir agregar recursos, también puede ser usado para superar las restricciones impuestas por las características físicas de determinados dispositivos.

Existen entornos, como los sistemas de tiempo real distribuidos, que se podrían beneficiar de las características de este paradigma, pero en los que, debido a las restricciones que deben cumplir, como fiabilidad, predictibilidad y alta disponibilidad, tradicionalmente se han desarrollado de forma *ad hoc* y/o monolítica. Durante los últimos años, se ha experimentado un auge en la aplicación de la tecnología basada en componentes a sistemas de tiempo real [117, 231, 217], tanto a nivel de modelado [127, 85] como a nivel de lenguajes y plataformas [170, 169] como una forma de dotar a estos sistemas de cierta flexibilidad. Sin embargo, los trabajos de investigación relacionados con la aplicación de los conceptos relacionados con la computación orientada a servicios a este campo, son muy recientes, encontrándose en una fase muy preliminar de investigación, sin haber alcanzado todavía resultados relevantes. Se restringen a la identificación de problemas a resolver en una arquitectura genérica de tiempo real basada en servicios [228], y a la extensión de los protocolos de comunicación utilizados por los servicios Web para que puedan soportar tiempo real [99, 96], sin llegar en ningún caso a proponer una arquitectura completa de tiempo real que dé soporte a este paradigma.

En este documento se plantea la realización de una tesis doctoral en el campo de la aplicación de conceptos propios de la computación orientada a servicios a sistemas de tiempo real.

1.2. Motivación de la Tesis

Muchos de los sistemas de tiempo real actuales son de planificación estática, es decir, están formados por un conjunto de servicios fijos, previamente seleccionados, que son asignados a los recursos disponibles y, para asegurar que todas las restricciones temporales son satisfechas, se realizan comprobaciones previas al tiempo de ejecución. En contraste, los sistemas de tiempo real adaptativos del futuro necesitarán [41, 182] más flexibilidad durante la fase de ejecución, ajustando el funcionamiento del sistema a nueva información dinámica que procederá tanto del entorno en el que se ejecuta como del propio sistema. Bajo estas circunstancias, las demandas al sistema cambiarán dinámicamente y estos sistemas adaptativos deberán poder modificar el conjunto de servicios que soportan y extender sus funciones para poder satisfacerlas.

Como ya se comentó en el apartado anterior, el paradigma de computación orientada a servicios puede aportar esta flexibilidad a los sistemas, permitiendo la creación dinámica de aplicaciones a partir de servicios existentes remotos, pudiendo ser usada ventajosamente en el desarrollo de sistemas de tiempo real. Aplicándose en entornos tan dispares como sistemas de control de red, donde diferentes sensores y

filtros pueden ser vistos como servicios que pueden ser compartidos y actualizados *on-line*, o sistemas multimedia distribuidos en los que los servicios pueden corresponder a filtros y codificadores/decodificadores que ofrezcan diferentes calidades de servicios. Así, la composición dinámica de servicios puede ser interesante no sólo desde el punto de vista del desarrollo de aplicaciones, a través de la composición automática de elementos *software*, sino también como un medio para dar soporte a la actualización dinámica y reconfiguración de servicios, por ejemplo para la gestión dinámica de *QoS*, equilibrado de carga o tolerancia a fallos:

- *Actualización dinámica*: cada vez que una nueva versión de un servicio aparece en el sistema, ésta será analizada para comprobar si el uso de esta nueva versión en las aplicaciones presentes en el sistema mejora el rendimiento de las mismas de acuerdo con una métrica especificada previamente. Dependiendo de este análisis, el sistema podría cambiar las versiones de los servicios usadas y recomponer las aplicaciones que los usan.

- *Tolerancia a fallos*: las diferentes implementaciones de un servicio pueden ser usadas como *backups*, para asegurar la supervivencia del sistema si uno de los nodos involucrados se cae. En esta situación, frente a un fallo de una implementación de un servicio, un detector de fallos solicita su retirada y substitución por otra implementación disponible del mismo servicio, causando así la recomposición de las aplicaciones que usaban la implementación defectuosa.

- *Equilibrado de carga*: la carga excesiva en un determinado nodo puede provocar un degradamiento en las prestaciones ofrecidas por los servicios residentes en el mismo que, por su parte, pueden provocar un empobrecimiento en las prestaciones de una o varias aplicaciones. En esta situación, el sistema puede buscar otras implementaciones del mismo servicio alojadas en nodos diferentes que exhiban mejores calidades de servicio en ese momento. Así, el sistema podrá invocar la recomposición de la aplicación para explorar dichas alternativas, permitiendo maximizar sus prestaciones en cada momento. El análisis de tiempos de respuestas de ejecución en el peor caso puede ser usado para deducir la *QoS* actual ofrecida por una determinada implementación de un servicio como función de la carga en el nodo en el que está alojado.

Como un ejemplo práctico, considérese un sistema de visión artificial para inspección textil basado en un *cluster* de computadores en paralelo [50]. Cada nodo dedicado a la adquisición de imágenes envía cuadros a un nodo de computación de un conjunto de n. En este tipo de sistemas, a veces los nodos de adquisición están desocupados, por ejemplo cuando se cambian los rollos de tejido, provocando que la carga en los nodos de computación varíe. Usando un entorno que facilite la composición de aplicaciones basadas en servicios, podemos considerar el procesamiento de cada flujo de datos proveniente de cada cámara como una aplicación y distribuir la carga de los nodos de computación para mejorar las prestaciones del conjunto del

sistema. Otro ejemplo son los sistemas de alta disponibilidad, como los involucrados en el transporte de energía [65], en los que debe ser implementado un sistema que les proporcione tolerancia a fallos. Las soluciones *ad hoc* a nivel de aplicación carecen de la flexibilidad, reusabilidad y portabilidad que podría proporcionar una solución basada en la composición dinámica de servicios, pudiendo además resultar más caras. Por otra parte, las soluciones que sólo operan a nivel *hardware* o de sistema operativo no resuelven todos los problemas formales de estos sistemas como indica el argumento extremo a extremo (*end-to-end argument*) [196].

Dentro de las numerosas propuestas que intentan dotar de flexibilidad a sistemas distribuidos de tiempo real, como pueden ser la utilización de un paradigma determinado de comunicación [177, 132], el uso conjunto de la tecnología de componentes y aspectos [231, 217] para el diseño y desarrollo de sistemas, o *middleware* (*software* de intermediación o intermedio) de tiempo real como puede ser RTCORBA [169] o DRTSJ [126, 238, 237] (cuyo proceso de especificación está en este momento en curso), todavía no se han definido arquitecturas con características de tiempo real flexibles en las cuales se apliquen de manera ventajosa conceptos del campo de la orientación a servicios. Las primeras aproximaciones se han realizado desde grupos de investigación especializados en SOA, identificando qué características debería poseer una arquitectura de tiempo real que dé soporte a SOA, pero sin realizar una definición de los elementos que debería poseer [228]. También se han realizado esfuerzos en el campo de los servicios Web, para extender el protocolo utilizado en éstos para intercambiar datos, SOAP [37, 86], de tal forma que pueda soportar tiempo real, pero todavía no se llegado a una especificación [99]. Esto justifica la necesidad de la definición de métodos y metodologías, así como de una arquitectura de tiempo real genérica que aplique este paradigma.

En esta tesis se pretende realizar aportaciones en el ámbito de sistemas de tiempo real distribuidos para flexibilizarlos con conceptos y aportaciones técnicas inspirados en la computación orientada a servicios. En concreto, se propone la aplicación de los conceptos de SOA al campo de los sistemas de tiempo real, mediante la definición de un modelo de aplicaciones basadas en servicios y de una arquitectura genérica que les dé soporte, así como de los protocolos necesarios para ello. Asimismo, se integrará dicha arquitectura con mecanismos y protocolos de tiempo real ya existentes.

1.3. Objetivos y Retos

En esta tesis se pretenden realizar contribuciones en el ámbito de los sistemas de tiempo real distribuidos, incorporando conceptos de las arquitecturas orientadas a servicios, con el fin de aumentar la flexibilidad de sistemas distribuidos de tiempo real, en concreto de sistemas de tiempo real no crítico.

El modelo de aplicaciones basadas en servicios presenta varias ventajas relacionadas con el desarrollo de aplicaciones, tales como la reducción de tiempo de

desarrollo, la reutilización de código, la actualización dinámica del mismo y la facilitación de la replicación de la funcionalidad para implementar tolerancia a fallos. En esta tesis se pretende demostrar que también las aplicaciones de tiempo real, distribuidas o no, pueden beneficiarse de las mismas ventajas.

El objetivo que nos planteamos es definir un modelo de aplicaciones de tiempo real basadas en servicios y una arquitectura genérica que les dé soporte, así como integrar en la misma los protocolos necesarios para garantizar determinismo temporal en la comunicación. Asimismo, se realizará un prototipo de esta arquitectura integrada con un protocolo de comunicaciones de tiempo real para validar los conceptos que se desarrollen.

Así, los objetivos específicos de la tesis son los siguientes:

1. Estudio de las principales características de los sistemas distribuidos de tiempo real y sus arquitecturas. Asimismo, estudio de las principales características de los entornos distribuidos, prestando un especial interés a aquellos entornos distribuidos y abiertos que proporcionan la posibilidad de gestionar la calidad de servicio basándose en la selección dinámica de componentes.

2. Estudio y análisis de las diferentes propuestas de composición de aplicaciones basadas en servicios y de los problemas de integración de estas técnicas en el dominio de los sistemas de tiempo real.

3. Estudio de los diferentes protocolos de comunicaciones de tiempo real existentes y su adecuación para una arquitectura de tiempo real que haga uso de conceptos de la computación orientada a servicios.

4. Proposición de una arquitectura y de un modelo para la adecuada composición dinámica de aplicaciones de tiempo real en entornos embarcados distribuidos, así como de técnicas para la integración de dicha arquitectura con mecanismos de comunicación deterministas.

5. Proposición de algoritmos de composición adecuados para aplicaciones de tiempo real basadas en servicios. Estos algoritmos deberán proporcionar soluciones a la composición en un tiempo acotado, o al menos obtener cotas máximas de su tiempo de ejecución para un espacio acotado de búsqueda, de tal forma que sea posible su utilización en tiempo de ejecución en un entorno con restricciones temporales.

6. Validación del comportamiento de los algoritmos propuestos:

   - Simulación del comportamiento temporal de los distintos algoritmos propuestos en diferentes escenarios.
   - Obtención y validación de las cotas temporales halladas teóricamente.

- Evaluación del tipo de sistema al que pueden aplicarse dichos algoritmos, ya que su aplicación depende de las restricciones temporales del entorno y del comportamiento temporal de dichos algoritmos.

7. Validación del modelo y de la arquitectura propuestos:

   - Realización de las modificaciones necesarias en el protocolo de comunicaciones elegido para que dé soporte al prototipo.
   - Elaboración de un prototipo del sistema en una red de tiempo real.
   - Evaluación del prototipo.
   - Análisis de resultados.

1.4. Visión general de las contribuciones propuestas

El principal objetivo marcado en este trabajo es la aplicación de conceptos que introduzcan flexibilidad funcional en sistemas distribuidos de tiempo real. Concretamente, este trabajo aplica algunas ideas del paradigma de orientación a servicios a sistemas y arquitecturas distribuidos de tiempo real.

El camino elegido para realizar este trabajo fue la definición de una aproximación inicial a una arquitectura genérica distribuida de tiempo real que aplicase dichos paradigmas y la realización de un prototipo que validase todas las ideas propuestas. Asimismo otras contribuciones son la definición de algoritmos para la composición de funcionalidad y la extensión de un paradigma de comunicación de tiempo real. Una de las aportaciones a destacar es el hecho de que las propuestas realizadas han sido desarrolladas con una fuerte orientación a comunicación de tiempo real, es decir, que todo el análisis y las arquitecturas propuestas no se aíslan de los requisitos temporales de la red subyacente, sino que son influenciados por su existencia. Siendo ésta una constante en todo el documento.

En la figura 1.1, se muestra una de las arquitecturas propuestas durante la elaboración de la presente tesis doctoral.

En primer lugar, se definió un modelo de sistema de tiempo real sobre el que desarrollar esta arquitectura. Dicho modelo concibe las aplicaciones basadas en servicios como grafos dirigidos donde los servicios son los vértices y las interacciones entre éstos (mensajes) los arcos que los unen. Los servicios los define como funcionalidades que son implementadas por entidades *software* concretas, perfiles de servicio, que residen en nodos físicos dispersos en el sistema. A cada una de las invocaciones periódicas de los perfiles le hace corresponder una tarea instanciada en el nodo físico. Así, una aplicación en ejecución basada en servicios puede definirse como una transacción distribuida de tiempo real. En el modelo dichas aplicaciones serán gobernadas por tiempo.

Por otra parte, se realizó una identificación de los procesos y entidades que deben estar presentes en una arquitectura genérica de tiempo real de estas características.

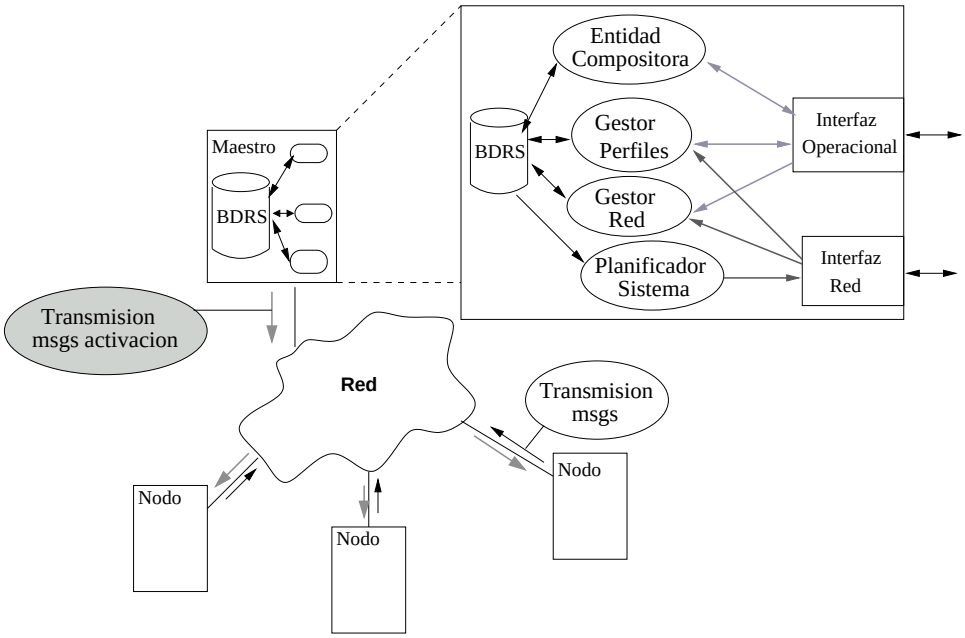


Figura 1.1: Propuesta de arquitectura

En primer lugar, deberá existir un registro o un directorio donde publicar las características de los perfiles. En la figura 1.1, esta información se guardará en la base de datos de requisitos del sistema (BDRS).

Los proveedores de servicios publicarán la información asociada a éstos en los directorios, pudiendo realizar esta publicación a través de *entidades de publicación*. En la figura, la funcionalidad de esta entidad es asumida por el *gestor de perfiles*, quien a través de la interfaz operacional o de la interfaz de red puede recibir datos sobre nuevos perfiles existentes en el sistema.

Por otra parte, los consumidores de los servicios deberán poder buscar perfiles de servicio atendiendo a su funcionalidad y utilizarlos en aplicaciones compuestas. En la arquitectura que se muestra en la figura, una *entidad compositora* a partir de la especificación dada por el cliente, elegirá los perfiles más adecuados para componer la aplicación. Esta composición es un problema complejo, pues a la complejidad de la elección de un conjunto de perfiles que cumpla unas determinadas características y restricciones, se le añade la necesidad de que este conjunto sea planificable y que no ponga en peligro las prestaciones del sistema completo. Así, para la composición inicial de aplicaciones se desarrollaron algoritmos que permiten a la entidad compositora realizar esta elección, cumpliendo los requisitos impuestos por el usuario y, además, asegurando la planificabilidad del sistema completo.

Por otra parte, una vez establecida la configuración de una aplicación, ésta se deberá ensamblar y desplegar en el sistema. Este cometido lo realiza el gestor del sistema, que recibe información del estado del mismo a través de monitorización. En la figura, la arquitectura propuesta descansa sobre un protocolo de comunicaciones

gobernado por tiempo, que sigue el esquema maestro-esclavo. En este paradigma, las activaciones de las tareas y mensajes son determinadas por el maestro, donde también se ubican todas las entidades vinculadas a la gestión y composición. Así, el maestro recibe información sobre perfiles que aparecen y desaparecen a través de la interfaz de red. Con esta información, el gestor de perfiles puede modificar la base de datos. El gestor de red es el encargado de distribuir el ancho de banda según los requisitos de las tareas y mensajes que aparecen reflejados en la BDRS, forzando el cumplimiento de las garantías temporales a través de la planificación de la red. Por último, la planificación del sistema completo que el maestro transmite a sus esclavos, la realiza el planificador del sistema

La propuesta de arquitectura genérica mostrada en la figura 1.1 fue adaptada al paradigma FTT, elaborándose como una extensión al mismo, portable entre las diferentes plataformas *hardware* en las que existen implementaciones disponibles de éste. Se elaboró un prototipo de dicha arquitectura, cuyos resultados experimentales muestran la validez de las ideas propuestas.

1.5. Organización de la memoria

La memoria está organizada en capítulos, que presentan las distintas partes de las que se compone la tesis:

- **Capítulo 2**: se presentan los tipos de sistemas que están más relacionados con este trabajo. Así como las técnicas, métodos e investigaciones más relacionadas con esta tesis.

- **Capítulo 3**: se introduce el modelo de sistema en el que se basó el trabajo desarrollado en esta tesis y el modelo de aplicación de tiempo real basada en servicios. Además, se presenta y caracteriza funcional y temporalmente el concepto de perfil de servicio, proponiéndose un esquema para la elección de un perfil determinado.

- **Capítulo 4**: este capítulo está dedicado a los requisitos que debería cumplir una arquitectura de tiempo real que soporte composición de aplicaciones basadas en servicios, presentándose una descripción funcional de dicha arquitectura y analizando los procesos principales que se ejecutarían en la misma. También se analizan los requisitos arquitectónicos para la composición, diferenciando entre composición dinámica o estática, para finalizar proponiendo para ambas aproximaciones, una posible arquitectura

- **Capítulo 5**: este capítulo se centra en los algoritmos para la composición inicial de aplicaciones. Se presenta, en primer lugar, la problemática de la composición de aplicaciones distribuidas de tiempo real, proponiéndose un algoritmo exhaustivo para realizar la búsqueda de una combinación inicial óptima

en términos de una figura de mérito. Como la excesiva complejidad computacional de un algoritmo de estas características, hace inviable su aplicación a contextos donde la composición inicial se realiza en tiempo de ejecución del sistema, se propone un algoritmo mejorado que ofrece una solución subóptima acotando el número de combinaciones a explorar.

- **Capítulo 6**: en este capítulo se presenta la validación de las ideas presentadas en forma de prototipo de la arquitectura para la composición dinámica sobre un paradigma concreto de comunicaciones tiempo real. El paradigma elegido es FTT, en concreto su implementación sobre *Switched Ethernet*, FTT-SE. Se exponen las razones de la elección realizada así como las características de FTT-SE. Posteriormente, se propone una arquitectura sobre dicho protocolo y se expone la implementación realizada del prototipo, así como los ajustes realizados en el protocolo de comunicaciones para que le ofrezca soporte.

- **Capítulo 7**: se presentan a grandes rasgos las conclusiones generales extraídas a lo largo de la elaboración de la misma, exponiéndose las principales contribuciones realizadas y se sugieren posibles líneas de trabajo futuras.

Capítulo 2

Estado del Arte

En este capítulo se realiza un estudio sobre las tecnologías más importantes relacionadas con las contribuciones propuestas en esta tesis doctoral. En primer lugar, se presentan los conceptos más importantes relacionados con los sistemas de tiempo real tanto centralizados (apartado 2.1), como distribuidos (apartado 2.2), haciendo especial énfasis en los algoritmos de planificación de tareas, cuya aplicación será necesaria durante la realización de la presente tesis doctoral. Seguidamente, en el apartado 2.3, se presenta el estado del arte de las redes y protocolos de comunicaciones de tiempo real más importantes. En el apartado 2.4 se exponen brevemente conceptos relacionados con las arquitecturas de calidad de servicio para sistemas embarcados. En los siguientes apartados, 2.5 y 2.6, se presentan sendos resúmenes de los principales conceptos de sistemas basados en componentes y del paradigma de orientación a servicios. Para finalizar con la presentación del estado de la técnica, en el apartado 2.7 se realiza un análisis de distintos trabajos relacionados íntimamente con el ámbito de la presente tesis.

2.1. Sistemas de tiempo real

Se entiende por sistema de tiempo real aquel sistema en el que el instante en el que se produce la salida es significativo, es decir, para que su funcionamiento sea correcto no basta con que las acciones sean correctas, sino que tienen que ejecutarse dentro del intervalo de tiempo especificado [40, 208]. Ese último instante de tiempo en el que un proceso puede completar su ejecución sin causar ningún daño al sistema se denomina *plazo*.

Dependiendo de la utilidad del resultado una vez pasado el plazo, las tareas de tiempo real se pueden clasificar [131, 203] como *firmes*, *flexibles* o *críticas*. Si una respuesta tardía carece de valor, el plazo será clasificado como firme, en otro caso como flexible. Cuando la pérdida de un plazo firme suponga consecuencias catastróficas para el sistema, éste se llama crítico. Un sistema de tiempo real que ejecute al menos una tarea con un plazo crítico es un sistema de tiempo real crítico, como por

ejemplo un sistema de control de vuelo. En otro caso, es un sistema de tiempo real flexible, como puede ser un sistema de adquisición de datos para una aplicación de control de procesos, o un sistema de procesamiento de datos multimedia.

2.1.1. Tareas de tiempo real

En el mundo real, las distintas operaciones ocurren en paralelo y un sistema que interaccione con él ha de imitar este comportamiento. La abstracción comúnmente utilizada consiste en la construcción del sistema como un conjunto de operaciones secuenciales que cooperan entre sí, interaccionando entre ellas y con el entorno [15, 31].

Dependiendo de sus características temporales, estas operaciones pueden clasificarse en: *periódicas*, su activación se produce con una periodicidad fijo, como el muestreo de un determinado dato a una determinada tasa; y *aperiódicas*, como la lectura de un sensor activada cuando se produce un cambio en el entorno. Dentro de este último tipo, las operaciones *esporádicas* son aquellas caracterizadas por tener períodos de activación variables. En este caso, se suele dar una medida de la tasa de llegada al sistema. Habitualmente, la separación mínima, T_{min}, entre dos invocaciones sucesivas.

En un sistema de tiempo real, estas operaciones o procesos suelen ser mapeadas a tareas *software*. Por tanto, las tareas representan actividades manejadas por el sistema *software*. Habitualmente, un sistema *software* ejecuta diferentes actividades, pudiendo cada una de ellas tener diferentes restricciones temporales. Algunas de estas tareas son completamente independientes de las demás, otras deben realizarse en un determinado orden, o deben compartir el acceso a diferentes recursos no sólo al procesador.

A la hora de implementar un sistema, se ha de poseer un conocimiento mínimo sobre las propiedades de cada una de las tareas. Un conjunto de N tareas periódicas Π se puede caracterizar como:

$$\Pi = \{\tau_i = (C_i, D_i, T_i, \phi_i, p_i), i = 1 \ldots N\} \tag{2.1}$$

donde,

- C_i es el tiempo de computación en el peor caso de la tarea τ_i,(también llamado WCET, *Worst Case Execution Time*)
- D_i es el plazo relativo de la tarea τ_i. Una vez la tarea es invocada, debe terminar, como mucho, en D_i unidades de tiempo, con $D_i \leq T_i$.
- T_i es el período de la tarea τ_i. Es el tiempo entre dos invocaciones sucesivas de la tarea τ_i. Cada tarea periódica puede producir una secuencia (posiblemente) infinita de invocaciones separadas T_i unidades de tiempo.

- ϕ_i es el desplazamiento de la tarea τ_i, con $\phi_i < T_i$. Es el desplazamiento respecto a $t = 0$ del momento de invocación de la tarea. La primera invocación de la tarea se realiza en $t = \phi_i$, y las siguientes estarán separadas T_i unidades de tiempo.

- p_i es la prioridad de la tarea τ_i, siendo p_1 la más alta y p_N la más baja. Las técnicas tradicionales de planificación usaban como prioridad el subíndice de la tarea, pero esta notación deja de ser válida cuando se extienden estas técnicas y se permite que distintas tareas tengan la misma prioridad, o que una misma tarea tenga dos prioridades distintas. De ahí, la necesidad de introducir la prioridad como otro parámetro.

Para la invocación k–ésima de la tarea periódica τ_i, su instante de activación ($a_{i,k}$) y el valor absoluto de su plazo ($d_{i,k}$) pueden ser calculados como

$$\begin{array}{rcl} a_{i,k} & = & \phi_i + (k-1)T_i \\ d_{i,k} & = & a_{i,k} + D_i \end{array} \tag{2.2}$$

Para caracterizar tareas esporádicas se puede emplear la misma notación, utilizando el período mínimo de activación, $T_{min,i}$, en lugar del período y dejando sin definir el desplazamiento temporal:

$$\sigma_i = (C_i, D_i, T_{min,i}, p_i), i = 1 \ldots M \tag{2.3}$$

En este caso, el instante de activación y el valor absoluto del plazo pueden ser calculados como:

$$\begin{array}{rcl} a_{i,k} & \geq & a_{i,k-1} + (k-1)T_{min,i} \\ d_{i,k} & = & a_{i,k} + D_i \end{array} \tag{2.4}$$

2.1.2. Planificación de sistemas de tiempo real

En un sistema de tiempo real es necesario asignar un correcto reparto de recursos entre las diferentes tareas del mismo, con el objetivo de asegurar que se cumplan sus requisitos temporales. Para llevar a cabo esta tarea es insuficiente someter a tests al sistema, ya que un test puede demostrar la existencia de un error pero no la ausencia completa de éstos. En definitiva, se debe probar que el sistema es correcto. En este marco surge la *Teoría de Planificación* [200] como la base teórica necesaria para probar la corrección de un sistema de tiempo real.

Se define *algoritmo de planificación* como el conjunto de reglas que determinan las decisiones que se han de tomar a lo largo de la vida del sistema para que, usando el mínimo de recursos, todas las tareas cumplan sus requisitos temporales o de otra índole, como pueden ser las restricciones de precedencia. Estas decisiones se refieren tanto al orden temporal en el que las tareas acceden al procesador o a otros

recursos, como al tiempo durante el que los usan. El término *planificación* se referirá al proceso de seleccionar una tarea concreta para su ejecución en un determinado instante temporal.

La problemática de la planificación se dividirá entonces en dos grandes subproblemas: por una parte, se ha de comprobar que el sistema es planificable, es decir, si existe una planificación para un sistema dadas unas tareas con sus requisitos temporales y un determinado algoritmo y, una vez comprobado, encontrarla. Al primer problema se le llama *análisis de planificabilidad* y al segundo *construcción de la planificación*.

El análisis de planificabilidad para un determinado algoritmo se basa en tests. El resultado de éstos debe reflejar la capacidad de ese algoritmo para encontrar una planificación factible dado un conjunto de tareas. Sin embargo, la precisión de estos tests depende en gran medida de su complejidad, siendo clasificados en *exactos*, *suficientes* y *necesarios* [225]. Los más complejos computacionalmente son los tests *exactos*, aquéllos que dan un resultado positivo siempre que exista una planificación posible para el sistema y negativo en caso contrario. Desgraciadamente, la mayor parte de este tipo de tests son computacionalmente intratables [82], o comportan una carga computacional excesiva que reduce su aplicación a las fases *off-line* del sistema. Más sencillos de implementar son los tests denominados *suficientes*, aquéllos cuyo resultado positivo asegura la existencia de una planificación posible, pero si es negativo no aseguran que no la haya. Nótese que este tipo de tests puede, por lo tanto, rechazar conjuntos de tareas planificables. Los tests denominados *necesarios* sólo aseguran con un resultado negativo la imposibilidad de planificar el conjunto de tareas, mientras que si es positivo puede ocurrir que éstas no sean planificables. Por otra parte, principalmente hay dos familias de tests aplicados a sistemas de tiempo real, basada cada una de ellas en conceptos diferentes [172]:

- Límite de utilización: entendiendo por utilización el porcentaje de ocupación del procesador. Cada tarea periódica τ_i carga el procesador con una utilización dada por:
$$U_i = \frac{C_i}{T_i} \tag{2.5}$$
- Tiempo de respuesta: se centran en el cálculo de los tiempos de respuesta en el peor caso de todas las tareas del sistema que comparará *a posteriori* con sus plazos respectivos.

Los algoritmos de planificación, por su parte, pueden ser clasificados (ver cuadro 2.1) como estáticos o dinámicos. Los algoritmos *estáticos* son aquellos en los que todas las decisiones de planificación se realizan antes de la ejecución del sistema, también llamados por esto *off-line*; los *dinámicos* o algoritmos *on-line*, dan libertad al planificador para decidir en tiempo de ejecución cuándo una tarea entra al procesador. Además, dentro de los algoritmos *on-line* podemos distinguir entre aquellos en los que las prioridades de las tareas son determinadas *off-line*, es decir,

son estáticas y los que, dado un cierto criterio, pueden modificar las prioridades de forma dinámica.

Por otra parte, dentro de los algoritmos dinámicos, el momento en el que una tarea de menor prioridad debe dejar el procesador los divide entre algoritmos *con apropiación* y *sin apropiación*. Si la decisión de quién entra en el procesador se pospone hasta la completa ejecución de las tareas, el algoritmo será sin apropiación, mientras que si es posible que una tarea vea interrumpida su ejecución, debido a que otra de mayor prioridad pasa a estar preparada, el algoritmo será con apropiación del procesador.

$$
\begin{array}{l}
\textit{Atendiendo} \\
\textit{momento} \\
\textit{decisiones} \\
\textit{de planificación}
\end{array}
\left\{
\begin{array}{l}
Off-line \left\{ \; Ejecutivo\ Cíclico \right. \\
\\
On-line \left\{
\begin{array}{l}
Prioridades\ estáticas \left\{ \begin{array}{l} RMS \\ DMS \end{array} \right. \\
\\
Prioridades\ dinámicas \left\{ \begin{array}{l} EDF \\ LLF \end{array} \right.
\end{array}
\right.
\end{array}
\right.
$$

$$
\begin{array}{l}
\textit{Atendiendo} \\
\textit{posibilidad de} \\
\textit{liberación del} \\
\textit{procesador}
\end{array}
\left\{
\begin{array}{l}
Sin\ apropiación \\
\\
Con\ apropiación
\end{array}
\right.
$$

Cuadro 2.1: Clasificación algoritmos planificación y ejemplos para planificación centralizada

2.1.3. Planificación centralizada

Las técnicas de planificación tradicionales consideran la existencia de un único procesador en el que las tareas son independientes. Las invocaciones de las tareas no dependen del inicio o conclusión de las demás tareas del sistema, y no hay relaciones de precedencia o de exclusión. Además, las tareas no se suspenden así mismas en el medio de su ejecución. También se asume que los tiempos de cambio de contexto, u otros tiempos inherentes al funcionamiento del sistema son ignorados (asumidos como nulos, o incluidos en los tiempos de ejecución en el peor caso de las tareas).

2.1.3.1. Planificación estática: ejecutivos cíclicos

Como vimos anteriormente, la planificación estática u *off-line* se caracteriza porque todas las decisiones de planificación se toman antes de la ejecución del sistema. El plan de ejecución resultante se almacena en una tabla estática que contiene la lista de las tareas, todas ellas periódicas, y los instantes en los cuales deben activarse y

finalizar. En tiempo de ejecución, el planificador, un ejecutivo cíclico [22], lee de forma secuencial la lista y activa las tareas en los instantes apropiados.

Como se puede intuir, el funcionamiento del planificador es muy sencillo, robusto y eficiente, pues su ejecución apenas supone carga computacional en el sistema. Además, este método es determinista y predecible [244], ya que se sabe con exactitud lo que ejecuta el procesador en cada momento. Sin embargo, esta aproximación produce sistemas inflexibles y difíciles de mantener [40].

El primer problema consiste en generar el plan de ejecución del sistema, habiendo dividido los procesos en secciones de código tales que se puedan ejecutar en una ranura temporal, sabiendo sus períodos y las restricciones temporales, plazos, que deben cumplir. Éste es un problema NP-Completo [31]: existen casos particulares en los que la planificación puede ser construida en tiempo polinómico, pero en la mayoría ha de abordarse a través del uso de heurísticos [39, 224, 244]. Por otra parte, el plan de ejecución tiene que elaborarse para un intervalo temporal igual al mínimo común múltiplo de los períodos de todas las tareas, lo que puede dar como resultado una tabla de planificación de longitud intratable. Una forma de solventar este último problema consiste en transformar los períodos de las tareas, obligando a que sean armónicos, incrementando artificialmente la utilización del procesador. Además de todos estos inconvenientes, cada vez que existe un cambio en el conjunto de las tareas, como puede ser el incremento de tiempo de ejecución de una de ellas o la irrupción de una nueva tarea en el sistema, se ha de volver a construir completamente el plan de ejecución, llegando incluso a tener que volver a dividir los procesos de una manera totalmente diferente a la anterior. Otro problema asociado a los ejecutivos cíclicos [147] es la dificultad de incluir en la planificación tareas no periódicas, debido al carácter estático del plan de ejecución.

Esta aproximación fue la utilizada hasta los años ochenta, produciendo sistemas inflexibles y difíciles de mantener.

2.1.3.2. Planificación dinámica

Una aproximación bastante más adecuada es la planificación en tiempo de ejecución o planificación dinámica. En ella, a cada tarea se le asigna una prioridad y, atendiendo a éstas, el planificador elegirá en cada momento la tarea más urgente, no siendo necesario así un plan de ejecución previo. Si la prioridad de una tarea no cambia una vez asignada, se hablará de prioridades fijas y, en caso contrario, de prioridades dinámicas. En el trabajo de Liu y Layland [146] se proponen dos de los más importantes algoritmos de planificación para tareas independientes en sistemas monoprocesador. El método de asignación de prioridades estáticas en función de los períodos de las tareas (*Rate Monotonic Scheduling* RMS), y el método de asignación dinámica de mayor prioridad a la tarea con el plazo más próximo (*End Deadline First* EDF). La importancia de estos algoritmos reside en el hecho de que son óptimos en su clase [176]. Si dado un conjunto de tareas algún algoritmo de la clase es capaz de generar una planificación admisible, el óptimo también lo hará.

Planificación dinámica basada en prioridades fijas Para poder tratar analíticamente el problema de la planificación admisible en un conjunto de tareas de tiempo real, el trabajo de Liu y Layland [146] se basa en las siguientes suposiciones que crean un modelo bastante restringido:

1. Todas las tareas críticas del sistema son periódicas.
2. El plazo de una tarea es igual a su período ($T_i = D_i$).
3. Las tareas del sistema son independientes entre sí, es decir no mantienen relaciones de precedencia, no comparten recursos ni se comunican.
4. El tiempo de ejecución de cada tarea es constante para esa tarea y no varía en tiempo de ejecución, entendiendo como tiempo de ejecución como el tiempo que le llevaría al procesador ejecutar dicha tarea sin interrupción.
5. Las tareas no se suspenden mientras ejecutan el código correspondiente a una activación.
6. Las tareas no periódicas del sistema son especiales: son rutinas de inicialización o de recuperación de fallos. Desplazan a las tareas periódicas mientras están ejecutándose y no tienen plazos de respuesta críticos.
7. Las tareas se ejecutan con un planificador basado en prioridades y expulsivo.

Basándose en las suposiciones anteriores, se enunció el método de planificación de prioridad a la tarea más frecuente (RMS). De acuerdo con el algoritmo RMS, las prioridades se asignan monotónicamente con respecto a los frecuencia de las tareas; cuanto mayor sea la frecuencia de una tarea (menor período), mayor será la prioridad:

$$\forall \tau_i, \tau_j \in \Pi : T_i < T_j : p_i > p_j \tag{2.6}$$

Además, se enunciaron las siguientes definiciones y teoremas:

Definición 2.1 (Liu y Layland, 1973). El instante crítico de una tarea se define como el instante en que una activación de dicha tarea tiene el tiempo de respuesta más largo.

Basándose en la anterior definición, se enuncia el siguiente teorema:

Teorema 2.1 (Liu y Layland, 1973). El instante crítico para una tarea ocurre cuando su activación coincide con la activación de todas las tareas de mayor prioridad que ella.

Si todas las fases de las tareas del sistema son nulas ($\phi_i = 0 \forall i = 0 \ldots n$), el instante crítico para todas las tareas del sistema tendrá lugar durante el inicio del mismo.

Teorema 2.2 (Liu y Layland, 1973). Si existe una asignación de prioridades a un conjunto de tareas con las características enumeradas en las suposiciones previas que produzca una planificación admisible, la planificación de prioridades RMS también será admisible.

Aseverar que una planificación es admisible implica que el conjunto de tareas cumple sus plazos de respuesta bajo cualquier circunstancia, debiendo comprobarse la planificabilidad en el caso más desfavorable, es decir, en el instante crítico de cada una de las tareas.

El test de planificabilidad de RM propuesto en el trabajo de Liu y Layland se basa en la definición de la tasa de utilización del procesador:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.7}$$

Teorema 2.3 (Liu y Layland, 1973). Sea $\Pi = \{\tau_i, i = 1 \dots N\}$ un conjunto de tareas con prioridades, entonces el conjunto es planificable, para cualquier relación de fases, si:

$$U = \sum_{i=1}^{N} \frac{C_i}{T_i} \leq N(2^{\frac{1}{N}} - 1) \tag{2.8}$$

Este teorema afirma que si la utilización del procesador es menor que un determinado umbral, entonces el conjunto de tareas será planificable. En particular, para $N = 2$ el umbral es de $0{,}8383$ y para valores más altos de N:

$$\lim_{N \to \infty} N(2^{\frac{1}{N}} - 1) = \ln 2 \simeq 0{,}69 \tag{2.9}$$

Este test es suficiente pero no necesario, existiendo conjuntos de tareas planificables pero que fallan el test.

Las suposiciones en las que se basa RMS generan un modelo de sistema poco realista, que reduce su aplicabilidad. Este hecho ha provocado la aparición de otros trabajos orientados a extender RMS a entornos menos restrictivos. Entre ellos destacan los métodos de análisis de planificabilidad precisos [141], o la posibilidad de comunicar tareas [202, 83].

En Lehoczky et al. [141] se presenta una técnica para el análisis exacto de planificabilidad bajo RMS. Ésta se basa en la comprobación de que la carga solicitada al procesador sea menor que el tiempo transcurrido entre la activación de la tarea y su finalización. Como no es viable realizar una comprobación en todos los instantes temporales, en el mismo trabajo se llega a la conclusión de que sólo han de comprobarse para cada tarea los instantes temporales en los que tareas de mayor o igual prioridad son activadas. A este conjunto de instantes lo denominaron *puntos de planificación*.

Teorema 2.4 (Lehoczky et al, 1990). Sea $\Pi = \{\tau_i,\, i = 1 \ldots n\}$ un conjunto de tareas ordenadas en orden decreciente de prioridades. El conjunto de puntos de planificación para una tarea $\tau_i \in \Pi$ coincidirá con los momentos en los que las tareas de mayor prioridad que ella son activadas y se define como:

$$S_i = \left\{ kT_j \,|\, j = 1 \ldots i,\; k = 1 \ldots \left\lfloor \frac{T_i}{T_j} \right\rfloor \right\} \tag{2.10}$$

Dicha tarea cumplirá todos sus plazos de respuesta para cualquier relación de fases si:

$$\min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lfloor \frac{t}{T_j} \right\rfloor \leq 1 \tag{2.11}$$

La carga en cualquier instante de tiempo debida a tareas de mayor o igual prioridad que la tarea τ_i o, lo que es lo mismo, el tiempo de computación pedido entre $[0, t)$ por todas las tareas de prioridad mayor que τ_i se define como:

$$W_i(t) = \sum_{j=1}^{i} C_j \left\lceil \frac{t}{T_j} \right\rceil \tag{2.12}$$

Siendo la utilización en el peor caso ($U_i(t)$), esta carga $W_i(t)$ entre la anchura.

$$U_i(t) = \frac{\sum_{j=1}^{i} \left(C_j \left\lceil \frac{t}{T_j} \right\rceil \right)}{t} \tag{2.13}$$

Por tanto, la condición expresada en la inecuación 2.11 se podrá reescribir como:

$$\min_{t \in S_i} U_i(t) \leq 1 \tag{2.14}$$

De esta ecuación se deduce que el tiempo de computación debido a tareas de mayor o igual prioridad que τ_i para que ésta sea planificable debe ser menor que el tiempo transcurrido hasta el momento. Así, el sistema será planificable si y sólo si se cumple dicha condición para todas las tareas:

$$\max_{\forall i = [1 \ldots n]} \left\{ \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \right\} \leq 1 \tag{2.15}$$

La inecuación resultante se puede aplicar de forma práctica ya que el algoritmo sólo realizará la comprobación en los puntos de planificación de cada tarea.

Como ya se comentó, las hipótesis en las que se basa RMS son muy restrictivas, creando un modelo de sistema poco realista. En primer lugar, la restricción impuesta a las tareas de que el plazo ha de ser igual al período ($D_i = T_i \,\forall i$) es muy severa. Existen numerosos sistemas en los que el plazo de finalización es menor que el

período. Por ejemplo, un sistema de muestreo en el que es importante que cada muestra esté separada el período de muestreo, siendo el plazo de finalización el margen de tolerancia con respecto al instante de toma de cada muestra. En esta clase de sistemas ya no es posible aplicar RMS, resultando necesario el desarrollo de nuevos métodos. Además, los tests hasta ahora explicados, basados en la utilización del procesador, no dan información acerca de los plazos de respuesta de las tareas, siendo necesario este dato en sistemas como el del ejemplo, en el que el plazo de respuesta es una restricción temporal en sí misma que ha de cumplirse.

En este marco, en el trabajo de Leung y Whitehead [142] se definió el método de prioridad a la tarea más urgente (*Deadline Monotonic Scheduling*, DMS):

Teorema 2.5 (Leung y Whitehead, 1982). Si existe una planificación de prioridades de un conjunto de tareas con las características definidas en el método RMS que produzca una planificación admisible, entonces la asignación de prioridades mayores a las tareas con plazos de respuesta menores, es decir, más urgentes, también será admisible.

Como se puede observar, RMS es un caso particular del método DMS cuando $D_i = T_i$. En este mismo trabajo, se demostró que cuando los plazos de finalización son menores o iguales que los períodos ($D_i \leq T_i$) la asignación DMS es óptima.

Audsley [13] desarrolló un test de planificabilidad suficiente basado en utilizaciones para DMS:

Teorema 2.6 (Audsley, 1991). Sea $\Pi = \{\tau_i,\, i = 1 \ldots N\}$ un conjunto de tareas ordenadas en orden decreciente de prioridades. La interferencia que la tarea τ_i experimentará debido a tareas con igual o mayor prioridad que ella está acotada por la expresión:

$$I_i = \sum_{j \in h_p(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \tag{2.16}$$

donde $h_p(i)$ es el conjunto de tareas que pueden interferir en la ejecución de τ_i, formado por aquellas con mayor o igual prioridad que ésta (excepto la propia τ_i).

El conjunto de tareas será planificable si cada una de ellas experimenta una utilización máxima definida dentro de su plazo menor al $100\,\%$:

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} \leq 1,\ \forall i = 1 \ldots N \tag{2.17}$$

El análisis exacto se obtiene extendiendo el test desarrollado por Lehocky et al [141] (ver teorema 2.4) al caso DMS, redefiniendo la utilización en el peor caso (ecuación 2.13) como:

$$U_i(t) = \frac{C_i \left\lceil \frac{t}{T_i} \right\rceil + \sum_{j \in h_p(i)} \left(C_j \left\lceil \frac{t}{T_j} \right\rceil \right)}{t} \tag{2.18}$$

También redefinen los puntos donde ha de comprobarse el test (ecuación 2.10):

$$Q_i = \left\{ kT_j \,|\, j \in h_p(i),\, k = 1 \ldots \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\} \cup \{D_i\} \tag{2.19}$$

Dentro de los métodos basados en el cálculo del tiempo de respuesta en el peor caso (WCRT, *Worst Case Response Time*), el método desarrollado por Joseph y Pandya en [125] y por Audsley et al [13] permiten estudiar conjuntos de tareas con plazos menores o iguales al período ($D_i \leq T_i$) y con cualquier asignación arbitraria de prioridades. Este método es equivalente al desarrollado por Harter [97, 98], quien en 1984 fue el primero en usar el tiempo de respuesta en el peor caso de una tarea. Harter desarrolló el Algoritmo de Dilación Temporal (*Time Dilation Algorithm*) que usa lógica temporal para obtener tiempos de respuesta.

El peor tiempo de respuesta, R_i, de una tarea τ_i es el menor $w \geq 0$ tal que:

$$w = C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w}{T_j} \right\rceil C_j \tag{2.20}$$

La planificabilidad del sistema viene condicionada a que exista una solución a la ecuación anterior, $\exists R_i \forall i = 1 \ldots N$ y a que $R_i < D_i \forall i = 1 \ldots N$. La solución a dicha ecuación no puede darse directamente sino de forma iterativa [17, 18]:

$$\begin{cases} w^0 & = & C_i \\ w^{n+1} & = & C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w^n}{T_j} \right\rceil C_j \end{cases} \tag{2.21}$$

Ésta es una serie monótona no decreciente, que converge a un valor ($w^{n+1} = w^n = R_i$) siempre y cuando $U \leq 1$ tal y como se demostró en [125].

Planificación dinámica basada en prioridades dinámicas En el mismo trabajo en el que se propone RMS [146] y bajo las mismas suposiciones, se introduce el algoritmo EDF (*Earliest Deadline First*) basado en la proximidad de los plazos de finalización, de forma que la tarea elegida para ejecutar en cada momento es aquella que tenga su plazo de finalización más próximo:

$$\forall \tau_i, \tau_j \in \Pi_R : d_i < d_j : p_i > p_j \tag{2.22}$$

donde Π_R es el subconjunto de Π que comprende las tareas preparadas para ejecutarse y (d_i,d_j) son los plazos absolutos de las tareas τ_i y τ_j.

En el mismo trabajo se enunció el siguiente teorema:

Teorema 2.7 (Liu y Layland, 1973). Sea $\Pi = \{\tau_i, i = 1 \ldots N\}$ un conjunto de tareas planificadas según EDF. Las tareas serán planificables si y sólo si:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \tag{2.23}$$

Puede observarse que la prioridad varía a medida que pasa el tiempo, siendo necesario un reordenamiento de las tareas cada vez que una nueva es activada. Así, la complejidad temporal de este algoritmo es de $O(n \log n)$. Comparándolo con RMS, EDF aumenta considerablemente la sobrecarga introducida por el planificador, pero es capaz de conseguir factores de utilización más altos y el número de apropiaciones es potencialmente menor. Así, la elección entre uno y otro dependerá del caso concreto.

En 1974, Dertouzos [66] demostró que EDF es óptimo entre todos los algoritmos de planificación con apropiación, en el sentido de que si alguno de ellos produce una planificación admisible, EDF también lo hará. Más tarde, Mok propuso otro algoritmo óptimo [154, 153] el del método de prioridad a la tarea con la menor holgura (*Least Laxity First* o LLF). En él, la holgura de una tarea en un cierto instante de tiempo se define como la diferencia entre su plazo de finalización absoluto y el tiempo estimado de finalización en el peor caso. LLF presenta propiedades similares a las de EDF, pero introduce una mayor sobrecarga en tiempo de ejecución, debido a los cambios de contexto provocados por cambios en la holgura de las tareas en tiempo de ejecución. Por esta razón, la comunidad de tiempo real se centró en el estudio de EDF, relajando las restricciones iniciales impuestas por Liu y Layland y extendiendo los análisis de planificabilidad a casos más generales. Entre estas técnicas destacan el análisis de demanda de procesador [23, 24] (*Processor Demand Analysis*), la planificación de tareas aperiódicas basándose en estructuras denominadas servidores (como el método *Total Bandwidth Server* [205, 206], TBS, o el método *Constant Bandwidth Server* [2, 1], CBS) técnicas para la correcta gestión de las sobrecargas en tiempo de ejecución (como el algoritmo *Capacity Sharing* [45] o CASH, el algoritmo *Greedy Reclamation of Unused Bandwidth* [144], GRUB, o la planificación elástica [44, 42, 43]), etc. En [192] puede encontrarse un resumen de las técnicas de análisis de planificabilidad para algoritmos de planificación basados en prioridades dinámicas.

2.1.3.3. Comunicación entre tareas

Los métodos de planificación anteriores presuponen que las tareas son independientes y que el único recurso compartido es el procesador. Estas suposiciones son demasiado restrictivas porque la interacción entre las tareas es necesaria en casi cualquier aplicación no trivial, y porque también existen otros recursos, como estructuras de datos en memoria, que también son de acceso exclusivo.

La comunicación entre procesos produce bloqueos de tareas: una tarea se bloquea cuando al solicitar un acceso a un recurso compartido con requisitos de exclusión mutua, éste está siendo utilizado por otra tarea. En un sistema de tiempo real, la utilización de los métodos de sincronización tradicionales, como pueden ser semáforos [67], monitores [100] o los distintos tipos de *rendez-vous* [40, 101], se ha demostrado que pueden volver difícil de analizar al sistema resultante [153], e incluso pueden provocar la aparición del fenómeno de *inversión de prioridades*: que una

tarea sea bloqueada por otra de menor prioridad, debiendo esperar a que la de menor prioridad abandone la zona de exclusión mutua para poder seguir su ejecución.

En este contexto, aparecen los algoritmos de sincronización de tiempo real, que tratan el problema de la inversión de prioridades mediante el establecimiento de una cota admisible de tiempo de inversión.

Herencia de prioridad El protocolo estándar de herencia de prioridad (*Priority Inheritance Protocol*, PIP) fue propuesto por Sha *et al* en [202]. Este protocolo propone que si una tarea bloquea a otra más prioritaria, herede dinámicamente la prioridad de la tarea bloqueada, restaurándose su prioridad una vez abandonado el recurso compartido.

Este protocolo presenta numerosas carencias [201]: el número de veces que una tarea puede verse bloqueada es alto; una tarea puede sufrir *bloqueos encadenados* (bloqueos excesivamente largos debido a cadenas de bloqueos entre tareas); y el protocolo no previene los *interbloqueos* (dos tareas que se bloquean una a la otra, esperando mutuamente por un recurso al que la otra está accediendo). Todos estos problemas los solucionan los protocolos de techo de prioridad.

Protocolo del techo de prioridad En [83, 202] se propone el protocolo de techo de prioridad (*Priority Ceiling Protocol*, PCP). Este protocolo tiene en cuenta las prioridades de las tareas que pretenden acceder a un recurso determinado asignándole al semáforo del recurso un *techo de prioridad*, coincidente con el valor máximo de la prioridad de las tareas que lo utilizan. Por otra parte, una tarea tiene una prioridad dinámica que es el máximo entre su prioridad estática y la que hereda por estar bloqueando a tareas más prioritarias que ella. El techo del sistema es el máximo de todos los techos de prioridad del conjunto de semáforos bloqueados. Una tarea determinada no podrá bloquear un semáforo libre a menos que su prioridad sea estrictamente mayor que el techo del sistema. El efecto del protocolo es tal que si tenemos un sistema en el cual está bloqueado sólo un semáforo, un segundo semáforo sólo puede ser bloqueado si no existe otra tarea de mayor prioridad que use ambos semáforos. Este protocolo previene el interbloqueo y cumple la propiedad de bloquear a una tarea como máximo una vez en cada invocación.

Uno de los principales problemas de este protocolo es el coste de su implementación y la sobrecarga en tiempo de ejecución. El planificador debe guardar la traza de qué tarea está bloqueada en qué semáforo y de cuáles son las prioridades heredadas. Además, la determinación de si una tarea puede o no bloquear un semáforo libre también consume recursos.

Herencia inmediata del techo de prioridad Este protocolo (*Immediate Priority Ceiling Protocol*, IPCP) es una variante del anterior [130, 201] más sencillo de implementar y con menos sobrecarga en tiempo de ejecución (menos cambios de contexto). Permite que una tarea que accede a un recurso herede el techo de prioridad

del mismo y baja su prioridad cuando lo deja. Esto lleva a que las tareas pueden ser innecesariamente retrasadas por tareas de menor prioridad pero, en el peor caso, una tarea sólo puede ser bloqueada una vez y se mantiene la propiedad de evitar el interbloqueo.

Control de semáforos Pretende evitar [186] las fuertes restricciones que impone el techo de prioridad en algunas situaciones. Como consecuencia se consigue que las tareas sean bloqueadas menos veces. Sin embargo, tiene un coste considerable en tiempo de ejecución.

Stack Resource Policy Éste es un protocolo de control de concurrencia propuesto por Baker [21] en 1991 para acotar la inversión de prioridad tanto en sistemas que utilizan prioridades estáticas como dinámicas. En este trabajo, se observó que utilizando EDF, las tareas con un plazo relativo muy grande pueden retrasar, pero nunca apropiarse del procesador cuando existen tareas con un plazo más corto. Consecuencia directa de esta observación es que una tarea no puede bloquear a otra con un plazo relativo mayor. Lo que lleva a la conclusión de que, utilizando EDF, sólo es necesario preocuparse de los bloqueos que realizan tareas con un plazo relativo grande a otras con un plazo relativo más corto. Así, definió niveles de apropiación de forma separada de las prioridades bajo EDF, de tal manera que eran inversamente proporcionales a sus plazos relativos. SRP garantiza que una vez que una tarea empieza, no se bloqueará hasta que se complete, y sólo pueden apropiarse del procesador tareas con una prioridad mayor. Por otra parte, previene los abrazos mortales, y acota el tiempo durante el cual una tarea puede ser bloqueada al que dure una sección crítica. No se necesitan colas de espera, de hecho una tarea nunca se bloquea durante su ejecución, simplemente no puede empezar a ejecutarse si su nivel de apropiación no es lo suficientemente alto. Además, SRP permite que las tareas compartan la pila en tiempo de ejecución, lo que constituye un gran ahorro en consumo de memoria si hay muchas más tareas que niveles de prioridad relativos.

Frente a otros protocolos utilizados para evitar la exclusión mutua en sistemas que utilizan EDF, como el protocolo de herencia dinámica de prioridad (*Dynamic Priority Inheritance* [207]), el protocolo de techo de prioridad dinámico (*Dynamic Priority Ceiling* [54]) o el protocolo de modificación dinámica de los plazos (*Dynamic Deadline Modification* [119]), SRP es uno de los más usados debido a sus propiedades y eficiencia.

Limitación de la inversión de prioridad evitando el uso de sincronización Se han propuesto diferentes protocolos [7, 6] que evitan la inversión de prioridad evitando el uso de sincronización. El protocolo, propuesto por Anderson en 1997, *Lock Free Shared Objects* (LSFO) [7] utiliza bucles que se ejecutan de forma continua hasta que la tarea consigue acceder al recurso compartido, utilizando primitivas de

sincronización basadas en instrucciones atómicas. Este protocolo, usado en combinación con RMS, es capaz de limitar la inversión de prioridad que una tarea puede sufrir. En otro trabajo, Anderson demostró la viabilidad del uso en sistemas de tiempo real de esquemas que no obligan a esperar por un objeto (*Wait Free Objects*) [6]. Estos esquemas se basan en cómo las tareas son planificadas para la ejecución en sistemas de tiempo real, y aunque tienen características en común con protocolos que evitan la inversión de prioridad, son no bloqueantes y están implementados a nivel de usuario.

Tests de planificabilidad Para llevar a cabo el análisis de planificabilidad de un conjunto de tareas con sincronización, el instante crítico varía con respecto a los métodos vistos anteriormente, donde se suponía que las tareas no se comunicaban, es decir, no se tenía en cuenta el factor de bloqueo. Así, para analizar si una tarea es planificable, a su tiempo de computación en el peor caso, calculado anteriormente como la suma de su tiempo de ejecución en el peor caso y de la interferencia en el peor caso que sufre por parte de tareas con mayor prioridad, se le añadirá un factor de bloqueo, B_i, que dependerá del método elegido.

Para los protocolos PCP e IPCP, cada tarea τ_i compartiendo recursos, sufrirá un bloqueo en su ejecución equivalente a la duración de la sección crítica más larga, con techo de prioridad igual o mayor que el asignado a τ_i, y ejecutada por tareas de prioridad menor que τ_i:

$$B_i = \max_{\forall j \in l_p(i)} s_j^i \tag{2.24}$$

donde,

- $l_p(i)$ es el conjunto de tareas con menor prioridad que τ_i, y
- s_j^i es la duración de la sección crítica con techo de prioridad mayor o igual que la asignada a la tarea τ_i ejecutada por τ_j.

El tiempo de bloqueo para PIP requiere un cálculo más elaborado que se puede encontrar en [40].

Considerando el término de bloqueo se pueden generalizar los test de planificabilidad vistos hasta el momento. Así, para el caso de RMS, Sha [199] generalizó el test de utilización de Liu y Layland (teorema 2.3):

Teorema 2.8 (Sha et al,1990). Sea $\Pi = \{\tau_i, i = 1 \dots N\}$ un conjunto de tareas ordenadas por prioridad decreciente, utilizando asignación de prioridades RMS. El conjunto es planificable si:

$$U_i = \sum_{j=1}^{i} \frac{C_j}{T_j} + \frac{B_i}{T_i} \leq i(2^{\frac{1}{i}} - 1) \; \forall i = 1 \dots N \tag{2.25}$$

El método de Sprunt et al [204], para tareas aperiódicas y asignación de prioridades RMS, generaliza el test de Lehoczky (teorema 2.4), introduciendo un nuevo término en la inecuación 2.14:

$$\min_{t \in S_i} \left(U_i(t) + \frac{B_i}{t} \right) \leq 1 \tag{2.26}$$

donde $U_i(t)$ es la utilización en el peor caso dada por la ecuación 2.13 y S_i el conjunto los puntos de planificación donde debe comprobarse la inecuación anterior definido por la fórmula 2.10.

Audley et al [16] generalizaron el método de Joseph y Pandya, basado en el cálculo de los tiempos de respuesta en el peor caso. Este método es válido para cualquier asignación fija de prioridades de tareas, con plazos de finalización menores o iguales que sus períodos. Además, contempla el efecto de activación retrasada en tareas periódicas (*release jitter*). Así, extienden la ecuación 2.20 llegando a la expresión iterativa :

$$w_i^{n+1} = B_i + C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w_i^n + J_i}{T_j} \right\rceil C_j \tag{2.27}$$

siendo J_i el máximo retraso en la activación de la tarea τ_i.

2.1.3.4. Plazos superiores a los períodos

El caso en el que las tareas posean un plazo superior a su período requiere una especial atención. Esta situación, como veremos en el apartado 2.2, suele ser habitual en sistemas distribuidos, donde una activación en respuesta a un evento debe recorrer distintos recursos hasta su finalización. Bajo esta condición, el análisis del primer instante crítico, como el realizado en el método desarrollado por Audsley et al [16], resulta insuficiente, pues en posteriores instantes críticos una activación anterior de la tarea podría modificar su tiempo de respuesta en el peor caso. Además, en este tipo de sistemas la asignación de prioridades en función de sus períodos (RMS) o plazos (EDF), deja de ser óptima, siendo necesario el uso de heurísticos [14].

Lehoczky [140] demostró que debe extenderse el análisis del instante crítico sobre todas las activaciones de una tarea que ocurran dentro de su período de ocupación de peor caso. Siendo para una tarea τ_i, el intervalo temporal en el que el procesador está ocupado ejecutando tareas de mayor o igual prioridad que la de τ_i. Basándose en esta definición y en el método propuesto por Audsley (véase ecuación 2.27), Tindell et al [222, 221] propusieron una técnica exacta basada en la obtención de los tiempos de respuesta, independiente de la asignación de prioridades y válida para cualquier relación período-plazo de las tareas:

$$w_i^{n+1}(p) = B_i + (p+1)C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w_i^n(p) + J_i}{T_j} \right\rceil C_j \tag{2.28}$$

Esta ecuación se resuelve de la misma manera que el método iterativo anterior, iniciando cada iteración sobre p con el valor $w_i^{(0)}(p) = (p+1)C_i$, y deteniendo su análisis cuando $w_i(p) \leq (p+1)T_i$.

El tiempo de respuesta en el peor caso de la tarea τ_i, será el máximo de los tiempos de respuesta obtenidos:

$$R_i = \max_{\forall p} \{w_i(p) + J_i - qT_i\} \tag{2.29}$$

El conjunto de tareas será planificable si todos los tiempos de respuesta en el peor caso son inferiores a sus plazos respectivos.

2.1.3.5. Tratamiento de eventos aperiódicos

Las tareas aperiódicas no son contempladas en la mayoría de los métodos comentados hasta el momento. Dado que de una tarea aperiódica no sabemos su instante de llegada al sistema, la aproximación seguida al planificarlas ha de ser distinta a la seguida hasta el momento con sistemas sólo compuestos por tareas periódicas. El problema radica en atender a estas tareas aperiódicas para que cumplan sus restricciones temporales en caso de que existan, sin interferir en el cumplimiento de los plazos de respuesta de las tareas periódicas. Si no tienen restricciones temporales, es deseable que sus tiempos de respuesta sean lo más reducidos posible.

Existen distintas aproximaciones al tratamiento de estas tareas [172], como los servidores de muestreo periódicos (en los que una tarea periódica de prioridad muy alta chequea la llegada de nuevos eventos y los procesa), la asignación de una prioridad muy alta a los eventos que llegan al sistema (procesando los eventos en cuanto llegan), o muy baja (asegurando así el cumplimiento de los plazos a las tareas con requisitos temporales estrictos), o los protocolos de reserva de ancho de banda, que reservan cierta capacidad de ejecución para las tareas aperiódicas. Entre estos últimos destaca el Servidor Esporádico [204], que se basa en la creación de tareas servidoras a las que se les asigna un tiempo de computación máximo para tratar eventos aperiódicos, y en definir un mecanismo de restauración del tiempo de cómputo consumido. Por otra parte, en este método, si el tiempo de cómputo asignado se agota, la tarea puede continuar ejecutándose a un nivel de prioridad bajo. Una propiedad interesante del Servidor Esporádico es que, desde el punto de vista de análisis de planificabilidad, se comporta como si fuera una tarea periódica, tal y como pone de manifiesto el siguiente teorema:

Teorema 2.9 (Sprunt et al). Un conjunto de tareas periódicas que es planificable con una tarea τ_i, sigue siendo planificable si se la substituye por un servidor esporádico con el mismo período y tiempo de cómputo.

En [200] se puede encontrar una perspectiva histórica de las distintas aproximaciones utilizadas en planificación de tiempo real.

2.1.4. Cambios de modo en sistemas de tiempo real

Tradicionalmente, se asumía que un sistema de tiempo real estaba formado por un conjunto fijo de tareas que eran planificadas para satisfacer los requisitos de las aplicaciones y exhibir el comportamiento deseado. Las aplicaciones que necesitan manifestar más de un comportamiento son denominadas aplicaciones multimodo [189]. Estas aplicaciones consisten en un número prefijado de modos de operación, cada uno de los cuales produce un comportamiento diferente, implementados por diferentes conjuntos de tareas. Como el conjunto de modos es establecido *off-line*, es decir, previamente a la ejecución del sistema, la flexibilidad que proporciona en tiempo de ejecución vista en conjunto es limitada, pese al dinamismo introducido permitiendo cambios de modo en tiempo de ejecución.

2.2. Sistemas de tiempo real distribuidos

En su origen, las aplicaciones informáticas eran centralizadas, desarrollándose en sistemas monoprocesador que realizaban todas las operaciones del sistema. A medida que la tecnología fue avanzando, fue posible la integración de múltiples procesadores, cada uno ellos con distinta funcionalidad dentro del sistema. Así, la arquitectura clásica centralizada ha dado paso a sistemas distribuidos, que pueden implementar funcionalidades más complejas y que, *a priori*, aportan más fiabilidad, reduciendo el riesgo de error debido a la caída de uno de los procesadores.

Según Coulouris et al [60], un sistema distribuido es[1]:

> Una colección de ordenadores autónomos conectados por una red de computación y equipados con *software* distribuido. El *software* distribuido permite a los computadores coordinar sus actividades y compartir los recursos del sistema - *hardware*, *software* y datos. Los usuarios de un sistema distribuido bien diseñado deben percibir una facilidad de computación simple e integrada aunque pueda estar implementada por muchos computadores en diferentes localizaciones.

Así, podríamos definir un sistema distribuido como un conjunto de entidades autónomas que cooperan entre sí para alcanzar un fin común. Para esto, deben intercambiar información mediante el uso de mensajes, necesitando un sistema de comunicaciones.

Un sistema de tiempo real distribuido será, entonces, un sistema distribuido con restricciones temporales, es decir, una sistema distribuido donde existen actividades con requisitos de tiempo real. Para cumplir los requisitos temporales, estas actividades además de necesitar ejecutar sus tareas en una unidad de procesamiento, pueden, ocasionalmente, necesitar intercambiar datos con otras tareas ubicadas en otras unidades. Para que el sistema completo cumpla sus restricciones temporales, tendrá que

[1] Traducido de la versión original en inglés por la autora

ser capaz tanto de realizar el procesamiento de dichas tareas en cada procesador, como de intercambiar datos entre las distintas entidades, todo de manera determinista, ajustándose a las cotas temporales impuestas por los requisitos temporales de cada una de las actividades de tiempo real existentes en el sistema [173, 221]. Así, la planificación de sistemas distribuidos presenta frente a la de sistemas centralizados el problema de adicional de tener que realizar la planificación de los canales de comunicaciones y la asignación de tareas a los procesadores.

En general, un sistema de tiempo real distribuido está compuesto por uno o varios procesadores, conectados entre sí a través de un bus o de una red de comunicación, que interaccionan con el entorno. Dicha interacción se realiza mediante sensores que informan del estado del sistema y de actuadores que modifican alguna característica del mismo. Por ejemplo, en un sistema de frenado ABS, existen sensores ubicados en las ruedas que controlan permanentemente la velocidad de giro de las mismas. A partir de los datos que suministra cada uno de los sensores, la unidad de control calcula la velocidad media. Si se compara la velocidad de una rueda concreta con la velocidad media de las cuatro ruedas, se puede detectar cuándo una rueda está a punto de bloquearse. En este caso, el sistema, por medio de actuadores, reduce la presión de frenado en dicha rueda hasta que alcance su velocidad un umbral fijado por debajo del límite de bloqueo. Éste es un sistema distribuido compuesto por la unidad de control y los sensores y actuadores, en el se pueden intuir las restricciones temporales: la eficacia del ABS está directamente relacionada con la diferencia temporal entre el momento en el cual detecta el problema y en el que actúa, no pudiendo ser éste arbitrario, sino que ha de ser acotado y determinista.

Por otra parte, existen dos enfoques principales para el desarrollo de sistemas distribuidos [131], que originan sendos paradigmas que analizaremos en el apartado 2.2.2.2:

- Sistemas gobernados por eventos: las acciones son activadas por la ocurrencia de eventos, que caracterizan el flujo de control del programa.
- Sistemas gobernados por tiempo: las acciones son activadas por el sistema en instantes de tiempo predeterminados, habitualmente de forma cíclica a intervalos de tiempo regulares. La acción del reloj es así el único evento que activa las operaciones en el sistema.

La adquisición de información a través de los sensores, puede ser realizada a través de distintas técnicas, que se pueden clasificar en las siguientes categorías[230]:

- Muestreo: se observa el valor de la entidad de forma periódica. Se suele usar este método en el caso de la observación de entidades continuas, las cuales exhiben curvas de variación más o menos predecibles, debido a que un cambio brusco puede ocasionar una pérdida de información. La frecuencia de observación es determinada por reglas de control bien conocidas. Un ejemplo típico es el muestreo de señales analógicas, como el audio, en la cual la frecuencia

de muestreo ha de ser al menos dos veces la frecuencia de Nyquist, definida como la frecuencia máxima de todos los armónicos de la forma de ondas que se observa.

- *Polling*: se realiza una observación de la variable en un bucle infinito, normalmente durante un período corto durante el que se espera un cambio en la entidad observada. Este método es análogo a la espera activa en programación, obligando al procesador a una atención en exclusiva durante el período de observación. Por esta razón, se suelen usar sensores que poseen microcontroladores dedicados (también llamados *sensores inteligentes*).

- *Latching*: los sensores sólo informan de cambios significativos en la entidad observada. Este método es apropiado para entidades discontinuas, como, por ejemplo, valores digitales. Un *latch* es un elemento de memoria que puede registrar uno o más cambios de estado, típicamente un biestable. Suelen estar combinados con otras técnicas como muestreo o interrupción.

- Interrupción: el sensor genera asíncronamente un requerimiento de interrupción al sistema de control, de manera que éste invoca a un manejador de interrupciones con el código oportuno para el procesado de la observación. Se suele utilizar con entidades esporádicas, en las que los cambios en su estado se producen en momentos aleatorios.

Por otra parte, en un sistema distribuido se deben caracterizar los eventos que disparan la ejecución de las acciones. Por ejemplo, la adquisición de información por parte de un sensor puede originar un evento en el sistema que se encargue de procesarlo. Una clasificación comúnmente aceptada es la basada en la fuente que origina dicho evento [172]:

- Eventos temporizados: producidos por temporizadores o relojes propios del sistema.

- Eventos externos: originados en el sistema físico que se está controlando y que desencadenan una serie de acciones en el sistema como respuesta a ese evento.

- Eventos internos: producidos dentro del sistema, necesarios para la correcta sincronización del mismo.

2.2.1. Planificación distribuida

Como se ha visto en apartados anteriores, el trabajo realizado en el análisis y planificación en sistemas de tiempo real en sistemas monoprocesador es bastante amplio, y han sido estudiados numerosos casos, aportando muchos resultados tanto a nivel teórico como práctico. Sin embargo, tanto el análisis como la planificación

de tiempo real en sistemas multiprocesador y distribuidos es aún un campo abierto a la investigación, en lo que se refiere a la búsqueda de mejores soluciones y más generales, que se puedan aplicar a un número mayor de sistemas de este tipo.

Esto se debe, en parte, a la complejidad computacional de este problema. Se ha demostrado [81, 209] que muchos problemas de planificación de tiempo real en multiprocesadores son NP-completos. En un estudio, realizado por Stankovic *et al* [209], se encontraron asociados a la planificación multiprocesador características contradictorias con las halladas en sistemas monoprocesador: el uso de apropiación de procesador no siempre es beneficioso; el uso con dos o más procesadores de cualquier algoritmo de planificación dinámica basada en plazos (como, por ejemplo, EDF), no puede ser óptimo sin un conocimiento completo del sistema; y, por último, el hecho de aumentar el número procesadores, o de relajar los requisitos puede provocar que un conjunto de tareas planificable deje de ser planificable. Todos estas características conducen a que se hayan usados heurísticos para la planificación de tiempo real distribuido, tales como el templado simulado de Tindell [224], el algoritmo *HOPA* [94] (*Heuristic Optimized Priority Assignment*), o los algoritmos *WCDO* [173] (*Worst Case Analysis for Dynamic Offsets*) y *WCDOPS* [174](*Worst Case Analysis for Dynamic Offsets with Priority Schemes*).

Existen varias estrategias para abordar la planificación distribuida de tiempo real. Una de ellas es abordarla en dos niveles [3]: en alto nivel, un planificador distribuido determina en qué nodo ha de ser procesada una tarea recién llegada al sistema. A bajo nivel, un planificador local dinámico determina la planificabilidad local de cualquier tarea recién asignada. Si la planificabilidad del conjunto de tareas totales del sistema está asegurada, la tarea se intenta planificar localmente. Si no se consigue, el planificador local se coordina con el planificador distribuido para la migración de la tarea a otro nodo. Esta estrategia es la seguida, por ejemplo, en el kernel *Spring* [187]. Litoiu et al [145] proponen una estrategia similar pero usando reglas de lógica borrosa (*fuzzy logic*) en el algoritmo de decisión para enviar tareas con restricciones temporales no garantizadas a procesadores con suficiente capacidad. Kao y García-Molina [128] proponen asimismo un sistema de asignación de plazos en un sistema de tiempo real distribuido flexible, siguiendo la misma aproximación.

2.2.1.1. Modelos y políticas de planificación para el análisis de sistemas distribuidos

Existen distintos modelos sobre los que basarse para realizar la planificación de un sistema distribuido. En este apartado se presentan los que guardan más relación con el modelo utilizado en esta tesis. En los modelos presentados, la red se modela como un recurso más sobre el que debe ejecutarse una acción, en este caso la transmisión de un mensaje. Esto se debe, como veremos más adelante, a la similitud entre el cálculo de tiempos de respuesta en el peor caso de la ejecución de tareas propiamente dichas (aquellas que se ejecutan en un procesador) y de las tareas de

transmisión de mensajes sobre una red [172][2].

Modelo transaccional de tiempo real Este modelo computacional, definido por Tindell en [223], se basa en la definición de transacciones de tiempo real. Una transacción, véase figura 2.1, es una entidad activada por un evento externo periódico, de período T_i, que agrupa tareas con período idéntico al evento externo, cuyos instantes de activación están relacionados.

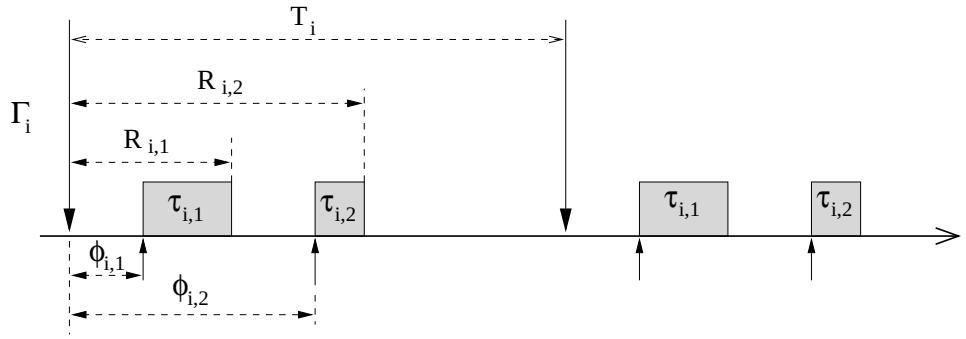


Figura 2.1: Ejemplo de transacción de tiempo real

Se define una transacción como un conjunto de tareas ordenado según sus instantes de activación:

$$\Gamma_i = \{\tau_{i,j} = (C_{i,j}, D_{i,j}, T_i, \phi_{i,j}, J_{i,j}, p_{i,j}), j = 1 \dots N\} \tag{2.30}$$

Cada ocurrencia del evento externo generará una instancia distinta de la transacción, y cada una de las tareas que la componen se activará transcurrido un intervalo de tiempo (desplazamiento temporal u *offset*), $\phi_{i,j}$, hayan o no concluido todas las tareas con menor desplazamiento temporal pertenecientes a su misma transacción. Si $\phi_{i,j}$ se corresponde con un valor numérico fijo, la transacción será con *offsets* estáticos. Si puede variar en el intervalo $\left[\phi_{i,j}^{min}\ ,\ \phi_{i,j}^{max}\right]$, será una transacción con *offsets* dinámicos [172].

El modelo permite el retraso en la activación de tareas, a través de la definición para cada una de ellas de un retraso máximo,$J_{i,j}$,de tal forma que la activación de cada tarea se podrá producir en cualquier instante en el intervalo $[t_0 + \phi_{i,j} - J_{i,j}\ ,\ t_0 + \phi_{i,j} + J_{i,j}]$, donde t_0 es el instante de llegada del evento externo. Para cada tarea se define su tiempo de respuesta como la diferencia entre t_0 y el instante de su finalización. Denotando $R_{i,j}$ a su tiempo de respuesta en el peor caso.

Diversas políticas de planificación utilizan este modelo para planificar tareas en un sistema de tiempo real distribuido. Entre ellas destacan el algoritmo de *Modifica-*

[2]A partir de aquí utilizaremos los términos tarea o acción para referirnos indistintamente a tareas y a mensajes.

ción de Fase [33], el análisis holístico de Tindell [225, 223] y el protocolo *Release Guard Protocol* [211].

El algoritmo de *Modificación de fase* tiene un especial interés debido a que reduce el efecto de las activaciones retrasadas. En este algoritmo, la activación de una tarea en respuesta a un evento se realiza un tiempo fijo después de la llegada de dicho evento. Dicho instante de activación se calcula teniendo en cuenta el tiempo de respuesta en el peor caso de las tareas precedentes, asegurando que la activación de una tarea se produce siempre después de que hayan terminado todas las que le preceden en la transacción.

El anterior algoritmo conlleva la sincronización entre los distintos procesadores que están involucrados en una transacción. Nótese que este problema desaparece en sistemas gobernados por paso de mensajes. Tindell propuso el uso de paso de mensajes indicando la ocurrencia del evento, considerando en cada tarea un retraso adicional debido al tiempo de transmisión de dichos mensajes. Este modelo es también el utilizado en parte de los algoritmos propuestos por Palencia en su tesis doctoral [172] [3], en la que propone mejoras sobre el análisis de planificabilidad en sistemas distribuidos con *offsets* estáticos y dinámicos, realizando la inclusión en el análisis del tiempo de respuesta en el mejor caso de las tareas precedentes.

La técnica de planificación propuesta por Sun y Liu, *Release Guard Protocol*, evita también el problema de la sincronización de relojes. Esta técnica es similar a la desarrollada por Tindell, pero aplicando el algoritmo de *Modificación de fase*. Cuando una tarea finaliza su ejecución, se duerme hasta un instante prefijado relativo a su instante de activación, en el cual esta tarea crea otra tarea agente de alta prioridad que envía un mensaje de activación a la tarea siguiente. Esta técnica se restringe a plazos y desplazamientos temporales menores que los períodos y no tiene en cuenta el efecto de las activaciones retrasadas, condiciones sí contempladas tanto en el trabajo de Tindell como en el de Palencia.

Gutiérrez propuso el uso de un *servidor esporádico de comunicaciones para comunicaciones* [94] que consigue el mismo efecto que el algoritmo anterior sin la dependencia del diseño del *software* con las restricciones temporales del sistema presente en la técnica anterior, manteniendo la independencia entre la planificación y el *offset* de las tareas[4].

Modelo lineal gobernado por eventos Este modelo para sistemas distribuidos gobernados por eventos, definido por Gutiérrez en su tesis [95][5], se muestra en la figura 2.2.

[3]Más concretamente en los capítulos 4, *Análisis de planificabilidad para tareas con offsets estáticos y dinámicos*, y 5, *Mejoras al análisis de tareas con offsets y relaciones de precedencia*.

[4]Para más información sobre este algoritmo, se recomienda la lectura del capítulo 6, *Servidor esporádico de comunicaciones*, de la tesis [95].

[5]La definición del modelo se encuentra en el capítulo 2, *Análisis de tiempo real para sistemas distribuidos*, de dicha tesis

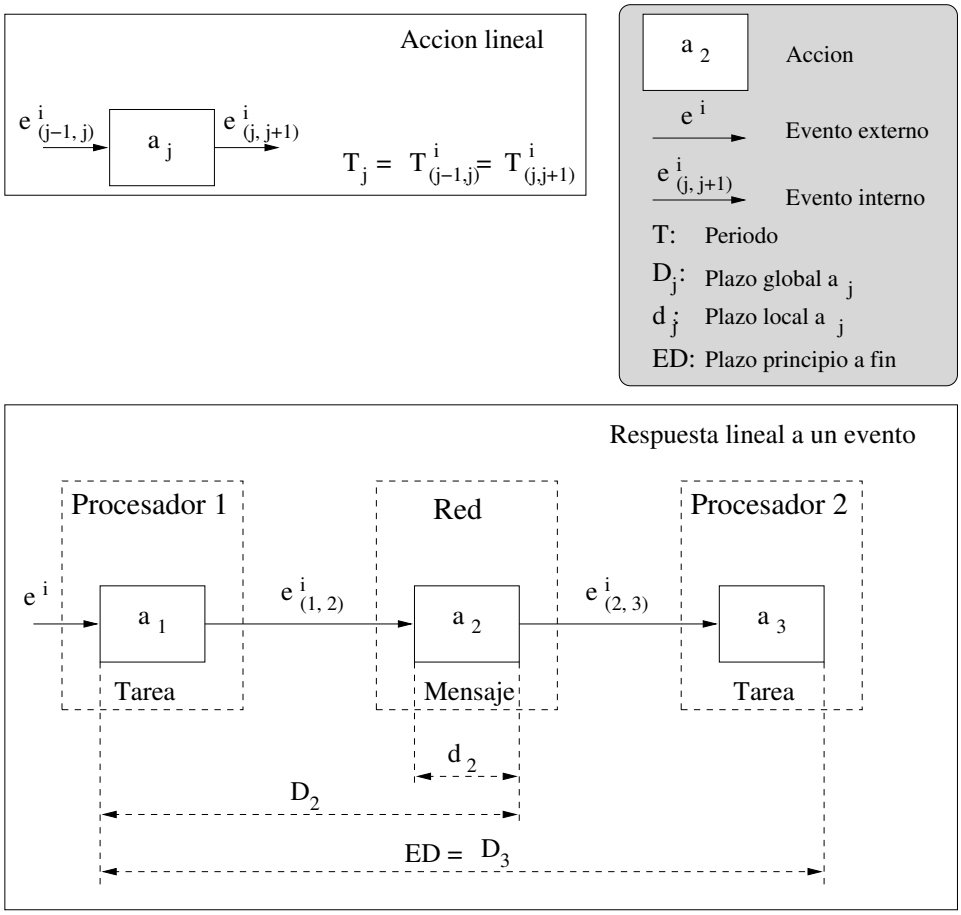


Figura 2.2: Modelo lineal gobernado por eventos

El modelo contempla dos tipos distintos de eventos: *externos* o *internos*. Los eventos externos son aquellos que llegan al sistema y que generan una respuesta en forma de secuencia de acciones, que se activan entre sí mediante la generación de eventos internos. Los eventos internos pueden ser generados por tareas o por mensajes. Las tareas los generarán para activar a otras dentro del mismo procesador o para transmitir un mensaje, y será la recepción de un mensaje el que activará a la tarea que los reciba. Las características temporales (por ejemplo, periodicidad) de los eventos internos de una secuencia se derivan del evento externo que la activa.

En este modelo, las acciones sólo pueden ser activadas por un único evento (interno o externo, si se trata de la primera acción de la secuencia) y sólo generan un único evento interno. Cada acción a_j tendrá un plazo global relativo a la llegada del evento e_i, D_j, un plazo local relativo a la activación de a_j, d_j, y un plazo de principio a fin para la secuencia, ED, igual al plazo global de la última acción de la secuencia.

Para facilitar el análisis de planificabilidad, tanto las secuencias de acciones como los períodos de los eventos externos se conocen por adelantado. Además, este

modelo supone que las tareas se asignan estáticamente a los procesadores al igual que los mensajes a las redes de comunicaciones.

Pese a que el modelo permite representar un gran número de arquitecturas, presenta algunas limitaciones: no permite la sincronización entre tareas ni la activación por combinaciones de eventos. Estas limitaciones son superadas por el modelo general desarrollado también por Gutiérrez en [95] [6].

La diferencia fundamental entre el modelo transaccional y el lineal radica en que en una secuencia de acciones, cada acción no se activa hasta la llegada de un evento interno generado por la acción anterior, es decir, no se activa hasta que hayan terminado todas las acciones previas. Hay una relación de precedencia explícita. Mientras que en una transacción, las relaciones de precedencia no son explícitas: cada tarea se activa a partir de un instante definido por su desplazamiento temporal.

Este modelo ha sido utilizado por Palencia en sus mejoras al análisis de planificabilidad propuesto por Tindell [7].

2.2.1.2. Análisis de planificabilidad

En el análisis de planificabilidad de un sistema distribuido se complica con respecto al de un sistema monoprocesador debido a que han de tomarse en consideración no sólo las tareas en cada procesador, sino también el tiempo de transmisión en la red y las relaciones de precedencia entre las tareas. Además, como ya se comentó, se deben emplear heurísticos para la asignación de prioridades a las acciones pertenecientes a una transacción o secuencia, pues no existe, como en los sistemas monoprocesador, una asignación óptima basada en plazos o períodos.

Análisis de la red de comunicaciones Para poder realizar un análisis de planificabilidad de una transacción o de una secuencia de acciones, debemos conocer no sólo el tiempo de ejecución en el peor caso de las tareas involucradas, sino también el tiempo de respuesta en el peor caso para la transmisión de cada mensaje.

El análisis de planificabilidad empleado habitualmente supone que los mensajes se envían fragmentados en paquetes de tamaño fijo, y que la red de comunicaciones planifica la transmisión de los mensajes mediante prioridades asignadas a los mismos. La red de comunicaciones es sin expulsión, por lo que deben tenerse en cuenta los tiempos de bloqueo debidos a mensajes de menor prioridad. Por otra parte, los tiempos debidos a la generación y fragmentación de los mensajes, y los debidos a la reconstrucción de los mismos, pueden asignarse o bien a las tareas productoras y consumidoras del mensaje, o bien a la transmisión del mismo [172].

[6] La extensión a un modelo más general se encuentra en el capítulo 4, *Extensión del análisis de tiempo real y de la planificación de sistemas distribuidos*, de [95].

[7] Véase capítulo 3, *Mejoras sobre el análisis*, de [172].

Para plazos menores que los períodos se obtiene la expresión:

$$R_m = C_m + B_m + \sum_{j \in hp(m)} \left\lceil \frac{R_m}{T_j} \right\rceil C_j \tag{2.31}$$

donde,

- C_m es el tiempo de transmisión en el peor caso del mensaje.

Se puede ampliar este análisis para considerar plazos mayores que los períodos y retrasos en la generación del mensaje:

$$w_m^{n+1}(p) = B_m + (p+1)C_m + \sum_{j \in h_p(m)} \left\lceil \frac{w_m^n(p) + J_m}{T_j} \right\rceil C_j \tag{2.32}$$

Obteniendo el tiempo de respuesta en el peor caso para la transmisión del mensaje:

$$R_m = \max_{\forall p} \{w_m(p) + J_m - qT_m\} \tag{2.33}$$

Efecto del retraso en un sistema distribuido En el análisis de planificabilidad de un sistema distribuido no sólo ha de tenerse en cuenta la red, sino también la precedencia en la activación de las tareas y mensajes de una secuencia o transacción.

Como ya se comentó, en el modelo propuesto por Tindell [221], el instante de activación de una tarea a_j será el instante en el que finalice la ejecución de la tarea precedente. Aunque la primera acción de la transacción se active con retraso nulo, $J_1 = 0$, las siguientes pueden experimentar un retraso máximo equivalente a la máxima variación del tiempo de respuesta de la acción precedente:

$$J_j = R_{j-1} - R_{j-1}^{best} \tag{2.34}$$

donde,

- R_{j-1} es el tiempo de respuesta global en el peor caso de la acción a_{j-1}. Si $j = 0$, es decir, si a_j es la primera acción de la transacción, se considerará $R_{j-1} = 0$
- R_{j-1}^{best} es el tiempo de respuesta global en el mejor caso de la acción a_{j-1}. Tradicionalmente, se considera este tiempo de respuesta arbitrariamente pequeño ($R_{j-1}^{best} \simeq 0$),

En los algoritmos de análisis de planificabilidad propuestos por Palencia [172, 173, 174], se incluye el tiempo de respuesta en el mejor caso, R_{j-1}^{best}, en la estimación de los tiempos de respuesta de las tareas, siendo ésta una de las razones, junto con la inclusión en el análisis de las relaciones de precedencia entre tareas, por las que esta aproximación es menos pesimista que la propuesta por Tindell y Clark [221], que se presenta en el siguiente apartado.

Algoritmo 2.1 Algoritmo iterativo para la obtención de los tiempos de respuesta de Tindell

```
Tiempos_Respuesta_Tindell(Γ)
{
  foreach τ_i in Γ
    let J_i^0 = 0
    let R_i = ∞
  end foreach
  let n = 0
  loop
    foreach τ_i in Γ
      if R_i = ∞ then (R_i^{n+1}, No_Planificable) := Calculo_R(J_i^n)
    foreach τ_i in Γ
      if R_i = ∞ then J_i^{n+1} := Calculo_J(R_i^{n+1})
    foreach τ_i in Γ
      if R_i^n = R_i^{n+1} then R_i = R_i^n
    if No_Planificable or ∄R_i = ∞ then
      return R = {R_i}
    let n = n + 1
  end loop
}
```

Planificación holística bajo la aproximación de tareas independientes En este apartado se presenta la técnica desarrollada por Tindell y Clark [221] para sistemas distribuidos. Esta aproximación no tiene en cuenta las relaciones de precedencia de las tareas, suponiendo que son independientes entre sí. Dada una transacción, realiza la estimación de los retrasos máximos en la activación de cada acción, y, para cada acción, calcula su tiempo de respuesta en el peor caso aplicando la metodología propuesta por Tindell para sistemas monoprocesador [222] (ver apartado 2.1.3.4), sólo en el recurso (red o procesador) donde se ejecuta la acción, redefiniendo el conjunto de tareas $hp(i)$. Como suponen que las tareas son independientes, la estimación de los tiempos de respuesta de las tareas es más pesimista que si tuvieran en cuenta las relaciones de precedencia.

El instante de finalización de la activación p-ésima de una acción a_i se define exactamente igual que para sistemas monoprocesador (véase ecuación 2.28)

$$w_i^{n+1}(p) = B_i + (p+1)C_i + \sum_{a_j \in h_p(a_i)} \left\lceil \frac{w_i^n(p) + J_i}{T_j} \right\rceil C_j \tag{2.35}$$

donde,

- $h_p(a_i)$ es el conjunto de acciones con mayor o igual prioridad que a_i que se

encuentran alojadas en el mismo recurso que ésta.

- B_i es el término de bloqueo de la acción a_i debido a la sincronización de recursos compartidos.

Para cada p, se resuelve de la misma manera que para el caso monoprocesador, es decir, iniciando cada iteración sobre p con el valor $w_i^{(0)}(p) = (p+1)C_i$, y detienendo su análisis cuando $w_i(p) \leq (p+1)T_i$. El tiempo de respuesta en el peor caso de la tarea τ_i, será el máximo de los tiempos de respuesta obtenidos:

$$R_i = \max_{\forall p} \{w_i(p) + R_{i-1} - qT_i\} \tag{2.36}$$

El sistema distribuido será planificable si todas las acciones tienen un tiempo de respuesta inferior a su plazo.

Como los tiempos de respuesta de cada tarea dependen no sólo de las tareas en su procesador sino también de las tareas precedentes en la transacción que pueden ser dependientes a su vez de las tareas en ese procesador, es necesario el uso de un método iterativo para el cálculo de los tiempos de respuesta en el peor caso. Tindell y Clark desarrollaron un método iterativo para el cálculo de los tiempos de respuesta que se muestra en el algoritmo 2.1.

2.2.2. Comunicación de tiempo real

En un sistema distribuido de tiempo real es necesario para su correcto funcionamiento no sólo que las tareas que se ejecutan en cada procesador se ejecuten dentro de su plazo, sino también que la información intercambiada entre los dispositivos involucrados cumpla con unos requisitos temporales previamente especificados. En este apartado se presentan cuestiones relacionadas íntimamente con la comunicación de tiempo real.

2.2.2.1. Propiedades de composición, escalabilidad y tolerancia a fallos

En sistemas grandes, como pueden ser los encontrados en una planta industrial o en la fabricación de un coche, el diseño y desarrollo se realiza mediante la división en distintos subsistemas bien especificados. Cada uno de los cuales se prueba individualmente, y, posteriormente, se realiza su integración en el sistema completo. Es importante, entonces, que las propiedades que cada uno de los subsistemas cumplen por separado, tanto temporales como funcionales, se mantengan en el momento de la integración. Denominándose a esta propiedad de la arquitectura, *propiedad de composición*.

La definición de *propiedad de composición* la enunció Kopetz en [131][8]:

[8]Traducido de la versión original en inglés por la autora

> Una arquitectura se dice que mantiene la *propiedad de composición* respecto a una propiedad determinada, si la integración del sistema no invalida esta propiedad una vez que la propiedad ha sido establecida a nivel de subsistema.

Por ejemplo, en un coche, existen múltiples subsistemas que controlan los frenos (ABS, ESP, etc.). Si la arquitectura del coche mantiene la *propiedad de composición*, se garantiza que los subsistemas validados, una vez integrados, seguirán funcionando, sin tener la necesidad de volver a diseñarlos o probarlos. De aquí, que el uso de arquitecturas que posean dicha propiedad sea una manera eficiente de construir sistemas grandes.

En el mismo texto, Kopetz [131] realiza una comparativa entre sistemas de comunicación de tiempo real gobernados por eventos y por tiempo, llegando a la conclusión de que mientras los sistemas gobernados por tiempo mantienen la *propiedad de composición* respecto a los requisitos temporales de la comunicación, los sistemas gobernados por eventos, no mantienen la *propiedad de composición* respecto a su comportamiento temporal.

Mientras la *propiedad de composición* se refiere a la integración de subsistemas para la creación de un sistema mayor, la *propiedad de escalabilidad* se refiere a la posibilidad de crecimiento y evolución del propio sistema. Estos cambios pueden referirse no sólo al tamaño sino también a la propia funcionalidad, siendo necesarios para que el sistema pueda seguir siendo usado durante mucho tiempo.

Un sistema escalable ha de ser *extensible*, es decir, debe poder crecer sin una cota predeterminada. Para conseguir dicha propiedad, el sistema no puede poseer ningún cuello de botella central [131] ni a nivel de capacidad de procesamiento, ni a nivel de capacidad de comunicación. Para aumentar la capacidad de procesamiento, deben poder añadirse nuevos nodos que aumenten dicha capacidad, y si la capacidad de comunicación dentro de un *cluster* llega a su máximo, uno de los nodos que lo conforman debe poder convertirse en una pasarela que permita la incorporación de un nuevo *cluster*. Además, en un sistema escalable se debe controlar la *complejidad* del sistema, entendiendo por complejidad el esfuerzo requerido para comprender el funcionamiento del mismo. Para reducir esta complejidad, se ha de dividir el sistema en subsistemas, cada uno de los cuales estará encapsulado y ofrecerá interfaces estables y simples. Además, ha de mantenerse un férreo control sobre los patrones de interacción entre subsistemas.

Los sistemas de computadores se desarrollan y despliegan para dar servicio a un determinado conjunto de usuarios, que pueden ser humanos u otros sistemas de nivel superior. Cuando el servicio proporcionado por el sistema, desde el punto de vista de los usuarios se desvía de la especificación convenida, se dice que el sistema ha fallado [131]. La experiencia nos dicta [19] que, aunque se realice un cuidadoso diseño y se utilicen buenos componentes, ningún sistema puede sobrevivir durante un período largo de tiempo sin técnicas *hardware* y *software* que permitan que sus programas sigan ejecutándose apropiadamente a pesar de la presencia de fallos.

A la hora de realizar un diseño *tolerante a fallos* de un sistema existen dos aproximaciones diferentes [65, 131]:

- A nivel de arquitectura, transparente a la aplicación: la propia arquitectura del sistema debe proveer de técnicas de redundancia (replicación temporal o espacial), detección de errores, aislamiento y mecanismos de recuperación ante fallos. La principal ventaja de esta aproximación es que los desarrolladores de aplicación no deben preocuparse sobre la tolerancia a fallos, reduciendo la complejidad del *software* de aplicación, pero, en contrapartida, todas las aplicaciones reciben el mismo trato y se incrementan los costes adicionales de *hardware*.

- A nivel de aplicación: mediante la construcción de soluciones *ad hoc* basadas en propiedades propias de la aplicación, e integradas dentro de su código. La principal ventaja de esta aproximación es que las soluciones están especialmente diseñadas para cada aplicación en concreto, por lo que su rendimiento es mayor. Por otra parte, la complejidad del *software* de aplicación aumenta considerablemente, lo que puede llevar a la existencia de errores de programación, dificultad para mantener y actualizar el código, volviendo casi imposible la reutilización del mismo.

En la práctica, hay que llegar a un compromiso entre ambas aproximaciones, por ejemplo, el hecho de complementar la aproximación a nivel arquitectónico con mecanismos a nivel de aplicación ayuda a aumentar la fiabilidad del sistema.

En el campo de tolerancia a fallos, han sido desarrollados mecanismos de replicación activos, pasivos e híbridos [156, 230]. La programación orientada a versiones (*N-version software*) [20] es una aproximación particular capaz de tratar un amplio espectro de modelos de fallo. Esta técnica está basada en la existencia de diferentes versiones de la misma entidad *software* pertenecientes a una misma *unidad*. El resultado de una determinada operación será decidido por consenso, a través de un algoritmo de decisión. El hecho de que esta técnica se asocie normalmente a técnicas de replicación activa y mecanismos de votación la vuelven especialmente costosa.

2.2.2.2. Arquitecturas gobernadas por eventos vs. gobernadas por tiempo

En un sistema gobernado por eventos, los mensajes son enviados por la aplicación cuando ocurre un determinado evento, como un cambio en el valor de alguna entrada o la activación de una alarma. Mientras que en un sistema gobernado por tiempo, los mensajes son enviados sólo en instantes temporales predefinidos. Seguidamente se presenta una comparativa de ambos tipos de sistemas atendiendo a sus características más importantes:

- *Predictibilidad*: en general, un sistema gobernado por tiempo se construye como un ejecutivo cíclico, siendo así determinista por diseño. Su planificación es

estática y *a priori*, lo que obliga a que, en presencia de nuevos nodos, se deba recalcular la planificación (ver apartado 2.1.3.1). Dado el carácter dinámico de los sistemas gobernados por eventos, su planificación ha de ser *on-line*, muchas veces con apropiación, preparado en cualquier momento para procesar un evento de mayor prioridad de manera inmediata (ver apartado 2.1.3.5), lo que las dota de mayor flexibilidad.

- *Utilización de recursos*: si el dimensionamiento de recursos se hace atendiendo a la cantidad necesaria en el peor caso, las arquitecturas gobernadas por tiempo son más eficientes que las gobernadas por eventos, donde el peor caso es cuando todos los eventos se dan al mismo tiempo. Sin embargo, si el dimensionamiento se hace atendiendo a la media de recursos necesaria, como un sistema gobernado por tiempo siempre trabaja a todo rendimiento, es decir, siempre existe la misma carga, se encuentre en situación de alarma o no, será menos eficiente en el uso de recursos. Mientras que, en media, un sistema gobernado por eventos tiene una utilización baja de la red, permitiendo la coexistencia de otros tipos de comunicaciones con requisitos temporales bajos o nulos.

- *Propiedad de composición respecto al comportamiento temporal*: esta propiedad asegura que, cuando dos subsistemas son integrados para formar uno nuevo, el comportamiento temporal de cada uno de ellos no se verá afectado. En un sistema gobernado por eventos, en el momento de la integración de un subsistema todos los demás se verán afectados por el tráfico generado por éste. En un sistema gobernado por tiempo, las transmisiones de los mensajes se dan en momentos especificados previamente, permitiendo definir las fases de éstos de tal forma que se minimice el número de mensajes preparados para transmitirse simultáneamente.

- *Comportamiento frente a avalancha de eventos*: en un sistema gobernado por tiempo bien dimensionado no se dan sobrecargas. En un sistema gobernado por eventos, aunque en principio son más susceptibles, el uso de representantes de las entidades de tiempo real (sensores, alarmas, actuadores, etc.), que filtren la información espuria y redundante, puede neutralizar el efecto de las avalanchas de eventos.

En general, se considera que un sistema gobernado por tiempos es adecuado para sistemas de control [176], donde existen muchos mensajes periódicos que deben ser intercambiados en el sistema con una variación del retardo baja o acotada. Mientras que los sistemas gobernados por eventos se adaptan mejor a la monitorización de alarmas, ya que, aunque ocurren ocasionalmente, han de ser tratadas dentro de una latencia mínima especificada y, en el caso de un período de sobrecarga, este tipo de sistemas pueden ser diseñados para exhibir lo que se denomina degradación

suave (*graceful degradation*), por ejemplo seleccionando algunas tareas para que no cumplan sus plazos y así evitar un fallo completo del sistema [230].

Actualmente, existe una tendencia a combinar de forma eficiente tráfico basado en eventos y en tiempo, para conseguir beneficiarse de las ventajas de los dos entornos, principalmente de la flexibilidad y eficiencia de los sistemas gobernados por eventos y la predictibilidad, propiedad de composición y seguridad de los sistemas gobernados por tiempo. Se han realizado distintas aproximaciones en esta dirección, como el trabajo sobre TT-CAN (Time Triggered CAN) [114], o la definición de FlexRay [59].

2.2.2.3. Modelos de cooperación

En un sistema distribuido, como ya se dijo anteriormente, los nodos cooperan entre sí para alcanzar un fin común. Para esto deben intercambiar datos entre ellos y, en función del tipo de aplicación, la misma información se puede necesitar en distintos nodos o los nodos pueden necesitar datos de más de un nodo. Así, la comunicación puede ser uno a uno, uno a muchos, muchos a uno, o muchos a muchos. La manera en la que la información se distribuye por la red se denomina modelo de cooperación, existiendo dos modelos básicos: cliente-servidor y productor-consumidor:

- **Modelo cliente-servidor**: en este modelo, todo nodo que contiene información relevante es un servidor. Los nodos que necesitan dicha información son los clientes. La comunicación se inicia en el cliente, realizando una petición, a la que el servidor responde. Así, la comunicación es uno a uno. Un ejemplo típico son las transacciones en una base de datos de tiempo real.

  Las extensiones a este modelo, en sistemas distribuidos, pueden llevar a inconsistencias en los datos. Por ejemplo, en sistemas cliente-servidor uno a muchos, con un servidor y varios clientes, el servidor ha de secuenciar las respuestas a los distintos clientes y si, antes de terminar, los datos que envía cambian, se producen inconsistencias espaciales entre los distintos datos relativos a la entidad que guardan los clientes. Por otra parte, si un cliente necesita datos de múltiples servidores, deberá realizar secuencialmente las peticiones, pudiéndose producir inconsistencias temporales.

  Otro problema de esta aproximación, es que las peticiones llegan de forma asíncrona al servidor, y a éste le lleva algún tiempo el procesamiento de cada una, así que el tiempo que se requiere para procesar una petición particular depende adicionalmente del patrón de llegadas de peticiones que, *a priori*, es no determinista [230]. Debiéndose acotar, además, para que sea adecuado el modelo en comunicaciones de tiempo real, el tiempo de respuesta del servidor (incluyendo tanto el tiempo de ejecución del método indicado, como el tiempo de procesamiento de la petición) a cualquier petición de los clientes. Muchos sistemas de tiempo real flexibles son construidos siguiendo este paradigma, acotando las diferentes fuentes de indeterminismo.

- **Modelo productor-consumidor**: en este modelo, todo nodo que contiene información relevante es un productor, y aquéllos que necesitan dicha información son consumidores. Los datos no se discriminan por la fuente o destino de los mismos, sino por un identificador lógico que llevan asociado. El productor de información pone a disposición de los consumidores el dato, quienes atendiendo al identificador lo procesan. Los productores no necesitan saber quiénes ni cuántos son los consumidores, ni viceversa. Así, en este modelo se soporta, inherentemente, un aislamiento entre productores y consumidores. Las transacciones son iniciadas en el productor.

  Este modelo soporta *per se* comunicaciones uno a uno y uno a muchos, siendo necesario para éstas últimas, sólo que la red de comunicaciones ofrezca el envío de mensajes *broadcast* o *multicast* de forma atómica. Además, en este caso, no se producen inconsistencias espaciales en los datos, ya que la misma información llega a todos los nodos consumidores de la misma entidad. En caso de que la red no ofrezca dicha facilidad, sí pueden producirse inconsistencias espaciales. Además, en este modelo pueden producirse inconsistencias temporales si un consumidor necesita datos de múltiples productores. La existencia de múltiples productores puede provocar que en el proceso de contienda por el medio, algunos mensajes se retrasen, siendo enviados mucho más tarde cuando ya no son válidos.

- **Modelo productor-distribuidor-consumidor**: variante del modelo anterior en la que se introduce sincronización entre productores y consumidores, eliminando las inconsistencias temporales [220]. En este modelo se introduce un nuevo nodo que ejerce de coordinador, llamado *maestro*. Los nodos consumidores y productores se llaman *esclavos*. En este modelo, todas las restricciones temporales y propiedades de los mensajes son conocidas por el *maestro*, quien planifica cuándo han de enviarse cada uno de ellos, enviando dicha planificación a los productores, quienes sólo enviarán los mensajes cuando estén autorizados.

2.3. Redes de comunicaciones de tiempo real

En este apartado se realiza un estudio de un subconjunto de las redes y protocolos de comunicaciones de tiempo real. No se pretende en este apartado realizar una revisión completa, pues existen multitud de estándares de comunicaciones, sino que se busca analizar un subconjunto concreto para facilitar la elección, durante el desarrollo de la presente tesis, del más adecuado para la implementación de la arquitectura que se propondrá. Si el lector desea profundizar en esta temática se recomienda la lectura de [12], donde se presenta un análisis de las redes más habitualmente usadas en entornos de sistemas embarcados distribuidos y de las tendencias de la investigación actual en este campo.

2.3.1. CAN

El protocolo de red de controlador de área (*Controller Area Network*, CAN) [193] fue desarrollado a mediados de los años 80 por Robert Bosch GmbH, como un bus de comunicaciones de coste razonable para aplicaciones en la automoción, aunque rápidamente se extendió su uso a otros campos como la robótica y la automática [12] y se estandarizó en las normas ISO 11898-2 [116], para 1Mbps, y en la norma ISO 11519-2 [115], para 125Kbps.

CAN se basa en una arquitectura de múltiples maestros en un bus compartido *broadcast*, utilizándose habitualmente como medio de transmisión un par trenzado cuya longitud está acotada en función de la velocidad de la red. Esta restricción se debe al uso de CSMA con arbitraje de bit no destructivo (*Carrier Sense Multiple Access with Non-destructive Bitwise Arbitration*, CSMA-NBA) como protocolo MAC, que usa un sistema de prioridades fijas basado en los identificadores de mensajes. Para poder transmitir un mensaje, todos los controladores interesados transmiten el identificador de éste bit a bit. El resultado de una colisión es la operación AND entre los bits transmitidos. Así, si un controlador transmite un bit 1 y escucha en el medio un bit 0 sabe que otro más prioritario que él desea transmitir y da por finalizada la transmisión del identificador. Este sistema de arbitraje tiene dos consecuencias directas: en primer lugar, como ya se ha mencionado, limita la longitud máxima del bus en función de la velocidad y del número de nodos presentes en la red, pues el tiempo de transmisión de un bit debe ser lo suficientemente largo como para permitir su decodificación por todas las estaciones de la red (a mayor velocidad de la red, menor longitud de bus, por ejemplo, 40m a 1Mbps, 1300m a 50Kbps); por otra parte, obliga a que todos los mensajes posean un identificador único, que es el que establece su prioridad. El direccionamiento es indirecto y basado en los mensajes, no siendo definida una capa de aplicación.

Así, CAN es un protocolo de red orientado a demanda (el ancho de banda se reparte dinámicamente en función de las necesidades de cada nodo), lo que provoca que sea imposible determinar los picos de carga de la red, requiriendo la definición sobre él de protocolos y/o arquitecturas que solventen este problema para poder ofrecer garantías temporales. Aún así, su sistema de prioridades puede ser usado por protocolos de comunicaciones y aplicaciones de tiempo real, para ofrecer dichas garantías.

2.3.2. Protocolo TTP/C

El protocolo TTP/C (*Time Triggered Protocol for Class C*) [134, 229] es un protocolo de comunicaciones para arquitecturas basadas en tiempo, en concreto para la arquitectura TTA (*Time Triggered Architecture*) [133], desarrollado para dar soporte a aplicaciones críticas que requieren garantías temporales y tolerancia a fallos en el campo de la automoción. Este protocolo ofrece los servicios necesarios para implementar transmisión de mensajes predecible, comunicación con confirmación

en grupo de mensajes, sincronización de relojes, *membership*, detección de errores, cambios de modo rápidos, y administración de la redundancia.

En una red TTP/C, los nodos están agrupados en grupos (*clusters*) tolerantes a fallos (cada uno de ellos denominado FTU, *Fault Tolerance Unit*), diferenciándose en cada nodo dos capas que se comunican a través de la interfaz CNI (*Communication Network Interface*): un computador con su nivel de aplicación (*host computer*), que es el que interacciona con los sensores y actuadores; y un controlador de comunicaciones, que dispone de dos puertos bidireccionales cada uno de ellos conectado a un canal *broadcast* independiente de una red con canal dual de comunicaciones.

Esta red utiliza como protocolo MAC TDMA (*Time Division Multiple Access*), que organiza la comunicación en rondas, cada una de ellas compuesta por ranuras temporales. En cada ronda, todos los nodos deben transmitir al menos un mensaje, siendo asignada *a priori* la ranura que les corresponde. La información sobre quién debe transmitir y cuándo está almacenada en una tabla estática que poseen todos los nodos, llamada tabla MEDL (*Message Descriptor List*). Así, no es necesario que se transmita información acerca del emisor o receptor de un determinado mensaje, ya que ésta está codificada en dicha tabla. Todos los nodos detectan a la vez los errores o fallos en los demás, pudiendo ser los nodos con fallos excluidos del sistema en, como mucho, dos rondas TDMA. Por otra parte, aunque la tabla es estática, soporta cambios de modo, siendo estos preconfigurados cuando se inicia el sistema y realizándose el cambio bajo demanda dentro de cada FTU.

El uso de TDMA requiere una sincronización de reloj global. TTP/C hace uso de la información almacenada en la tabla MEDL: cada nodo mide la diferencia entre el tiempo esperado y el observado en la recepción de un mensaje correcto, para calcular la diferencia entre el reloj del emisor y del receptor del mensaje. Este dato se usa para calcular periódicamente factor de corrección del reloj local con respecto a otros miembros del grupo. Por otra parte, TTP/C define cuatro clases de velocidades de transmisión (500Kbps, 1Mbps, 2Mbps y 25Mpbs), y provee una capa de aplicación que ofrece servicios de mensajes y configuración.

2.3.3. FlexRay

El protocolo FlexRay está siendo desarrollado por el consorcio del mismo nombre, que incluye, entre otros, a BMW, DaimlerChrysler, Motorola, GM, Bosch y Philips. Su objetivo principal son las aplicaciones de control en coches. FlexRay es interesante debido a que mezcla de manera eficiente comunicación orientada a eventos y a tiempo [195], y porque es potencialmente importante debido a sus desarrolladores. Sin embargo, los detalles completos de su diseño no son públicos, siendo las únicas fuentes de las que disponemos las descripciones informales publicadas hasta el momento [59, 35], y la página web del consorcio[9].

[9] www-flexray-group.com

Este protocolo soporta tanto transferencias de datos síncronas como asíncronas a través de un acceso al bus determinista y libre de colisiones. El tiempo de bus se organiza en ciclos de comunicación de duración fija. Cada uno de los ciclos está dividido en una parte estática y una dinámica. La parte estática, donde se transmiten los mensajes síncronos, es orientada a tiempo y basada en una estrategia TDMA, lo que asegura que no existen colisiones. En la parte dinámica, el tráfico se transmite bajo demanda, usando un mecanismo de mini-ranurado (CSMA/CA) basado en el protocolo ByteFlight [180], desarrollado por BMW para su sistema de *airbag*, que también evita las colisiones.

FlexRay es flexible con respecto a la topología (tanto de bus como de estrella), soportando la opción de habilitar canales de comunicación redundantes.

2.3.4. DeviceNet

DeviceNet [167] fue desarrollado por Rockwell Automation como un estándar de bus de campo abierto basado en el protocolo CAN y diseñado específicamente para la industria de la automática. La asociación Open DeviceNet Vendor Association, Inc (ODVA) es la responsable de la especificación y mantenimiento del estándar DeviceNet. DeviceNet forma parte con ControlNet y EtherNet/IP, de una familia de estándares que usan una capa de aplicación común, denominada *Control and Information Protocol* (CIP). La parte de control maneja el intercambio de datos de entrada/salida con características de tiempo real, mientras que la parte de información define el intercambio de datos para configuración, diagnosis y gestión.

DeviceNet soporta tráfico basado en eventos y en tiempo. El tráfico periódico basado en tiempos se suele usar en un bucle de control asociando a una variable de estado un período, enviando cada vez que éste vence, los mensajes que tenga asociados dicha variable. Cuando se usa una aproximación basada en eventos, los dispositivos sólo envían cuando el valor de la variable sufre una variación desde la última transmisión superior a un determinado umbral. Si no, espera hasta que un contador periódico expira. Así, un consumidor puede detectar un fallo en un productor si no recibe datos una vez expirado dicho contador.

2.3.5. Profibus

El protocolo Profibus es un protocolo de bus de campo incluido en el estándar europeo EN 50170 [52] y en el estándar internacional IEC61158 [104], diseñado para soportar comunicaciones deterministas entre ordenadores, PLCs y dispositivos como sensores y actuadores. El mecanismo MAC utilizado está basado en un *token-passing* que garantiza a las estaciones maestras el uso exclusivo del bus (usando comunicación *broadcast*), y en un mecanismo maestro-esclavo para comunicar a las estaciones maestro con las esclavo (usando comunicación uno-a-uno). Sólo el maestro que posee el testigo puede mandar mensajes *broadcast* o iniciar una comu-

nicación con un esclavo. Los esclavos deben responder al maestro en un período de tiempo acotado. Esta característica es especialmente importante para la comunicación de tiempo real, ya que permite dar una cota superior a la duración de las transacciones, y así realizar análisis de tiempos de computación en el peor caso. Los mensajes se transmiten siguiendo un ciclo, que se inicia con una trama de acción (petición o envío/petición de trama), una trama de respuesta (asentimiento o trama respuesta) y posibles reintentos. Además, el protocolo distingue entre tráfico de alta prioridad y de baja prioridad.

El intervalo temporal que cada maestro puede poseer el testigo (*Token Holding Time*, TTH) lo calcula cada estación como la diferencia entre el tiempo que, en teoría, debería llevarle al testigo realizar una vuelta completa por el anillo y el tiempo real entre dos llegadas consecutivas del testigo a la estación. Con el objetivo de reducir el bloqueo a la que una estación puede estar sujeta cuando otras estaciones envían tráfico muy pesado, en cada visita del testigo se asegura que al menos se pueda transmitir un mensaje de prioridad alta.

Se recomienda la consulta de [227, 51] para más detalles acerca del uso de Profinet en redes de tiempo real. En estos artículos también se presentan mejoras a Profinet, consistentes en usar planificación local de mensajes basada en prioridades.

2.3.6. Ethernet

Ethernet apareció hace más de 30 años con el propósito de interconectar equipos de oficina y el protocolo fue estandarizado en el estándar IEEE 802.3, que contempla diferentes velocidades de transmisión, desde los 2.94Mbps originales [110] hasta los 10Gbps [107]. Con respecto al medio físico y a la topología, Ethernet evolucionó desde una topología en bus con un cable coaxial [106, 105] a topologías en estrella [108, 109]. Manteniéndose en todas las especificaciones el mecanismo de arbitraje, CSMA/CD, y el envío en *broadcast* de los mensajes, de tal forma que todas las estaciones los reciben.

De acuerdo con CSMA/CD, cuando una estación desea transmitir, debe esperar a que el bus esté desocupado y, una vez empieza a transmitir escuchar el medio por si se produce una colisión. En ese caso, todas las estaciones involucradas abortarán el envío y esperarán un tiempo aleatorio para volver a intentarlo. Para reducir la probabilidad de futuras colisiones, se utiliza un algoritmo binario de *back-off* exponencial. El número de reintentos se limita a dieciséis. Sin embargo, es común que cuando la carga en la red es superior a un determinado umbral, se produzcan colisiones encadenadas, por lo que este protocolo, por sí mismo, no provee de tiempos de transmisión predecibles.

Recientemente, el uso de conmutadores en lugar de concentradores, *Switched Ethernet*, permitió conseguir un tráfico con comportamiento más predecible, siempre y cuando el flujo de los mensajes en cada puerta del conmutador pueda ser caracterizado adecuadamente y no provoque un desbordamiento en las colas de cada una de las

puertas, debido a ráfagas excesivamente largas o a un exceso de paquetes *multicast* o *broadcast*.

Para superar la carencia de determinismo temporal debido al uso de CSMA/CD, se propusieron numerosas técnicas. Desde la modificación del protocolo MAC, hasta el uso de protocolos de control sobre el MAC original, como pueden ser el uso de protocolos Maestro-esclavo, paso de testigo, *Timed Token*, TDMA, *Virtual Time Protocol* [148, 155] o Traffic shaping, pasando por protocolos industriales como NDDS [175] o PowerLink [75].

En [178] se pueden encontrar más detalles acerca de las aproximaciones realizadas para dotar a Ethernet de un comportamiento temporal predecible, algunas de las cuales comentaremos en más detalle en los siguientes apartados.

2.3.7. Ethernet PowerLink

ETHERNET Powerlink [75] es un protocolo comercial que provee de comunicación isócrona determinista sobre redes Fast-Ethernet basadas en concentradores. La versión 2.0 permite operar sobre redes *Switched Ethernet* pero sólo con aplicaciones con restricciones temporales flexibles. El protocolo soporta comunicaciones periódicas (isócronas) y orientadas a eventos (asíncronas), usando una técnica de control de transmisión basada en el paradigma maestro-esclavo, la cual evita completamente las colisiones al acceder bus [32].

El maestro, llamado *Powerlink Manager*, controla todas las comunicaciones, asignando ranuras temporales al resto de las estaciones (*Powerlink Controllers*). Los esclavos sólo mandarán información después de una petición explícita del maestro. La comunicación está estructurada en ciclos, cada uno de los cuales está compuesto de cuatro fases o períodos: (1) periodo de inicialización, en el que el maestro enviará en *broadcast* un mensaje especial, indicando el inicio del ciclo; (2) periodo cíclico, en el que se produce la comunicación isócrona, gobernada completamente por el maestro mediante peticiones de mensaje *unicast* a los esclavos. Cuando un esclavo recibe una petición, responderá en *broadcast* con los datos pedidos, permitiendo así la distribución de sus datos a través de la red. Este patrón petición *unicast*-respuesta *broadcast* se repetirá para todos los mensajes isócronos que deban transmitirse en ese período. Una vez completadas todas las transacciones, el maestro transmitirá un mensaje para señalizar el fin del período cíclico; (3) periodo asíncrono, donde se producen las transacciones orientadas a evento, ya sean iniciadas por el maestro, ya iniciadas por los esclavos. Como todas las comunicaciones se realizan bajo petición expresa del maestro, si un nodo esclavo tiene datos asíncronos que transmitir, deberá notificárselo previamente al maestro. La notificación se realizará durante el periodo isócrono, indicándolo en la respuesta *broadcast*. El maestro planificará las peticiones de envío de datos asíncronos en el período asíncrono correspondiente, si hay tiempo suficiente; y (4) periodo desocupado, tiempo añadido al ciclo para respetar la periodicidad del mensaje de inicio de ciclo.

2.3.8. Protocolos FTT

El paradigma FTT [177] se basa en una arquitectura asimétrica, con un nodo maestro que controla a múltiples nodos esclavos. El maestro controla los requisitos del sistema (tanto de comunicación como de computación), las políticas de planificación y el control de admisión *on-line*. Esto implica que, dada una determinada localización de los requisitos, datos operacionales y mecanismos, esta arquitectura soporta cambios rápidos en la configuración del sistema. La posibilidad de fallo en el maestro se solventa mediante el uso de redundancia de maestros, evitando así un posible punto único de fallo. Este paradigma ha sido implementado con éxito tanto en redes de bus de campo, como CAN [4], como en Ethernet [179] o *Switched Ethernet* [149].

Type		TM Flags		Num. msgs	ID	Time	...
TM type	Master ID	Reserv.	Num Sec.	sincronos		Tx	
2 Bytes		2 Bytes		2 Bytes	2 Bytes	1 Byte	...
$[b_{15}, b_{12}]$	$[b_{11}, b_0]$	No def	$[b_7, b_0]$	$[b_{15}, b_0]$	$[b_{15}, b_0]$	$[b_7, b_0]$	...
TM_MESG_ID	$[0, 4096)$	No def	$[0, 256)$	$[0, 65536)$	$[0, 65536)$	$[0, 256)$	...

Cuadro 2.2: Estructura original del *Trigger Message* en FTT-Ethernet

La distribución de las decisiones de planificación a través del sistema la realiza el maestro periódicamente mediante un mensaje, llamado *Trigger Message* (TM), véase tabla 2.2, enviado al principio de cada ciclo elemental (EC), como se muestra en la figura 2.3. El TM es el único mensaje de control enviado cada EC y puede desencadenar la transmisión de diferentes mensajes de esclavos y ejecuciones de tareas. Los mensajes usan direccionamiento de fuente indirecto con identificadores únicos asignados a todas las fuentes. El interfaz entre tareas y mensajes se lleva a cabo a través de búferes de datos sin señales de control explícitas.

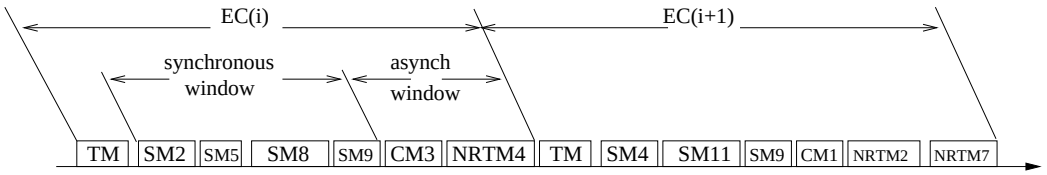


Figura 2.3: Ciclo elemental en FTT-Ethernet

Algunas características importantes en el paradigma FTT son:

- Los nodos esclavos no son conscientes de la política de planificación utilizada y el maestro puede cambiar ésta de forma autónoma o bajo demanda.

- Los nodos esclavos están sincronizados a través de la recepción del TM.

- Toda la información que se va a enviar cada ciclo elemental se comunica en el TM.
- El maestro conoce todos los requisitos temporales del sistema en cada instante de tiempo.
- Cualquier cambio a dichos requisitos está sujeto a control de admisión para filtrar aquellos que puedan poner en peligro la planificación del sistema.
- Los períodos de activación, los plazos y los desplazamientos temporales son múltiplos enteros de la duración del ciclo elemental.

2.3.9. ARINC 629

El estándar de comunicaciones ARINC 629 [10] fue desarrollado como un sucesor del estándar ARINC 429 diseñado para el Boeing 777, y provee de comunicaciones de datos de propósito general para sistemas de aviónica. El estándar define un protocolo multinivel para las comunicaciones de datos entre subsistemas, usando un bus bidireccional de múltiple acceso. Soporta tanto mensajes *broadcast* como *unicast*, los cuales pueden ser de tamaño variable. En estos momentos, la tasa de transmisión soportada es de 2Mbps, pero se está considerando soportar tasas mayores, mediante el uso de fibra óptica. El estándar especifica dos posibles protocolos de enlace: el básico y el combinado. Ambos protocolos soportan transmisiones periódicas y esporádicas. Los mensajes periódicos son planificados siguiendo una estrategia TDMA predefinida antes de la ejecución. Los mensajes aperiódicos son transmitidos bajo demanda dentro de unas ventanas temporales especificadas previamente. El arbitraje de los mensajes aperiódicos se basa en CSMA-CA, también llamado *mini-slotting* o mini-ranurado, de acuerdo al cual sólo se permite transmitir a un nodo después de detectar que el bus está vacío durante una ventana temporal cuya longitud es diferente para cada uno de los nodos involucrados.

2.3.10. AFDX / ARINC 664

La especificación ARINC 664 tiene como objetivo permitir el uso de *hardware* y protocolos comerciales en redes de datos de aviones, con la finalidad de reducir costes y tiempo de desarrollo. En particular, ARINC 664 define una red IEEE 802.3 Ethernet que usa direccionamiento IP y como protocolos de transporte TCP y UDP. Esta especificación está dividida en 8 partes, donde se definen una serie de buenas prácticas y requisitos.

La parte 7 de ARINC 664, AFDX (*Avionics Full-Duplex Switched Ethernet*) [11], define una aplicación especial para una red de datos para aviónica determinista, *full-duplex* sobre Ethernet para el intercambio de datos entre subsistemas. La falta de determinismo temporal, de garantías de ancho de banda y de calidad de

servicio inherentes al uso de Ethernet, AFDX las aborda aplicando los conceptos del estándar ATM (*Asynchronous Transfer Mode*) a Ethernet. Los principales características de AFDX son las siguientes:

- Red parametrizada: se definen los parámetros de cada estación final en tablas de configuración de los conmutadores en el momento del arranque del sistema.

- *Full duplex*: el medio físico es un par trenzado, con pares separados para transmitir y recibir.

- *Switched Ethernet*: la topología de red es en estrella. Cada conmutador conecta como máximo 24 estaciones finales y pueden ser interconectados en cascada para construir una red mayor. El conmutador reenviará los datos recibidos al destino apropiado en cada ocasión (consultando su tabla interna de reenvío), siendo del tipo *store and forward*, es decir, capacitado para almacenar tanto los datos de entrada como los de salida en colas FIFO. La posibilidad de desbordamiento se elimina mediante un apropiado modelado del tráfico y dimensionamiento de los búferes de entrada y salida.

- Determinismo: la red emula una red punto a punto determinista a través del uso de enlaces virtuales con ancho de banda garantizado.

- Redundancia: el uso de una red dual es más fiable que esquemas simples.

- Rendimiento: la red opera a 10Mbps o 100 Mbps, siendo el modo por defecto 100Mbps.

En Ethernet, las tramas son enrutadas a enlaces de salida en función de su dirección Ethernet. En AFDX, la dirección Ethernet se divide en dos campos: un campo de valor constante de 24 bits y un identificador de enlace virtual de 16 bits. Cada enlace virtual está originado por una y sólo una estación final, sin embargo su destino puede ser más de una estación final. El puerto de las estaciones finales destino a la que va dirigido el paquete está codificado en la cabecera IP. Si es un mensaje *unicast* contendrá el identificador de la estación destino, en caso de sea un mensaje *broadcast* contendrá el identificador de enlace virtual.

El aislamiento entre enlaces virtuales que comparten el mismo enlace físico se consigue limitando la tasa de transmisión y el tamaño de las tramas Ethernet de cada enlace virtual. La planificación de los enlaces virtuales de cada estación final origen se realiza internamente. Cada enlace virtual puede tener varios subenlaces pertenecientes cada uno a un puerto de la estación origen. Cada subenlace tiene asignada una cola FIFO. El planificador elegirá un enlace virtual, asegurando que todos se ajustan a su límite de ancho de banda, y los paquetes de cada subenlace serán transmitidos en *round-robin*.

2.4. Arquitecturas de Gestores de Calidad de Servicio

La calidad de servicio (*QoS*) es una noción que se emplea debido a la necesidad de caracterizar el comportamiento y el rendimiento de las aplicaciones. La mayoría del trabajo existente en el área de la calidad de servicio ha sido desarrollado para el entorno de los sistemas multimedia distribuidos [78]. Por otra parte, la gestión de recursos hace referencia a las tareas de un sistema, y a cómo se planifica la utilización de los recursos *hardware* entre las tareas que compiten por ellos. Normalmente, se realiza a nivel de sistema operativo y es implementada por las políticas de planificación de éste. Sobre la gestión de recursos de tareas, a menudo se implementan bibliotecas de especificación de necesidades de recursos para las aplicaciones, que constituyen la base para la gestión de calidad de servicio.

Las técnicas de gestión de calidad de servicio se originaron en el contexto de utilización de ancho de banda en redes y gestión de pérdidas de paquetes. Con el tiempo han sido aplicadas al dominio del procesamiento multimedia de tiempo real no crítico. En el caso más común, un sistema de calidad de servicio ofrece garantías de asignación de recursos a las aplicaciones, de tal forma que éstas tienen la certeza de que va a poder contar con los recursos contratados al inicio de su ejecución. Si se pretendiese obtener garantías totales sobre la ejecución de la aplicación, se debería hacer una estimación de los recursos necesarios en el peor caso. Sin embargo, es extremadamente difícil calcular el peor caso para una aplicación multimedia y durante la mayoría del tiempo, la aplicación funciona perfectamente con una asignación considerablemente menor, por ejemplo estimando el caso medio. Por tanto, las aplicaciones multimedia necesitan unas garantías razonables de que sus necesidades de recursos serán atendidas, pudiendo la infraestructura que les ofrece calidad de servicio relajar sus restricciones, para poder acomodar al mayor número de aplicaciones en el sistema, ofreciéndoles una calidad aceptable en el funcionamiento de las mismas.

Existen diferentes enfoques al problema de la ejecución de aplicaciones en sistemas multimedia:

- Sistema operativo: suelen centrarse en el ámbito de la gestión de recursos, de forma que incorporan políticas de planificación y asignación de recursos para tareas y/o grupos de tareas. Estos mecanismos suponen que las aplicaciones pueden ajustar su comportamiento de acuerdo a la disponibilidad de recursos, pero no ofrecen un modelo general de desarrollo de aplicaciones para un entorno semejante. Dentro de este enfoque, destacan trabajos como RT Mach [151, 138] y Rialto [123, 122, 49], donde se permite que una tarea negocie con el sistema operativo la cantidad de recursos que necesita, o SMART [159, 160], donde se permite especificar parámetros de importancia relativa de las aplicaciones del sistema para que se pueda calcular una asignación.

- *Middleware*: suelen combinar la gestión de recursos con técnicas propias de

la gestión de calidad de servicio. Por ejemplo, DQM [38] permite a las aplicaciones cooperar con el sistema operativo en su utilización de recursos del sistema.

- Arquitectura: se centra en la gestión de calidad de servicio, normalmente cuenta con un planificador de tareas a nivel de usuario, que se implementa sobre los mecanismos de gestión de recursos de un sistema operativo, sin especificar, en principio, ninguno en concreto. Generalmente, están orientadas a la gestión de calidad de servicio para entornos en red como la propuesta por NEC en [171]. Aunque otras, como AQUA [137] propone un planificador de CPU basado en el protocolo RAP [242, 137], donde las aplicaciones demandan una determinada utilización de recursos, cada uno de los cuales está gestionado por su propio planificador.

Por otra parte, existen otros trabajos como Q-RAM[184, 185], que enfocan la gestión de la calidad de servicio a través de la definición de un modelo analítico de *QoS*. Este trabajo en concreto se centra en aplicaciones distribuidas que deben satisfacer múltiples requisitos como tiempo, esquemas fiables de entrega de datos, seguridad criptográfica, calidad de datos... Además, estas aplicaciones requieren acceder a múltiples recursos, como CPU, ancho de banda o memoria, exigiendo una asignación mínima de recursos para su correcto funcionamiento, teniendo la posibilidad de mejorar sus prestaciones a través de una asignación superior de recursos, esta mejora la miden como una función de utilidad. Así, en este modelo, los recursos pueden ser asignados a aplicaciones individuales para maximizar la función de utilidad global del sistema completo.

2.5. Sistemas basados en componentes

La popularidad en los últimos años de la aplicación de la tecnología de componentes se ha debido, en gran parte, a su éxito en el mercado, basándose sobre todo en el concepto de reutilización de *software* [214]. Aplicando esta tecnología, los sistemas pueden construirse ensamblando componentes previamente desarrollados, a través de herramientas que generan automáticamente la parte no funcional de código. Así, los desarrolladores de sistemas pueden centrarse en desarrollar el núcleo funcional de sus sistemas sin perder tiempo en esas partes comunes no funcionales. Consiguiendo una disminución del tiempo invertido en los distintos ciclos de desarrollo de *software* y un gran ahorro en los costes asociados. Además, el hecho de poder, *a posteriori*, realizar versiones de los componentes utilizados para reconfigurar los sistemas desarrollados, asegura un mayor control sobre la evolución de éstos, facilitando su mantenimiento.

Además, los estándares de componentes [168, 213, 152] definen interfaces que permiten a un tercero seleccionar componentes independientes, procedentes de di-

ferentes proveedores, que podrían estar implementados en lenguajes e incluso plataformas diferentes, e interconectarlos entre sí para formar una aplicación.

En la tecnología basada en componentes, un componente ha de poseer tres características básicas [214]:

- *Aislamiento*: un componente ha de poder desarrollarse e implementarse como un elemento aislado. El componente es la unidad mínima de desarrollo, así que no puede ser desarrollado como suma de partes. Con respecto a esta propiedad no existe consenso en la literatura asociada, pues otros autores [217] consideran una composición de componentes como un componente mayor con una funcionalidad más amplia.

- *Composición*: ha de poder formar junto con otros componentes una aplicación mayor. Por lo tanto, su funcionalidad debe ser autocontenida y deberá poseer interfaces bien definidas.

- *Opacidad*: no ha de poder accederse a detalles internos ni de su implementación.

UML 2.0 [163, 162] es un lenguaje de modelado orientado a objetos de propósito general, que, al proporcionar un modelo de componentes, es especialmente adecuado para el diseño y desarrollo de aplicaciones que utilizan dicha tecnología. El mecanismo de perfiles de UML permite a las metaclases de metamodelos existentes ser extendidas con diferentes propósitos, permitiendo el uso de UML en distintas plataformas (J2EE, .NET, CORBA) o dominios (modelado de sistemas de tiempo real [164], sistemas con características de calidad de servicio [166] o procesos de negocio). Sin modificar el metamodelo, únicamente dotándolo de una nueva sintaxis para adaptarse al dominio de aplicación concreto. En el caso por ejemplo, de CORBA [168] y su modelo de componentes [170], se está trabajando en estos momentos en la definición de un perfil UML común para ambos [165].

Hoy en día existen técnicas muy diferentes para modelar sistemas de tiempo real, cada una de ellas con un conjunto diferente de terminología y notación, dependiendo del tipo de sistema que pretendan modelar. Un sistema tolerante a fallos requiere explicitar en su diseño muchas de las características de la plataforma sobre la que va a ejecutarse, pues la aplicación va a tener que manipular directamente dicha infraestructura, mientras que en otro tipo de sistemas de tiempo real podemos abstraernos de estos detalles pero nos tenemos que centrar en otros. El perfil de tiempo real de UML [164] surgió para dar un marco de trabajo común para todas estas técnicas pero permitiendo la suficiente flexibilidad para que pueda utilizarse para cualquier sistema de tiempo real. Este marco de trabajo contiene todos los elementos comunes que son usados en los distintos métodos de análisis utilizados.

Tal y como se ve en la figura 2.4, RT–UML se divide en dos grandes módulos compuestos a su vez de subperfiles. El primer módulo está dedicado a modelar los recursos del sistema, el tiempo y la concurrencia, mientras que el segundo está

compuesto por los diferentes modelos de análisis que definidos, existiendo en este momento un subperfil para el análisis de planificabilidad y otro para el análisis de rendimiento. La especificación también incluye una biblioteca de modelos que contiene un modelo UML a alto nivel de RT–CORBA [169] que pretende servir de base para los modelos que se definan utilizando este perfil. Para facilitar el análisis de aplicaciones RT–CORBA, dentro del módulo de análisis se ha definido un subperfil especial para aplicaciones RT–CORBA.

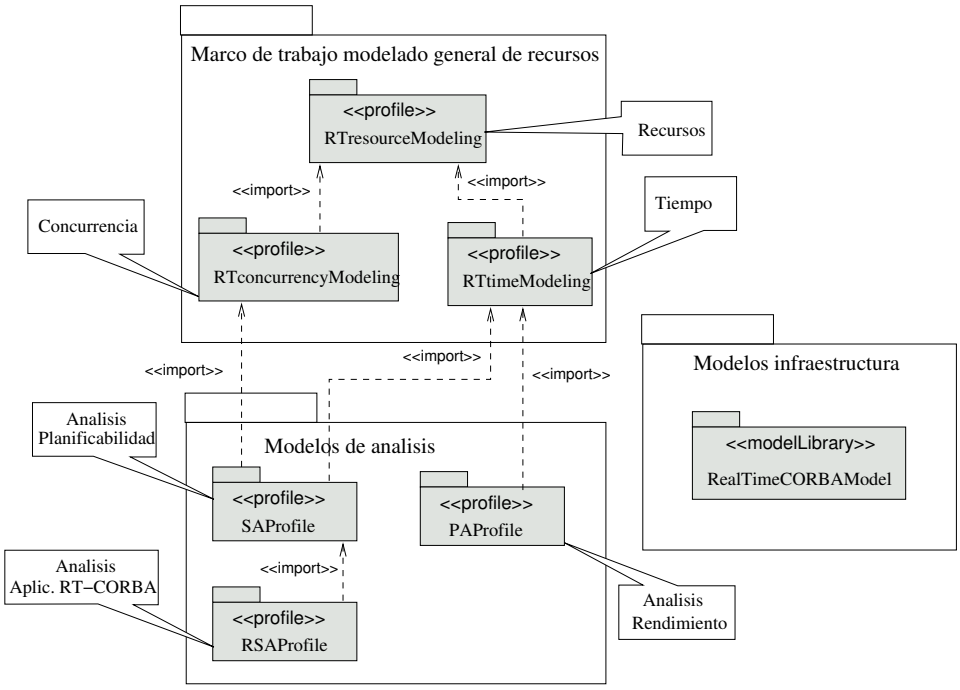


Figura 2.4: Estructura RT–UML

2.6. Paradigma de orientación a servicios

Tal y como se ha adelantado en la introducción, dentro del modelo SOA, un servicio es [124]: *"un dominio de control de una envergadura acotada que contiene un conjunto de tareas que cooperan para alcanzar objetivos relacionados"*, es decir, una entidad *software* autocontenida (sin dependencias con otros servicios, es decir, desarrollada, implementada y desplegada individualmente), con interfaces bien definidas, que proporciona una determinada funcionalidad y que, además, es sin estado. Estos servicios asumen la provisión de una determinada funcionalidad que aporta un valor añadido a la organización, entorno o herramienta y cuya especificación no depende sólo del servicio en sí, sino también de las características del entorno en el que este servicio va a ser utilizado. Por ejemplo, en el caso de un servicio que ofrece

información meteorológica, no es necesario que su interfaz defina atributos de seguridad, fiabilidad o integridad en las transacciones, que deberían estar presentes en, por ejemplo, la interfaz de un servicio de comercio electrónico.

Por otra parte, la implementación y uso de los servicios no deberán estar limitados a un determinado lenguaje o plataforma de ejecución, debiendo ofrecer, idealmente, la posibilidad de que diferentes servicios implementados en plataformas heterogéneas y/o en lenguajes distintos puedan interaccionar a través interfaces bien especificadas.

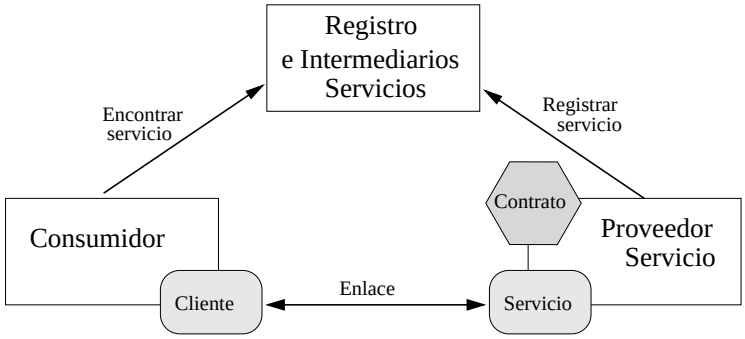


Figura 2.5: Modelo conceptual de una arquitectura orientada a servicios

En la figura 2.5 se muestra el modelo conceptual de una arquitectura orientada a servicios. Los proveedores de servicio deberán registrar éstos. Parte del acoplamiento débil entre proveedores y consumidores que requiere este paradigma se consigue a través, no sólo de la definición de interfaces bien diseñadas, sino también de la utilización de metadatos, que incluirán su funcionalidad y características más importantes, como por ejemplo su calidad de servicio. Los usuarios, consumidores, en base a los metadatos, encontrarán un determinado servicio y establecerán un contrato con el proveedor del mismo, que obligará a ambos a cumplir una serie de requisitos (como por ejemplo, calidad de servicio de entrada, de salida, número de peticiones máximas en un intervalo temporal, etc.).

La composición de servicios consiste en la combinación de las funcionalidades de servicios ya existentes para la creación de un servicio complejo. Como se puede intuir, una arquitectura que dé soporte a este paradigma ha de incluir la posibilidad de configurar sistemas y aplicaciones de forma dinámica, manejando diferentes niveles de abstracción, y dar soporte a la colaboración entre servicios. Por otra parte, aunque un servicio puede aparecer y desaparecer dinámicamente del sistema, se debe asegurar que los servicios tengan un tiempo de vida lo suficientemente largo como para poder manejar posibles excepciones e interaccionar con otros. Dicho tiempo de vida está directamente relacionado con la confianza que otras entidades tengan en ellos. En determinados entornos, este parámetro puede ser el único que dé una idea de su fiabilidad.

Un servicio compuesto se define desde el punto de vista de varias dimensiones.

En [5] se identifican seis:

- Modelo de componentes: define la naturaleza de los elementos a ser compuestos de acuerdo con las suposiciones que el modelo hace acerca de ellos.
- Modelo de orquestación: define abstracciones y lenguajes para especificar el orden en que los servicios deben ser ejecutados.
- Modelo de datos y acceso a datos: define cómo se especifican los datos utilizados en la composición y cómo son intercambiados entre los distintos componentes.
- Modelo de selección de servicio: describe cómo se seleccionan, estática o dinámicamente, los servicios a ser utilizados como componentes.
- Transacciones: define las semánticas transaccionales a aplicar en la composición.
- Gestión de excepciones: define cómo se gestionan los errores y situaciones excepcionales ocurridas durante la ejecución de la composición.

Los servicios Web [5] son una tecnología de soporte a la computación distribuida nacida a principios de esta década. Según el World Wide Web Consortium (W3C), se define un servicio Web como *un sistema software diseñado para dar soporte a interacciones de máquina a máquina interoperables sobre la red. Posee una interfaz descrita en un formato procesable por una máquina (WSDL específicamente). Otros sistemas interaccionan con el servicio Web tal y como se especifica en su descripción utilizando mensajes SOAP, que típicamente se transmiten utilizando HTTP con serialización XML y en conjunción con otros estándares relacionados con la Web* [240].

Es decir, se basan en la descripción de los servicios mediante el estándar WSDL, publicación de los mismos en repositorios UDDI y acceso a los servicios mediante el protocolo SOAP, aunque no existe consenso acerca de hasta qué punto son estas tecnologías concretas las que definen el concepto de servicio Web. Dadas sus características, resultan una tecnología *middleware* adecuada, aunque no ideal, por el momento, para la implementación de SOA.

La composición de servicios Web consiste en el desarrollo de un servicio complejo mediante combinación de la funcionalidad ofrecida por otros servicios Web. Dado que la implementación de estos servicios compuestos no es, en absoluto, trivial [5], es necesario contar con *middleware* de composición de servicios, que provea abstracciones e infraestructura para la definición y ejecución de servicios compuestos. Uno de los aspectos de este *middleware* son los lenguajes de modelado de definiciones de servicios compuestos. Actualmente, BPEL4WS (Business Process Execution Language for Web Services) [8] es el lenguaje dominante para composición de servicios Web, y está en proceso de estandarización dentro de OASIS con el nombre de WS-BPEL.

Otras propuestas que describen arquitecturas de composición de servicios son el proyecto CANS [77], el proyecto SAHARA [188] de la universidad de Berkeley, SWORD de la de Stanford [181], el proyecto Ninja [84] o el amplio trabajo realizado por el grupo de Klara Nahrstedt en la universidad de Illinois [157, 120, 121, 90], parte del cual se analizará en más detalle en el apartado 2.7.2.

2.6.1. Composición de servicios para la construcción de aplicaciones genéricas

La composición de servicios dispersos en una red para la construcción de aplicaciones genéricas (no de tiempo real) basándose en restricciones impuestas por el sistema, por los usuarios o por ambos, es un problema de las mismas características que el enrutado en redes con calidad de servicio. Dado que las especificaciones del usuario con respecto a la aplicación final pueden contemplar múltiples requisitos, el problema de encontrar un camino válido dentro de ese grafo es equivalente al problema MCP (*Multiple Constrained Path selection*) el cual es un problema NP-completo [118, 234] y que se define a continuación.

Definición 2.2. Dada una red representada por el grafo dirigido $G = (V, E)$, donde V es un conjunto de n nodos y E el conjunto de m enlaces. El problema de seleccionar un camino p desde un nodo origen n_i a un nodo destino n_j, tal que se cumpla para dicho camino $f_1(p) \leq k_1$ y $f_2(p) \leq k_2$, donde $f_1(p) = \sum_{\forall e_i \in p} f_1(e_i)$ y $f_2(p) = \sum_{\forall e_i \in p} f_2(e_i)$.

Debido a la importancia de sus aplicaciones, este problema ha sido estudiado de manera exhaustiva, habiéndose desarrollado múltiples heurísticos tanto para el caso de restricciones múltiples de carácter aditivo ($K \geq 2$) [243, 135, 241], como para sólo dos restricciones ($K = 2$) [55, 136]. En este último caso, conocido como DCLC (*Delay-Constrained Least Cost path problem*) los heurísticos desarrollados son capaces de obtener una solución en tiempo polinómico.

Este problema también se presenta, como ya se mencionó, en la selección de un camino válido en la composición de servicios complejos. Basándose en la solución propuesta por Chen y Nahrstedt [55] para DCLC, en [88], Gu et al proponen un algoritmo heurístico para la selección de un camino. La solución propuesta tiene en cuenta no sólo la calidad de servicio actual que ofrece un determinado servicio, sino también la calidad de servicio en media, y considera la carga del sistema como una variable más a ponderar. El heurístico selecciona todo el grafo a evaluar, realiza una transformación, a través de una combinación no lineal, de los parámetros de los nodos y arcos a un único parámetro, que será el coste de cada arco en su grafo, y aplica, posteriormente, el algoritmo de Dijkstra [68] para encontrar el camino más corto.

Los parámetros en los que se basa para elegir este camino son:

- Para cada servicio, s_i, $(LP_{s_i}, ST_{s_i}, CR_{s_i})$, que son, respectivamente, su probabilidad de pérdidas, su tiempo de ejecución y el ratio de CPU ($CR_{s_i} = cpu_{s_i}^{necesaria}/cpu_{s_i}^{disponible}$).

- Para cada enlace entre servicios, l_j, $(LP_{l_j}, D_{l_j}, BR_{l_j})$, su probabilidad de pérdidas, su retardo de transmisión y el ratio de ancho de banda requerido ($BR_{l_j} = bw_{l_j}^{necesario}/bw_{l_j}^{disponible}$).

La probabilidad de pérdidas total y el tiempo de servicio total de un camino lo hayan a partir de las siguientes ecuaciones:

$$1 - LP_x = \prod_{i=0}^{n+1}(1 - LP_{s_i}) \cdot \prod_{j=0}^{n}(1 - LP_{l_j}) \tag{2.37}$$

$$ST_x = \sum_{i=0}^{n+1} ST_{s_i} + \sum_{j=0}^{n} D_{l_j} \tag{2.38}$$

Para poder convertir la ecuación 2.37 en aditiva (requisito impuesto por el problema MCP) realizan la siguiente transformación:

$$\ln \frac{1}{1 - LP_x} = \sum_{i=0}^{n+1} \ln \frac{1}{1 - LP_{s_i}} + \sum_{j=0}^{n} \ln \frac{1}{1 - LP_{l_j}} \tag{2.39}$$

Dados estos parámetros y $(LP_{request}, ST_{request})$ requisitos impuestos por el usuario, el problema de encontrar un camino válido consiste en encontrar un camino x, tal que se cumplan:

$$\begin{aligned} \left(\ln \frac{1}{1 - LP_x} \leq \ln \frac{1}{1 - LP_{request}}\right) \wedge (ST_x \leq ST_{request}) \\ CR_{s_i} \leq 1, \forall s_i \\ BR_{l_j} \leq 1, \forall l_j \end{aligned} \tag{2.40}$$

.

Para calcular el coste de cada enlace se basan en los valores medios establecidos en los contratos a nivel de servicio, SLA, y en los valores actuales medidos a través de monitorización constante de los servicios. El cálculo del coste de cada enlace $l_k = (s_i, s_j)$ para poder aplicar el algoritmo de Dijkstra lo calcula de la siguiente

manera:

$$
\begin{aligned}
c(l_k) = & \underbrace{\frac{\ln \frac{1}{1-LP_{l_k}^{SLA}}}{\ln \frac{1}{1-LP_{l}^{min}}} + \frac{\ln \frac{1}{1-LP_{s_j}^{SLA}}}{\ln \frac{1}{1-LP_{s}^{min}}} + \frac{D_{l_k}^{SLA}}{D^{max}} + \frac{ST_{s_j}^{SLA}}{ST^{max}}}_{\text{Valores medios especificados por los contratos SLA}} + \\
& + \underbrace{\frac{\ln \frac{1}{1-LP_{l_k}^{actual}}}{\ln \frac{1}{1-LP_{l}^{min}}} + \frac{\ln \frac{1}{1-LP_{s_j}^{actual}}}{\ln \frac{1}{1-LP_{s}^{min}}} + \frac{D_{l_k}^{actual}}{D^{max}} + \frac{ST_{s_j}^{actual}}{ST^{max}}}_{\text{Valores actuales}} + \\
& + \underbrace{\frac{CR_{s_j}}{CR^{max}} + \frac{BR_{l_k}}{BR^{max}}}_{\text{Condiciones de carga}}
\end{aligned}
\tag{2.41}
$$

Este algoritmo heurístico, basado en una extensión al algoritmo de Dijkstra, está pensado para redes que ofrecen calidad de servicio sobre Internet, y no es aplicable en el modelo de aplicación que se presentará, pues aunque este modelo cumple las restricciones de calidad de servicio impuestas por el usuario, no contempla las restricciones temporales impuestas por un sistema de tiempo real. En el modelo de aplicación elegido en la presente tesis, la selección de un determinado perfil de servicio no sólo ha de asegurar una calidad, sino también la planificabilidad en el nodo que alberga dicho servicio y, además, la planificabilidad del protocolo de comunicaciones. Es decir, el algoritmo de composición de aplicaciones estará muy vinculado a las características temporales del protocolo de red elegido.

2.7. Trabajos relacionados

En el apartado anterior se han presentado distintas tecnologías relacionadas con los objetivos de esta tesis. En este apartado se realiza un análisis de distintos trabajos cuyas temáticas se relacionan íntimamente con el ámbito de la misma. En primer lugar, se analizarán propuestas de aplicación de los principios de arquitecturas orientadas a servicio tanto a sistemas de tiempo real como a sistemas con calidad de servicio. En el apartado 2.7.3, analizaremos una propuesta para dotar de flexibilidad en la gestión de la calidad de servicio a entornos multimedia embarcados. Después nos centraremos en trabajos cuyo objetivo es dotar de flexibilidad a sistemas de tiempo real, tanto centralizados como distribuidos, mediante la extensión del modelo de componentes y la definición de arquitecturas que lo soportan. El cuadro 2.3 presenta un resumen de las principales características de los mismos. Finalmente, como conclusión a este análisis, se expone cuáles son los principales problemas sin resolver en este ámbito, o cuya solución podría ser mejorada.

Proyecto	Paradigma	Tecnología	Contribución
RTSOA (Arizona State University)	Orientación a servicios + Tiempo Real	Middleware basado en máquina virtual	Identificación de elementos que debería poseer una arquitectura de tiempo real orientada a servicios
SpiderNet. Tesis de X. Gu (University of Illinois)	Orientación a servicios + Composición dinámica + Adaptación y gestión dinámica QoS	Middleware a nivel de aplicación genérico. Prototipo desarrollado en PlanetLab	Definición de un entorno middleware de QoS sobre Internet que permita la composición automática de servicios así como su reconfiguración en función de los requisitos de QoS impuestos por el usuario. El entorno se basa en la definición a nivel de aplicación de una red virtual formada por los proveedores de servicio y los usuarios finales.
HOLA-QoS (UPM y UC3M)	Tiempo Real + Adaptación y gestión dinámica de recursos	Arquitectura gestor QoS de middleware	Gestión dinámica de QoS en entornos multimedia embarcados
ACCORD y *RTCOM*. Tesis de Tešanović (Linköping Universitet)	Componentes + Composición estática + Aspectos + Adaptación y gestión dinámica QoS + Tiempo Real	Entornos centralizados	Definición de un modelo y una metodología de diseño de sistemas de tiempo real para la composición de sistemas y su configuración que soporte reconfiguración dinámica de calidad de servicio
Componentes para RTRMI. (Texas University)	Componentes + Tiempo Real+ Protocolos de descubrimiento	RTSJ	Definición de modelo de componentes para RTSJ y modelo de distribución para entornos móviles RTRMI
Tecnología de componentes (Texas Univ.)	Componentes + Tiempo Real	Arquitectura cliente-servidor	Definición de modelo de componentes genérico reutilizable y arquitectura cliente-servidor con requisitos temporales
QuO (BBN Technology) y CIAO (University of Washington)	Componentes + Composición estática + Aspectos + Adaptación y gestión dinámica de QoS + Tiempo Real	Middleware RTCORBA y su modelo de componentes	Gestión de QoS estática y dinámica en un middleware RT-Corba a través del uso de componentes y aspectos

Cuadro 2.3: Resumen de trabajos relacionados

2.7.1. RTSOA (Arizona State University)

El grupo de trabajo de la Universidad de Arizona, proviene del campo de la investigación en arquitecturas orientadas a servicios. En el trabajo "*RTSOA: Real-Time Service Oriented Architecture*" [228], presentado en Octubre de 2006, se hacen eco de que la investigación existente hasta el momento en el campo se ha centrado en la calidad de servicio desde el punto de vista del proveedor de servicios, pero que no ha ahondado en cuestiones como la integración de servicios e interoperabilidad desde el punto de vista del consumidor o del desarrollador de aplicaciones. En este documento realizan una primera aproximación, haciendo una lista de los elementos genéricos que, a su entender, debería contemplar el diseño de una arquitectura orientada a servicios que dé soporte de tiempo real, como podría ser la implementación

de una ontología que refleje las propiedades de tiempo real, que los servicios estén almacenados en un repositorio de servicios previamente verificados o la necesidad del uso de una máquina virtual genérica de tiempo real. También asumen protocolos de reserva de ancho de banda que ofrezca garantías de tiempo real, o el uso de un protocolo de intercambio de mensajes como RT-SOAP [99] que ofrezca garantías temporales. Finalmente, presentan un algoritmo heurístico de composición de servicios. Dicho algoritmo se centra en aplicaciones en secuencia, suponiendo que son todos planificables y que la instanciación de una nueva aplicación no afectará a las prestaciones ni a la planificación del sistema. En otras palabras, obviando las restricciones de tiempo real existentes tanto de los nodos como de la red, sobre la que no realizan ninguna consideración.

2.7.1.1. Análisis

Este es un trabajo muy preliminar y genérico. Se limita a enumerar, sin profundizar, las capacidades que debería soportar un entorno para el desarrollo de sistemas de tiempo real orientados a servicios, sin realizar un estudio detallado ni proponer ninguna solución concreta, y hablando en todo momento en abstracto. El principal problema de este trabajo es que parten de su conocimiento en arquitecturas SOA y no ahondan en las cuestiones relacionadas con tiempo real, resultando en un análisis excesivamente somero.

La importancia de este trabajo radica en que pone de manifiesto el interés por parte de la comunidad científica de dotar a arquitecturas basadas en servicios de características y soporte de tiempo real.

2.7.2. Tesis doctoral de Xiaohui Gu (University of Illinois)

El objetivo de la tesis doctoral de Xiaohui Gu [90], presentada en el 2004, es la definición de un entorno *middleware* de calidad de servicio para entornos distribuidos heterogéneos basado en servicios, *SpiderNet*, que permite la composición automática de los mismos en base a los requisitos de *QoS* impuestos por el usuario. Este entorno ha sido diseñado e implementado sobre la infraestructura que ofrece hoy en día Internet, no exigiendo ningún cambio en la misma ni en los sistemas operativos sobre los que se despliega.

Las funcionalidades más importantes que ofrece este entorno *middleware* son: (1) *descubrimiento de servicios* basado en palabras clave; (2) *composición de servicios inicial*, en base a su funcionalidad y a los requisitos impuestos por el usuario de *QoS*, para lo que ofrece un entorno de especificación de servicios al usuario basado en XML [89, 239], así como en las restricciones impuestas por los recursos; (3) *recuperación en presencia de fallos en tiempo de ejecución*, detectando los fallos o violaciones del contrato de calidad de servicio (SLA) a través de la monitorización de la *QoS* de los servicios compuestos. En la figura 2.6, se muestra el protocolo seguido para la inicialización y ejecución de servicios compuestos en *SpiderNet*.

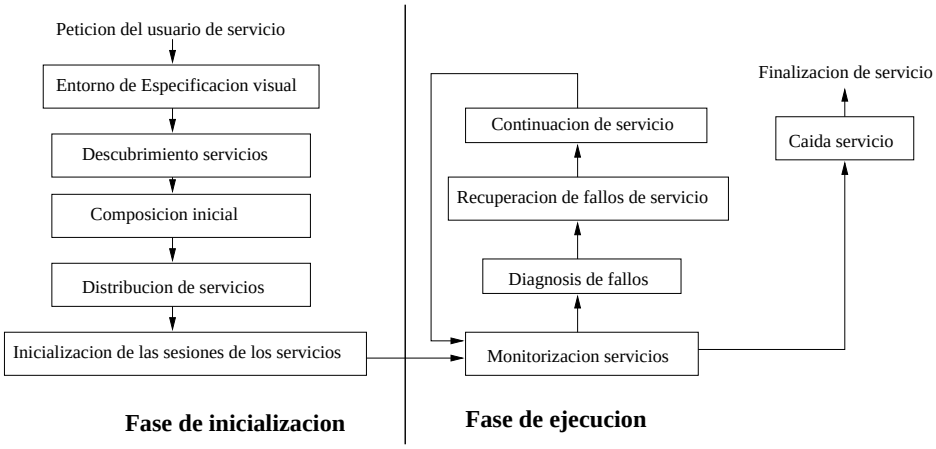


Figura 2.6: Protocolo de composición de servicios de *SpiderNet*

Además, permite a los proveedores de servicio añadir o eliminar servicios dinámicamente, y ofrece la posibilidad de gestionar la topología de la red de servicios usada por el núcleo de su arquitectura, *core SpiderNet*, así como de realizar reparto de carga de forma dinámica.

Dicha arquitectura está compuesta, como se puede observar en la figura 2.7, por dos subsistemas: (1) una abstracción en forma de red de enlaces virtuales a nivel de aplicación entre los servicios dispersos por Internet que ofrece el sistema, llamada *core SpiderNet*, que provee los servicios y el enrutamiento de datos a nivel de aplicación. En esta tesis se presentan dos diseños diferentes de este núcleo: $(1a)$ *Utility SpiderNet* [88], para entornos reducidos y fácilmente gestionables, como la infraestructura que utiliza una empresa, y $(1b)$ *Peer-to-peer SpiderNet* [91, 92], para entornos auto-organizados, como una red P2P; (2) el subsistema *access SpiderNet* cuya función es garantizar y adaptar la entrega de una determinada *QoS* de servicio para que no influyan en ésta la movilidad de los usuarios en entornos ubicuos, las limitaciones en los recursos ni las fluctuaciones de éstos. En la tesis de Gu se presenta un diseño para entornos ubicuos, *Ubiquitous SpiderNet* [87], que gestiona la distribución de servicios en este tipo de entornos utilizando agregación de dispositivos.

Por otra parte, la arquitectura definida es una arquitectura de tres capas: la primera constituye la interfaz entre el usuario y *SpiderNet*, que permite al usuario expresar requisitos de *QoS* y de recursos así como especificar funcionalidades complejas usando un grafo de funciones; la segunda crea dinámica y automáticamente las aplicaciones a partir de los recursos disponibles en la red; y la tercera es la red a nivel de aplicación de enlaces virtuales entre servicios.

En este trabajo también se presenta un modelo de componente de servicio. Cada servicio posee puertos de entrada y de salida y se comunica con los demás a través de mensajes asíncronos en forma de unidades de datos de aplicación (ADUs, *Application Data Units*). Divide además cada componente en 4 ítems: (1) su funcionalidad;

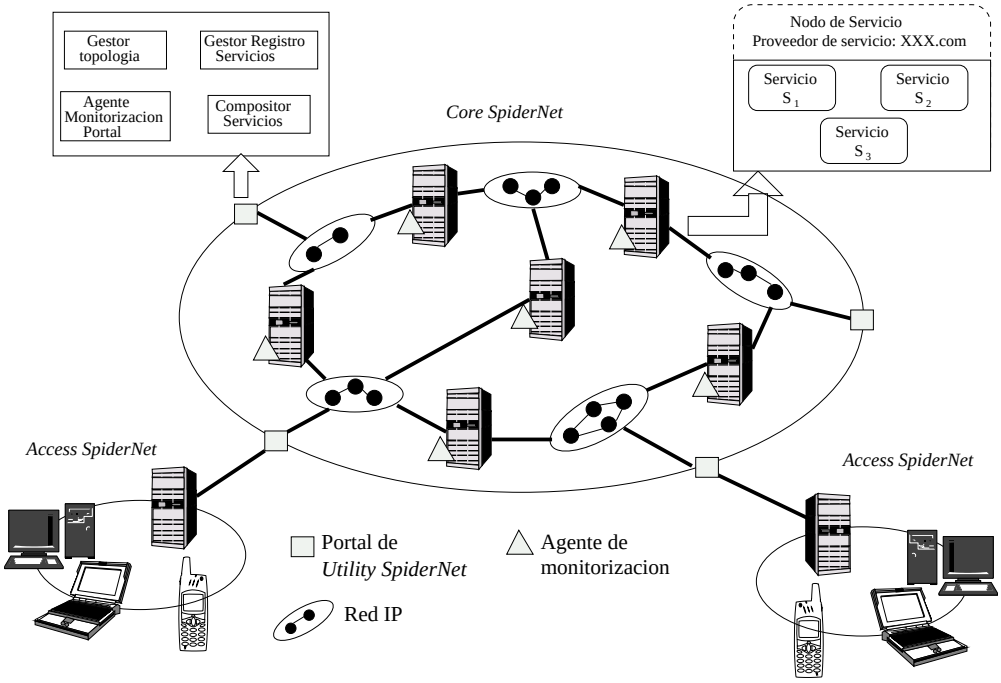


Figura 2.7: Estructura en subsistemas de *SpiderNet* y gestión de la monitorización en *Utility SpiderNet*

(2) su código asociado; (3) sus meta-datos estáticos, como pueden ser su localización, requisitos estáticos de *QoS* entrante (por ejemplo, formato, tasa) y saliente y políticas de adaptación (expresadas mediante condicionales); y (4), sus metadatos dinámicos, que caracterizan su rendimiento dinámico, como su tiempo de servicio o su tasa de pérdidas que pueden ser expresadas como un valor determinado, o en forma de distribución estadística.

La composición de servicios requiere la exploración del grafo de posibles configuraciones en la inicialización del sistema, y que se compruebe para cada par productor-consumidor su compatibilidad, es decir, que no se violan los requisitos de entrada del servicio consumidor. Esta comprobación se realiza en la inicialización del sistema y cada vez que hay una reconfiguración, y se divide en dos fases: (1) la comprobación de la consistencia de parámetros de *QoS*, de orden $O(D)$, siendo D la dimensión de los vectores de calidad de servicios; y (2) la comprobación de sus políticas de adaptación, cada una de ellas representada por un hipercubo en un espacio C-dimensional. Esta comprobación es de orden $O(T_a \cdot T_b \cdot D \cdot C^2)$, siendo T_i el tamaño del conjunto de reglas de adaptación del servicio s_i.

La monitorización de los sistemas en *Utility SpiderNet* la realizan los portales, situados en las redes de entrada a *SpiderNet*, que mantienen la información sobre las sesiones activas y los servicios involucrados. En cada nodo del sistema, como se puede observar en la figura 2.7, existe un agente de monitorización de fallos, que en el caso de detectar un fallo en ese nodo o en algún vecino, notifica a todos los

portales. Estos, comprueban su tabla local y en el caso de ser el portal de entrada de una aplicación afectada, ejecutan un algoritmo de recomposición de la aplicación, con un porcentaje de éxito del $90\,\%$. En *P2P SpiderNet*, la composición se realiza en cascada, una vez seleccionado un nodo de entrada de la aplicación compuesta, éste realiza comprobaciones de todos los candidatos a ser el siguiente nodo, teniendo en cuenta el siguiente salto desde cada uno de ellos. Crea, a partir de esta información, un árbol, que guarda en memoria, y realiza su decisión. Si existe la necesidad de reconfigurar el sistema, recurre al árbol creado durante su inicialización para tomar una decisión. La monitorización la realiza el nodo de entrada de la aplicación, sondeando periódicamente el sistema a través del envío de mensajes a tasa baja.

En base a las ideas desarrolladas en este trabajo y a la infraestructura *SpiderNet* elaborada en el mismo, Gu ha desarrollado *BridgeNet* [93]. En esta propuesta, utilizando los mismos mecanismos desarrollados en *SpiderNet*, los servicios ofrecidos por la red se transforman en flujos multimedia que deben ser distribuidos a los usuarios finales a través de una red virtual formada por los nodos del sistema. Las aplicaciones creadas con esos servicios son el camino que debe, a través de esa red, seguir el flujo de datos, ofreciendo la posibilidad de agregar nuevas fuentes, disminuir el número de las existentes, realizar reparto dinámico de carga y recuperarse frente a errores.

2.7.2.1. Análisis

El trabajo de esta tesis en composición de servicios para la provisión de *QoS* a nivel de aplicación en entornos distribuidos es especialmente completo, puesto que define un modelo para la composición de servicios, algoritmos para dicha composición y constituye, hasta donde tenemos conocimiento, la primera aproximación a la composición de servicios en entornos P2P. Sin embargo, las ideas desarrolladas en este trabajo, no se pueden implementar directamente en entornos embarcados distribuidos de tiempo real, donde los dispositivos y las redes son de capacidad limitada, sujetos a restricciones temporales muy estrictas.

Para poder ser aplicable esta solución, siempre en entornos de tiempo real flexibles y sobre una infraestructura de tiempo real (tanto de red como de sistema operativo), debería tener en cuenta la planificabilidad en los nodos y en la red subyacente, de forma que se tuviese en cuenta en la composición no sólo los requisitos de aplicación, sino también los de todo el sistema, exigiendo una comunicación tanto con el sistema operativo como con la capa de red y una especificación de requisitos que los contemplase. También exige la implementación de algoritmos de composición acotados temporalmente.

Por otra parte, la solución para la monitorización del sistema, con la existencia de agentes en cada nodo, debería ser implementada de tal forma que no fuese una fuente de indeterminismo, dado el carácter asíncrono que las caracteriza. También puede ser fuente de indeterminismo la solución basada en enviar tramas de prueba para la comprobación de que las aplicaciones no tienen errores. Además, el uso sin

ninguna política temporal de mensajes asíncronos para la comunicación entre nodos no asegura el cumplimiento de las restricciones temporales que tiene una aplicación de tiempo real, puesto que un servicio puede recibir mensajes de múltiples nodos con distintas restricciones temporales, lo que haría necesario el uso de, por ejemplo, colas con prioridades u otro mecanismo de comunicación.

2.7.3. HOLA-QoS (Universidad Politécnica de Madrid y Universidad Carlos III de Madrid)

Este trabajo realizado de forma conjunta por el grupo de tiempo real de la UPM y el de la UC3M, desarrollado en la tesis de Marisol García Valls [78] para sistemas monoprocesador y, posteriormente, extendido en la tesis de José Ruiz [194] para sistemas multiprocesador, tiene como objetivo el desarrollo de una arquitectura de un sistema de gestión de calidad de servicio con niveles [79], HOLA-QoS (*Homogeneneous and Open Layered Architecture for Quality of Service management*), que dé soporte para construir sistemas multimedia embarcados flexibles y abiertos.

Esta arquitectura actúa como un *software* intermedio que se ejecuta sobre un sistema operativo de tiempo real y que gestiona la ejecución de las aplicaciones multimedia. Este cometido lo lleva a cabo mediante el control de la asignación de recursos a las aplicaciones en cada momento. La operación básica del sistema se asienta sobre un *modelo de contrato* que tiene lugar entre las aplicaciones y el gestor. Según ese modelo, el gestor garantiza a las aplicaciones una determinada asignación de recursos y, por su parte, las aplicaciones son capaces de ofrecer una determinada calidad de salida para dicha asignación. Las aplicaciones pueden ofrecer distintos niveles de calidad de salida desde el punto de vista del usuario, y a cada uno de éstos le corresponde una determinada asignación de servicios, incluso pudiendo tener la aplicación una configuración interna distinta para cada nivel (por ejemplo, el conjunto de filtros que utilizan). Además, las tareas de las aplicaciones tienen asignadas cuotas para ejecutar en un recurso del sistema, como pueden ser tiempo de computación, tamaño de memoria o ancho de banda.

La arquitectura de HOLA-QoS, como se puede ver en la figura 2.8, consta de cuatro capas y una base de datos que contiene información necesaria para la gestión. La base de datos contiene el modelo de todo el sistema, es decir, la descripción de todas las aplicaciones y sus parámetros necesarios, como son los niveles de calidad a los que puede funcionar cada aplicación. Todas las capas, excepto la capa de control de ejecución, acceden a la base de datos para llevar a cabo su operación.

La funcionalidad de cada capa es la siguiente:

- Capa de gestión de calidad de servicio: mantiene la calidad del sistema percibida por el usuario. Decide qué aplicaciones se ejecutan, establece los requisitos básicos de la aplicación (niveles de calidad) y establece los parámetros de sistema.

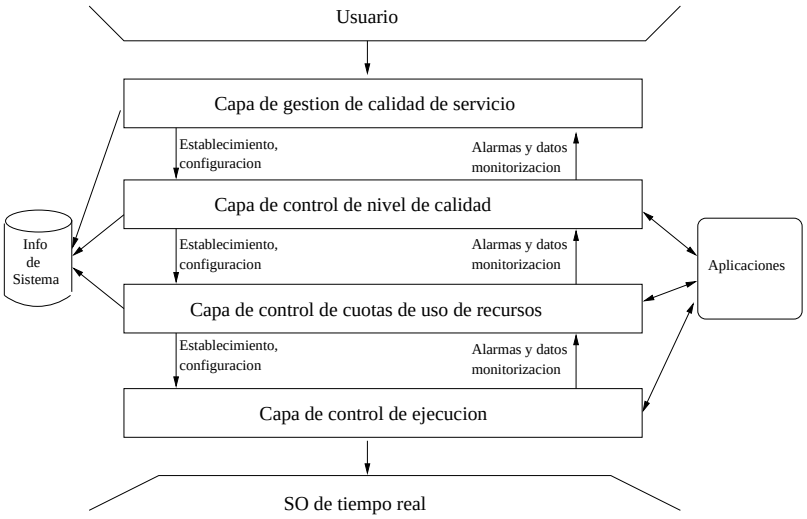


Figura 2.8: Vista general de la arquitectura HOLA-QoS

- Capa de control de nivel de calidad: mantiene la calidad de una aplicación. Cumple los requisitos de una aplicación (nivel de calidad actual), encuentra las cuotas apropiadas (configuración del nivel de calidad) para ejecutar el nivel de calidad impuesto por el nivel superior. Ejecuta el mecanismo de cambio de modo (establece los niveles de calidad).

- Capa de control de cuotas de uso de recursos: aplica la prueba de planificabilidad. Realiza la prueba de admisión para los grupos de tareas o aplicaciones, registra las operaciones de las aplicaciones sobre sus tareas.

- Capa de control de ejecución: impone las cuotas de tiempo de ejecución de cada tarea. Vigila la imposición de cuotas que han sido garantizadas a las aplicaciones mediante la contabilidad del uso de recursos, establece la correspondencia entre las cuotas y los recursos disponibles.

Cada uno de las capas de la arquitectura consta de los mismos componentes: (1) componente de admisión; (2) componente de establecimiento de configuraciones, encargado de instaurar las nuevas configuraciones substituyéndolas por las actuales, esto implica un cambio de modo en el sistema; (3) componente de monitorización, determinada la carga del sistema, puede adoptar decisiones de adaptación a la carga actual del sistema ;(4) componente de tratamiento de alarmas; y (5), componente de interfaz con actores externos, por ejemplo las aplicaciones y el usuario.

La estructura de control del sistema es jerárquica: las capas superiores de la arquitectura contienen las estrategias o políticas (reglas de decisión o algoritmos a alto nivel) que gobiernan los mecanismos (actividades automáticas del sistema) que pertenecen a las capas inferiores. Las capas superiores ejecutan tareas de gestión menos frecuentemente que las capas inferiores, por ejemplo, la frecuencia en la que

el sistema debe modificar las cuotas de recursos asignados a las tareas es superior a la frecuencia a la que se producen cambios en los niveles de calidad, o peticiones de ejecución de nuevas aplicaciones en el sistema.

2.7.3.1. Análisis

El interés de este trabajo, radica en la aproximación realizada al definir de forma jerárquica los niveles internos de una arquitectura de gestión de calidad de servicio que permite una carga dinámica de las aplicaciones que va a ejecutar. Permite, además, la definición de distintos modos en las aplicaciones a ejecutar, soportando cambios de modo. Las aplicaciones, en esta arquitectura, pueden estar compuestas por diferentes filtros y cambiar su configuración interna según la calidad de salida deseada. Si entendemos un filtro como un servicio, las aplicaciones estarían compuestas de servicios que cambian dinámicamente según las necesidades del sistema, del usuario y de la propia aplicación. Así, aunque no se puede aplicar de forma directa a una arquitectura orientada a servicios, podría servir de base para el gestor de calidad servicio que sí sería necesario. Muchos de sus principios, como la separación de niveles o cómo gestiona los recursos también podrían ser utilizados.

2.7.4. Tesis de Tešanović (Linköping Universitet)

La tesis doctoral de Tešanović [217], presentada en el 2006, continuó el trabajo empezado en su proyecto de fin de carrera (*master thesis*) [216] del 2003. Estos dos trabajos tienen como objetivo la definición de un modelo y una metodología de diseño de sistemas de tiempo real que, utilizando componentes y aspectos, permita el diseño, la configuración y composición de sistemas de forma dinámica y estática, así como la provisión de componentes para la reconfiguración dinámica de la calidad de servicio, basándose en una estrategia de bucle de realimentación.

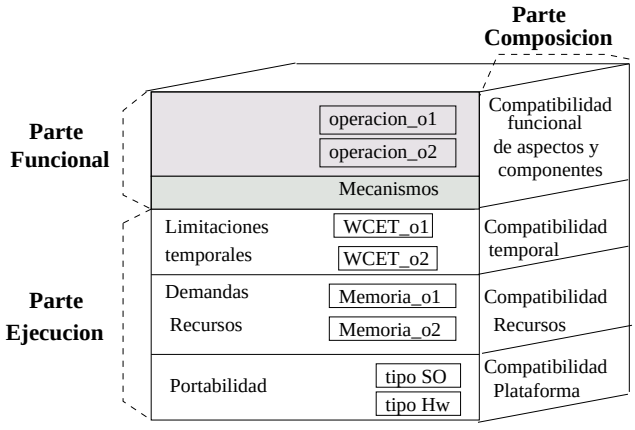


Figura 2.9: Modelo RTCOM

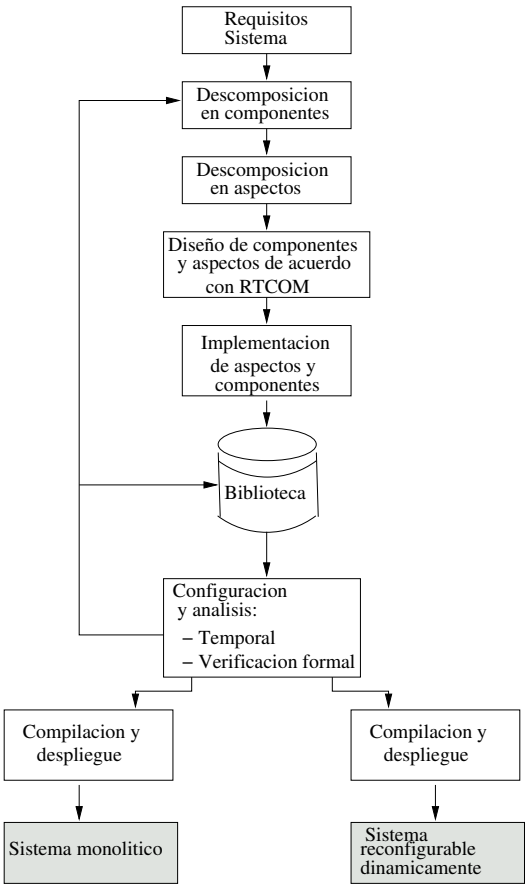


Figura 2.10: Desarrollo de un sistema de tiempo real atendiendo a la metodología ACCORD

En su proyecto fin de carrera define, en primer lugar, un modelo de componentes de tiempo real, RTCOM (*Real Time Component Model*), representado en la figura 2.9, que soporta la definición de aspectos, forzando a un ocultamiento de la información relacionada con la implementación de los componentes. Mientras que un componente se considera una *caja negra* para el usuario, la programación basada en aspectos es una *caja blanca*, donde el usuario puede modificar el comportamiento de un componente y su funcionalidad Así, los aspectos [129] son considerados como propiedades que modifican el comportamiento interno, semántico o de rendimiento de un componente *software*. En base a estas definiciones, se podría decir que el modelo RTCOM llega a un compromiso en un modelo de componentes de *caja gris*, donde se puede modificar el comportamiento de un componente sin llegar a saber por completo cómo está desarrollado.

El trabajo presentado en su proyecto fin de carrera define, además, una metodología de diseño de sistemas de tiempo real, ACCORD [219] (*Aspectual Component-Based Real-Time System Development*), ver figura 2.10, en tres fases: en primer

lugar, descompone el sistema en un conjunto de componentes, después analiza los aspectos, centrándose en requisitos no funcionales, presentes en el sistema de tiempo real (que pueden restringirse a un componente o extenderse a lo largo de todo el sistema) y, finalmente, utilizando RTCOM, los implementa. La especificación del WCET de cada componente se compone de dos partes: una parte *interna* que no se ve influenciada por la definición de aspectos y una parte *externa* que representa el comportamiento temporal que puede ser modificado por influencia externa. Basándose en esta definición, implementa un analizador de WCET que analiza el comportamiento temporal en el peor caso de cada componente y ofrece la posibilidad de que, una vez seleccionados una serie de componentes para formar un sistema, el usuario pueda cambiar la configuración manualmente.

En su tesis doctoral extiende el modelo RTCOM y ACCORD para ofrecer composición estática y dinámica en los sistemas diseñados con este último (reconfiguración). Además, define un método para reconfigurar dinámicamente la calidad de servicio ofrecida en función del rendimiento deseado y del nivel de rendimiento ofrecido a la aplicación, utilizando un sistema realimentado de control [218].

2.7.4.1. Análisis

Pese a que el objetivo de este trabajo es el diseño de sistemas centralizados de tiempo real basados en componentes, y no se puede aplicar directamente a nuestro campo de interés, es interesante la solución que ofrece para la reconfiguración y composición de sistemas.

El hecho de que los componentes puedan exhibir comportamientos diferentes en función de los aspectos que ofrece, se puede asimilar a los mecanismos de cambio de modo utilizados en sistemas de tiempo real tanto centralizados como distribuidos. Los aspectos de cada componente son conocidos previamente a la ejecución del sistema y la reconfiguración de la *QoS* se basa en este conocimiento previo, pudiendo elegir de entre todas las posibles configuraciones del componente la más adecuada para el momento actual, pero restringiéndose a un conjunto limitado de combinaciones en el sistema. Esta aproximación es menos flexible que la planteada en SOA, donde una misma funcionalidad puede ser soportada por diferentes servicios, localizados en el mismo o en nodos diferentes, que pueden aparecer en el sistema dinámicamente, sin necesidad de conocer previamente sus características.

Un punto débil de su aproximación reside en que para cada componente sólo estará disponible en el sistema una configuración, y el hecho de reconfigurarlo obliga a una carga dinámica del nuevo código que afirman, basándose en comprobaciones empíricas, que no compromete los requisitos de tiempo real del sistema.

Por otra parte, el soporte a la reconfiguración dinámica de *QoS* lo realizan a través de la inclusión de componentes específicos en el sistema diseñado que, configurados a través de aspectos, realizan una monitorización de todos los componentes y reajuste del sistema usando un bucle de realimentación. Esta estrategia es válida en entornos centralizados, como el que nos presenta este trabajo, donde puede no

suponer una carga excesiva la monitorización constante, sin embargo, no es directamente aplicable a entornos distribuidos, ya que podría comportar una carga excesiva en la red.

2.7.5. Tecnología de componentes para RTRMI (Texas A & M University)

El trabajo del grupo de tiempo real de la Universidad de Texas se centra en proveer de flexibilidad a los sistemas de tiempo real mediante la definición de una tecnología de componentes capaz de migrar a otras localizaciones, para así poder realizar reconfiguraciones *on-line*.

Para conseguir esta meta, partieron del trabajo del grupo en el sistema *Agile* [58], una extensión a la especificación de Java de tiempo real [74] (*Real-Time Specification for Java*, RTSJ). El sistema *Agile* dota a RTSJ de planificadores EDF [146], y TBS [205, 206] (*Total Bandwidth Server*). En la tesis de Sangig Rho [191], además, realizan las extensiones necesarias para dotar a RTSJ de un mecanismo de invocación remota, RMI, con características de tiempo real.

Cada componente *Agile* exhibirá un conjunto de métodos, cada uno de los cuales tendrá asociado un hilo *escuchador*, encargado de atender a las peticiones que llegan de los clientes. Cada una de estas peticiones generará un hilo *trabajador*. Cada conjunto de hilos trabajadores de un método estará planificado por TBS, y existirá, además, un hilo planificador encargado de decidir quién entra en cada momento en la CPU. Para migrar tanto el código como el estado de un componente [57] utilizan la serialización y deserialización estándar Java, no permitiendo que un componente migre hasta que haya procesado todas las peticiones de los clientes (es decir, el único hilo asociado al componente que debe estar vivo antes de la migración es el hilo *escuchador*). En la figura 2.11 se muestra el protocolo seguido para migrar un componente a otra ubicación.

Como un servidor no debería poder migrar un componente a cualquier ubicación, propusieron en [56] un protocolo de descubrimiento y ubicación de componentes, *REsource ALlocator* (REALTOR). Cada servidor crea una comunidad, que mantiene dinámicamente, de posibles nodos capaces de recibir uno de sus componentes. Para ello, cuando llega a una red, manda un mensaje `HELP` indicando las características y consumo de recursos de sus componentes, para que posibles nodos interesados le envíen un mensaje de aceptación, `PLEDGE`, indicando a su vez, la cantidad de recursos libres que poseen. El mantenimiento de la comunidad se realiza mediante el envío de mensajes `HELP` y `PLEDGE` de forma periódica, con comunicaciones originadas en el organizador. En caso de que los recursos utilizados, el conjunto de componentes o los requisitos de éstos varíen antes del vencimiento del período, se evaluará si esos cambios sobrepasan un umbral previamente definido. En ese caso, se comunica a la comunidad mediante el mensaje adecuado (`HELP` si es el organizador de la comunidad, `PLEDGE` si es un nodo partícipe).

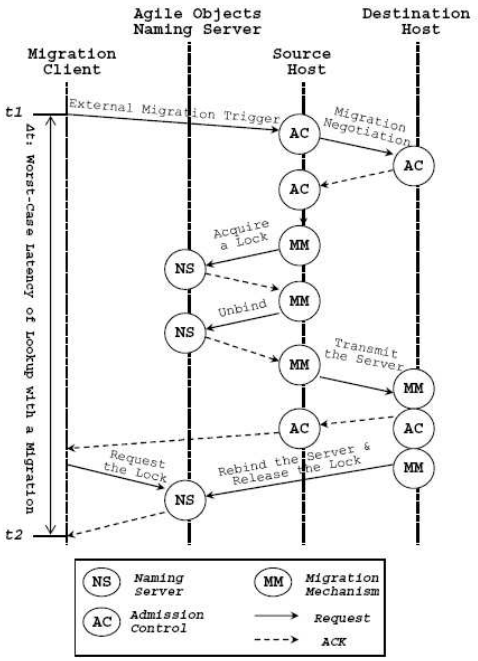


Figura 2.11: Protocolo para la migración de un componente a otra ubicación (figura tomada de [191])

2.7.5.1. Análisis

El interés de esta propuesta radica en la aplicación del modelo de componentes a una tecnología específica como RTSJ, así como la aproximación realizada para dotar de flexibilidad a un sistema distribuido de tiempo real basado en una arquitectura cliente-servidor.

Sin embargo, su solución carece de una definición de los elementos que debería poseer una arquitectura distribuida de tiempo real. Se reduce a la definición del modelo de hilos que debería soportar cada componente y a un mecanismo de migración definido sobre la serialización estándar de RMI, sin realizar consideraciones sobre la falta de determinismo temporal tanto de RMI como de dicha serialización. No se tiene en cuenta en este trabajo la planificación de red ni en el momento de realizar la migración de servicios ni en el protocolo REALTOR. En términos generales, se trata de un trabajo poco maduro y no aplicable directamente a entornos distribuidos de tiempo real.

2.7.6. Tecnología de componentes de tiempo real (Texas A & M University)

El grupo de la universidad de Texas, basándose en el trabajo expuesto y analizado en el apartado anterior, ha definido [233] una arquitectura y un modelo de

componentes de tiempo real genéricos para aplicaciones distribuidas, basadas en el modelo cliente-servidor. El modelo de componentes y de composición de los mismos permite a cada servidor de aplicaciones ofrecer capacidades de tiempo real, pero no contempla la situación en la que una aplicación está formada por invocaciones sucesivas de componentes ubicados en diferentes servidores.

Este modelo define una capa de abstracción de los recursos utilizados (*hardware*, sistema operativo...), tal y como se puede ver en la figura 2.12, consiguiendo así aislar completamente al programador de aplicaciones de las características internas de cada componente y permitiendo que el diseño de las aplicaciones a alto nivel no deba ser modificado ante el cambio en la implementación de alguno de estos, o en el soporte *hardware*.

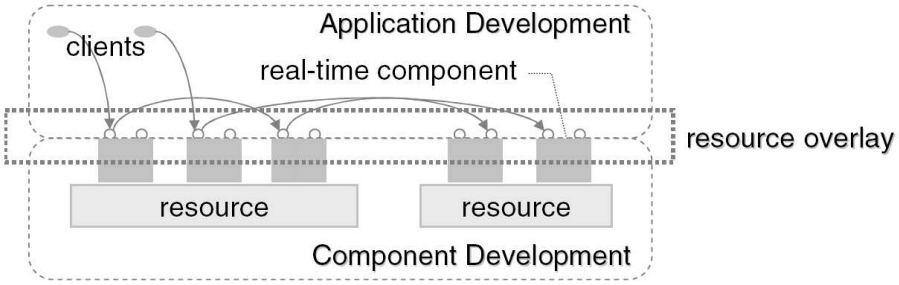


Figura 2.12: Aislamiento de los recursos mediante el uso de componentes (figura tomada de [233])

Cada componente ofrece un conjunto de métodos al exterior. Para caracterizar temporalmente estos métodos utilizan el concepto de clases de servicio. Cada clase de servicio i del componente e queda definida por un conjunto de métodos, $\Theta_{e,i}$, una función de llegadas de invocaciones, $\mathcal{A}_{e,i}(I)$ (por ejemplo, una función de llegadas de ráfagas puede ser definida por $\mathcal{A}(I) = \sigma + \rho I$, siendo σ el tamaño de ráfaga, ρ tasa media de llegadas e I el intervalo temporal considerado) y un plazo asociado $D_{e,i}$. Así, si un usuario selecciona una determinada clase i, se le asegura que sus invocaciones serán atendidas y procesadas dentro del plazo de la clase, $D_{e,i}$, siempre y cuando éstas se ajusten temporalmente a la función de llegadas de invocaciones de esa clase, $\mathcal{A}_{e,i}(I)$. Nótese que un método puede pertenecer simultáneamente a varias clases de servicio e, incluso, los conjuntos de métodos de diferentes clases de servicio pueden ser los mismos, permitiendo al usuario elegir la calidad de servicio que necesita. Por otra parte, definen para cada clase un tiempo de ejecución en el peor caso, $C_{e,i}$, que es el máximo de todos los tiempos de ejecución en el peor caso de los métodos de dicha clase.

La arquitectura desarrollada para cada servidor es la mostrada en la figura 2.13, donde para cada componente se utiliza un planificador de prioridades estático, utilizando el identificador i de cada clase como prioridad, y para asegurar el aislamiento temporal entre componentes, un servidor TBS [205, 206] (*Total Bandwidth Server*).

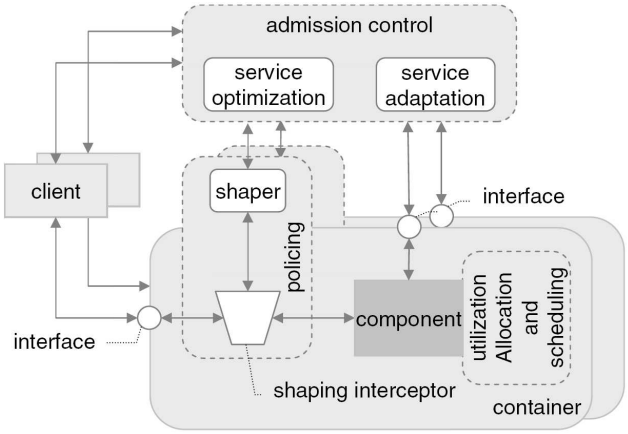


Figura 2.13: Arquitectura del sistema (figura tomada de [233])

Esta planificación a dos niveles la realiza el módulo *Utilización, Asignación, Planificación.*

Antes del despliegue de cada componente se define su interfaz de tiempo real, $(\Theta_{e,i}, \mathcal{A}_{e,i}, D_{e,i})$, y se guarda en una base de datos interna al control de admisión. Cuando un cliente desea empezar una nueva tarea, envía una petición de admisión al módulo de control de admisión. Éste, basándose en un mecanismo de control de admisión (definido en [233]), admite o no la nueva tarea. En caso de que se admita, se crean nuevas instancias de los componentes involucrados en la tarea y se configuran las políticas de admisión para que la tasa de llegadas de invocaciones no exceda un determinado límite. Como se puede ver en la figura 2.13, en esta arquitectura se utiliza un *leaky bucket* como mecanismo de admisión. Han implementado esta arquitectura sobre el RTRMI definido en [191] utilizando Enterprise JavaBeans [213].

2.7.6.1. Análisis

Como ya se ha comentado anteriormente, la solución de distribución propuesta por el grupo de tiempo real de Texas se basa en una arquitectura cliente-servidor y, aunque no se diga expresamente, el modelo de componentes está fuertemente influenciado por el paradigma de objeto remoto. Así, la composición que proponen es análoga a una solución centralizada en el nodo servidor, donde la aplicación compuesta, cuyos componentes son conocidos *a priori*, reside en un único nodo y es la resultante de invocar a uno o más componentes en secuencia. Esta solución está en un estado muy preliminar, donde para soportar la composición dinámica de aplicaciones, que proponen como uno de los objetivos del trabajo, es necesaria la definición de mecanismos de composición de aplicaciones, y de selección de los distintos componentes por parte del cliente o del *middleware*. De momento, debido a la solución de planificación en los nodos que proponen, la arquitectura propuesta en este trabajo no puede aplicarse en entornos donde la aplicación invocada por un

cliente, se expanda a través de diferentes nodos, siendo necesaria una extensión en este sentido, que permita planificación distribuida y comunicación entre diferentes servidores o la extensión del servidor como cliente, a su vez, de otros.

2.7.7. QuO (BBN Technology) y CIAO (University of Washington)

En el ámbito de *middleware* de tiempo real, en concreto RT-CORBA, BBN Technology ha desarrollado un *middleware* de tiempo real adaptativo que permite el desarrollo de sistemas distribuidos de tiempo real, Quality Objets (QuO) [245], que permite a los desarrolladores usar técnicas de programación orientada a aspectos para separar la lógica de negocio de los requisitos de calidad de servicio. El entorno QuO permite a los desarrolladores especificar los requisitos de calidad de servicio, los elementos del sistema que deben ser monitorizados y las políticas de adaptación de *QoS* durante el tiempo de ejecución.

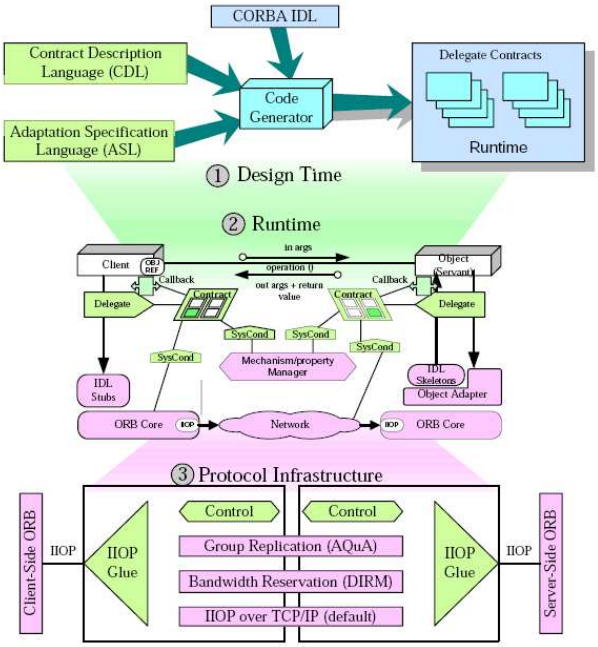


Figura 2.14: Provisión dinámica de QoS en QuO (figura tomada de [232])

Los elementos de la arquitectura que se encargan de dar soporte a esta calidad de servicio adaptativa son: (1) Contratos para especificar el nivel de calidad de servicio deseado por un cliente, el nivel de calidad de servicio que un componente espera dar, y las acciones a tomar cuando cambian los niveles de calidad de servicio; (2) Delegados, que actúan como *proxies* locales de los componentes remotos. Cada delegado tiene la misma interfaz que el componente remoto, pero añade localmente el com-

portamiento dinámico basado en el estado actual de *QoS*; (3) Objetos de condición del sistema, que ofrecen interfaces a los recursos disponibles del sistema.

La adaptación de la calidad de servicio se puede realizar a través de dos métodos diferentes: (1) Adaptación en banda, aplicada en el contexto de una invocación remota y con un efecto directo en ésta. Un delegado QuO puede realizar esta adaptación cuando un cliente realiza una invocación remota o cuando la operación invocada termina, chequeando el estado de los contratos involucrados y eligiendo el estado en función del estado actual del sistema; (2)Adaptación fuera de banda, realizada a través de la monitorización de los objetos de condición del sistema. Cuando el estado de uno de éstos cambia, se realiza una evaluación asíncrona del estado de los contratos involucrados.

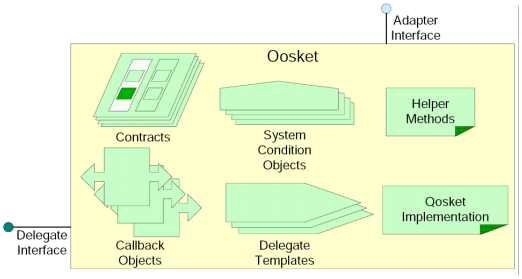


Figura 2.15: Qosket (figura tomada de [232])

Para que los comportamientos de *QoS* puedan ser encapsulados en unidades reemplazables que se puedan seleccionar, personalizar y enlazar en nuevos programas de aplicación, se han definido los Qoskets [198] (ver figura 2.15) como la unidad mínima *software*, definida por: (1) los contratos, objetos de condición y componentes *software* que encapsula; (2) definición de especificaciones parciales de comportamiento adaptativo, a través del uso de plantillas implementadas con un lenguaje de especificación de adaptación; (3) código para instanciar los componentes Qosket, de inicialización, adaptación, control, etc.; y (4) las interfaces del Qosket.

Dado que la implementación actual de Qoskets en QuO requiere que los desarrolladores de aplicación modifiquen el código manualmente para dar soporte a comportamientos dinámicos, se hace necesario el uso de tecnologías de componentes para encapsular la gestión de *QoS*, tanto dinámica como estática. El grupo de trabajo de la universidad de Washington está trabajando para extender CIAO (Component-Integrated ACE ORB), su implementación del modelo de componentes de Corba (Corba Component Model, CCM) [170] cuyos elementos se muestran en la figura 2.16 , de tal forma que soporte la implementación de Qoskets en su arquitectura en forma de componentes [232]. La forma en la que CIAO convierte en componentes un Qosket se muestra en la figura 2.17. Así, CIAO podría soportar la adaptación dinámica de *QoS* en cada par cliente/componente, siendo una línea de investigación abierta la gestión de la calidad de servicio a través de toda la aplicación y entre

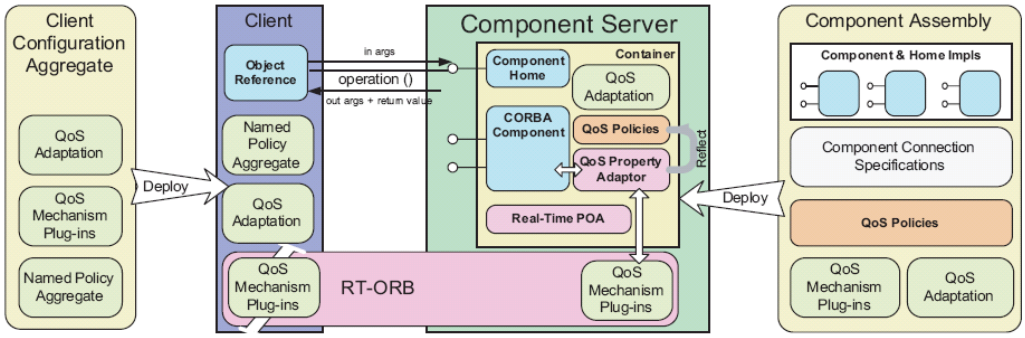


Figura 2.16: Elementos de CIAO (figura tomada de [232])

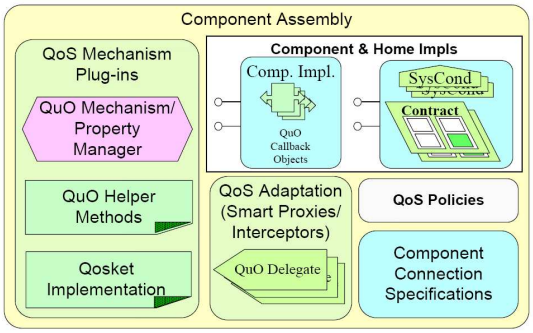


Figura 2.17: Componente Qosket usando CIAO (figura tomada de [232])

componentes, de tal forma que se pudiese modificar el comportamiento de varios componentes a la vez de forma distribuida.

Anteriormente, la tesis de Nanbor Wang [231], de la misma universidad, estudió el uso de aspectos en el modelo de componentes de Corba (Corba Component Model, CCM) [170], presentándose un diseño sobre CIAO que incorpora parámetros de tiempo real y *QoS*. Esta tesis sólo soporta, como CIAO en ese momento, la composición estática, siendo la adaptación dinámica de la *QoS* una de las líneas de investigación futuras que propone este trabajo.

2.7.7.1. Análisis

Esta propuesta se basa en el uso de componentes, que permite la reutilización de código, y aspectos para realizar una gestión de la calidad de servicio para sistemas distribuidos de tiempo real basados en RT-Corba.

Pese a ser una solución mucho más madura y especificada que el trabajo anterior, esta aproximación adolece de la misma carencia respecto a la composición de aplicaciones distribuidas: la solución está excesivamente influenciada por el paradigma cliente-servidor, carencia lógica puesto que RT-Corba está basado en esta arquitec-

tura y este trabajo trata de integrar el entorno QuO en una implementación de RT-Corba, CIAO. Además, la aproximación seguida por el entorno QuO, con contratos en los delegados, que monitorizan la calidad de servicio, se presta a una implementación de una arquitectura cliente-servidor (o cliente-componente) más que al uso de diferentes servidores (componentes) que deben trabajar en común y, de alguna manera, establecer una comunicación (sea mediante el uso de un servidor principal, sea mediante mensajes intercambiados por los propios delegados) para ofrecer una *QoS* de toda una aplicación. Así pues, la definición de la gestión dinámica de *QoS* en una aplicación que se expande en varios nodos la plantean como trabajo futuro.

Con respecto a los trabajos analizados anteriormente, es importante destacar el uso del término "contrato" relacionado con un sistema de tiempo real, en su caso a nivel de Delegado, pero que son análogos a los SLAs.

2.7.8. Otros trabajos relacionados

Tal y como se ha puesto de manifiesto en apartados anteriores, existen numerosas propuestas para dotar de flexibilidad en tiempo de ejecución a sistemas de tiempo real, tanto centralizados como distribuidos, centrándose cada uno de ellos en diferentes aspectos de los mismos. En el campo de tecnologías basadas en componentes, el trabajo de De Miguel [62, 63] propone una solución para la integración de servicios básicos para la provisión de calidad de servicio, como reserva de ancho de banda y negociación, en tecnologías Java, más concretamente en EJBs (*Enterprise Java Beans*) sobre RMI. El contenedor EJB implementa algunos algoritmos básicos de negociación sobre el número de peticiones por segundo que pueden realizar los clientes y aisla a los componentes de negocio de los servicios de reserva. Además, puede realizar adaptaciones básicas de la *QoS* basadas en la renegociación de recursos con otros componentes Este trabajo fue extendido para RTSJ por Tolosa [226], la cual define un modelo de contenedor EJB y usa los servicios proporcionados por RTSJ para mejorar la predictibilidad de los componentes de las aplicaciones. Orientado a tecnologías CORBA, el trabajo de Tatibana [64] se centra en la extensión de los tests de aceptación en un entorno de componentes distribuido como CIAO para asegurar en el momento del enlace que todas las peticiones periódicas realizadas pueden ser atendidas cumpliendo sus requisitos temporales.

En el campo de los servicios Web, el protocolo SOAP se utiliza para codificar los datos que intercambian los servicios y aplicaciones en un formato XML. El trabajo de Helander [99] se centra en realizar extensiones al protocolo SOAP para dotarlo de determinismo temporal. En concreto, propone que la serialización y deserialización de los mensajes se base en tablas predefinidas conteniendo representaciones de todos los mensajes, interfaces y campos que el servicio entiende. Otra línea de investigación en este campo es dotar de movilidad a los servicios Web [96], permitiendo su migración a otras localizaciones para mejorar la calidad de servicio ofrecida por el sistema, en un enfoque similar al trabajo del grupo de la Universidad de Texas,

analizado en el apartado 2.7.5.

Existen también aproximaciones a la flexibilidad de aplicaciones de tiempo real mediante la definición de un modelo de ejecución para agentes móviles de tiempo real [190], y así permitir un reparto dinámico de la carga en el sistema. Estos trabajos definen un agente como una entidad *software* independiente que debe realizar una tarea para el sistema pudiendo interrumpir su ejecución en un nodo, moverse a otro y retomar la ejecución en otro distinto. En un grado de abstracción alto, la propuesta analizada en el apartado 2.7.5 podría ser considerada como una aproximación a la realización de agentes de tiempo real con la salvedad de que no interrumpen su ejecución sino que se trasladan antes de la invocación.

2.7.9. Conclusiones

Existe una carencia de soluciones que apliquen los principios de la orientación a servicios a sistemas de tiempo real. Las soluciones más cercanas versan sobre sistemas con calidad de servicio, pero no son aplicables a entornos distribuidos de tiempo real, donde existen restricciones temporales tanto en la red como en los propios nodos. Las propuestas orientadas a dotar de flexibilidad sistemas de tiempo real mediante la definición de tecnologías de componentes, se basan en el concepto de cambio de modo. Estas propuestas asimilan los modos a distintos aspectos o configuraciones de un componente, pero restringen su ubicación a una localización determinada, y no proporcionan mecanismos generales de composición que permitan la existencia de varios elementos que implementen la misma funcionalidad entre los cuales elegir o realizar un reparto de carga. La flexibilidad que aportan se restringe a mecanismos de control de *QoS*, sin dotar a las aplicaciones de la posibilidad de reconfigurarse en tiempo de ejecución. Tampoco definen mecanismos para la inclusión de nuevas entidades en el sistema.

En los siguientes capítulos, se propone la aplicación del concepto de servicio a sistemas de tiempo real distribuidos como una manera de dotar a éstos de flexibilidad funcional de forma dinámica. Esta propuesta se desarrolló teniendo presentes las comunicaciones de tiempo real subyacentes, que influirán tanto en la adaptación del concepto de servicio, como en el modelo de la aplicación y en las diferentes arquitecturas propuestas.

Capítulo 3

Modelo de Aplicaciones Basadas en Servicios

En este capítulo, se introduce el modelo de sistema en el que se basará esta tesis y el modelo de aplicación de tiempo real basada en servicios. En primer lugar, se presenta de forma razonada el modelo y las restricciones impuestas al mismo. A continuación, se describe el modelo de servicio de tiempo real y se presenta el concepto de perfil de servicio, el cual se caracteriza formalmente. Además, se caracterizan las peticiones de los clientes y se propone un análisis de compatibilidad entre petición y perfil que deberá ser aplicado durante la composición de aplicaciones. Se presenta la caracterización temporal de un perfil de servicio. Lo que da paso a la presentación del modelo de aplicaciones de tiempo real basadas en servicios. Se realiza una caracterización formal tanto de las aplicaciones deseadas por los clientes como de las finales que se instanciarán en el sistema. Este capítulo se cierra con la presentación del modelo temporal de una aplicación basada en servicios.

3.1. Introducción

Los sistemas *software* distribuidos, como ya se comentó en el capítulo 1, se han vuelto más dinámicos permitiendo distribución, auto-reconfiguración, portabilidad y migración de aplicaciones. Como consecuencia, han aparecido nuevos paradigmas de desarrollo de aplicaciones basados en el uso de múltiples servicios dispersos en el entorno [103, 197], que permiten aportar mayor flexibilidad tanto en el desarrollo como en la ejecución de éstas. Concretamente, el uso del paradigma de orientación a servicios, permite crear aplicaciones de forma dinámica a partir de servicios con interfaces bien especificadas que pueden ser invocados en secuencia, tal y como se muestra en la figura 3.1.

Esta tesis se centra en la aplicación de conceptos del paradigma de orientación a servicios a sistemas de tiempo real, basándose en la premisa de que éste puede usarse de manera ventajosa también en estos entornos. Dentro de los sistemas que podrían

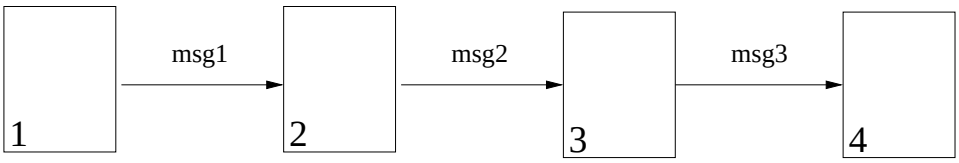


Figura 3.1: Abstracción de una aplicación

beneficiarse de esta aproximación, destacan aquellos sistemas de control distribuidos donde los diferentes controladores y filtros pueden ser compartidos y actualizados en tiempo de ejecución, y sistemas multimedia distribuidos, en los cuales los diferentes servicios se corresponderían con filtros y codificadores/decodificadores, con diferentes niveles de calidad de servicio.

Las características de las aplicaciones a las que va dirigida esta tesis son las siguientes:

- Aplicaciones distribuidas, aunque el modelo permite realizar composición centralizada en un único nodo físico.

- Flexibles, capaces de ejecutarse como parte de un entorno que cambia de forma dinámica y ajustarse a éste.

- Temporalmente predecibles, formando parte de entornos en los que se exige un comportamiento determinista en funcionalidad y temporalidad. El conjugar distribución con garantías temporales, implica que las comunicaciones deberán ser deterministas. Por otra parte, los modelos que se presentarán en esta tesis, son más adecuados para aplicaciones de tiempo real no críticas, aunque podrían ser aplicados en otros entornos.

- Calidad de servicio, donde la calidad que es capaz de ofrecer una aplicación depende de los recursos que tenga asignados.

Las aplicaciones estarán compuestas de servicios, entendiendo como servicio una entidad *software* autocontenida que proporciona una determinada funcionalidad. Cada uno de estos servicios podrá tener una o varias implementaciones coexistiendo en el sistema, y estas implementaciones concretas, a partir de ahora perfiles de servicio, pueden presentar características temporales o de calidad de servicio distintas, que generan resultados de distinta calidad. Así, una aplicación en ejecución será un grafo ordenado de perfiles de servicio concretos y sus respectivas interacciones (mensajes). Estas aplicaciones podrán ser desde una cadena de acciones a realizar sobre distintos elementos de la red, hasta una secuencia de filtros necesaria para procesar una imagen.

Las aplicaciones, en principio, serán distribuidas, es decir, los perfiles estarán dispersos en la red en diferentes nodos físicos, pudiendo varios perfiles residir en

el mismo nodo y varias aplicaciones podrán compartir el mismo perfil. La comunicación entre perfiles que residen en nodos diferentes se realizará mediante paso de mensajes, regulados por los protocolos de red subyacentes. Mientras que la comunicación entre perfiles residentes en el mismo nodo físico se realizará mediante búferes de comunicación. El dimensionamiento de estos búferes de comunicación intranodal, así como sus posibles reconfiguraciones, provocadas por reconfiguraciones de las aplicaciones es un problema NP-completo, que queda fuera del ámbito de la presente tesis.

En primer lugar, el sistema realizará una composición inicial de la aplicación. Esta composición inicial podrá ser distribuida, con los perfiles residiendo en nodos físicos diferentes o centralizada, como se verá en el siguiente capítulo. Dependiendo de las características del sistema subyacente, las aplicaciones compuestas podrán reconfigurarse en tiempo de ejecución o no. Si se permite la reconfiguración, estaremos hablando de composición dinámica y si no se permite de composición estática. La multiplicidad de perfiles permitirá al sistema elegir, de entre el conjunto de combinaciones de perfiles posibles, aquélla que más se adecúe a una métrica especificada previamente.

Así, un entorno distribuido que permita composición de aplicaciones de tiempo real deberá proporcionar:

- Comunicaciones deterministas, es decir, el protocolo de red debe asegurar tiempos de transmisión acotados.

- Flexibilidad con respecto a la configuración del sistema, que ofrezca facilidades para la instanciación y reconfiguración de tareas y mensajes, es decir, aplicaciones.

- Gestión de la información relativa a cada perfil. El conferir al sistema libertad a la hora de seleccionar los perfiles, provoca que sea necesario el conocimiento *a priori* de las características de éstos, que deberá ser facilitado por parte de los desarrolladores de perfiles de servicio. En este trabajo se realiza un reparto de responsabilidad, mientras que la arquitectura y el modelo propuestos soportan la caracterización de los perfiles y su posterior composición, los desarrolladores de perfiles deben proporcionar las características de éstos.

- Un mecanismo que realice la composición y que gestione los perfiles, controlando de manera transparente la ejecución de las tareas y la transmisión de los mensajes. Este gestor deberá estar incluido en la arquitectura del sistema, como se verá en el capítulo 4. Asimismo, también será el encargado de dar el soporte necesario para la reconfiguración de las aplicaciones, en entornos donde se permita la composición dinámica.

- Mecanismos de monitorización. El hecho de que cada perfil ofrezca una calidad de servicio determinada, expuesta en la información proporcionada por

su programador, implica que, al pertenecer a una aplicación compuesta, se establezca un contrato tácito entre partes, que deberá ser monitorizado, bien por el nodo que ofrece el servicio, bien por otra entidad del sistema. Todo esto supone una carga adicional de control que deberá ser correctamente gestionada.

Por otra parte, como se verá en siguientes apartados, con respecto a los enfoques existentes en el desarrollo de sistemas de tiempo real distribuido (ver apartado 2.2.2.2), este trabajo presenta como más adecuada para este modelo una aproximación híbrida, que aúne características de ambos enfoques, la flexibilidad y eficiencia del paradigma gobernado por eventos, y la predictibilidad y seguridad del gobernado por tiempo.

En entornos donde sea posible la composición dinámica, la multiplicidad de perfiles permitirá al sistema seleccionar, en tiempo de ejecución, de aquellas implementaciones disponibles de un servicio, la que más se ajuste a sus necesidades para una aplicación determinada. Así se permite un nivel alto de adaptabilidad para variar condiciones operacionales, mejorando el uso de los recursos del sistema [182], y, como los perfiles podrán aparecer y desaparecer en tiempo de ejecución del sistema, aportándole flexibilidad para realizar equilibrados de carga entre nodos en los cuales residan perfiles del mismo servicio, actualizaciones dinámicas de versión de servicios, o el uso de éstos como técnica de tolerancia a fallos a nivel de aplicación.

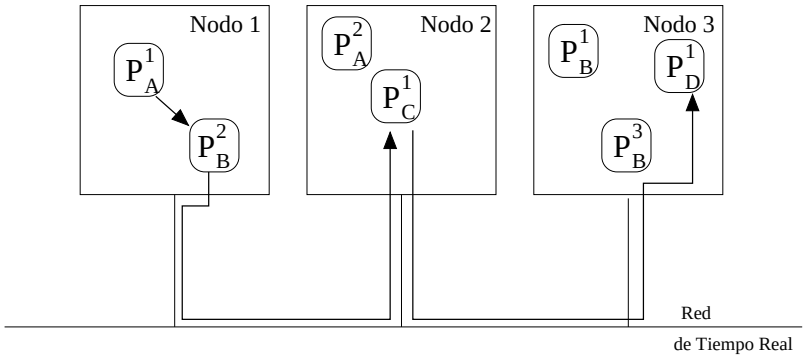


Figura 3.2: Ejemplo de aplicación distribuida

En el ejemplo de la figura 3.2, existe una aplicación, A, compuesta por el conjunto ordenado de servicios $\{S_A, S_B, S_C, S_D\}$. Cada uno de estos servicios está implementado por diferentes perfiles, localizados en diferentes nodos en la red, siendo el sistema el responsable de elegir los perfiles más adecuados en ese momento determinado para la composición de la aplicación. Se puede observar que puede existir más de un perfil residiendo en cada nodo, y que éstos pueden pertener al mismo servicio. En el ejemplo, los servicios S_A y S_B son implementados por los perfiles $\mathcal{P}_A = \{P_A^1, P_A^2\}$ y $\mathcal{P}_B = \{P_B^1, P_B^2, P_B^3\}$ respectivamente. El sistema, en este ejemplo, eligió los perfiles $\{P_A^1, P_B^2, P_C^1, P_D^1\}$, que forman la aplicación o servicio

compuesto:

$$P_A^1 \xrightarrow[N_1,N_1]{msg_1} P_B^2 \xrightarrow[N_1,N_2]{msg_2} P_C^1 \xrightarrow[N_2,N_3]{msg_3} P_D^1 \tag{3.1}$$

Si, en un momento determinado el perfil P_B^2 falla, el sistema podrá reconfigurar la aplicación para evitar un fallo general de la misma, eligiendo, por ejemplo, el perfil P_B^1. Con lo que la aplicación resultante sería:

$$P_A^1 \xrightarrow[N_1,N_3]{msg_1^*} P_B^1 \xrightarrow[N_3,N_2]{msg_2^*} P_C^1 \xrightarrow[N_2,N_3]{msg_3} P_D^1 \tag{3.2}$$

Nótese que al cambiar un perfil pueden cambiar también todos los mensajes relacionados con el mismo. El modelo desarrollado en la presente tesis, parte de un principio de transparencia de ubicación con respecto a éstos, es decir, si cambia un perfil intermedio, debe ser transparente al resto de servicios de la red, aunque interaccionen directamente con él.

La estructura de la aplicación, reflejada tanto en su propio modelo como en el de servicio, los cuales se presentarán en este capítulo, está fuertemente relacionada con el modelo de sistema distribuido en el que se basa esta tesis. En el siguiente apartado se presenta dicho modelo, así como una serie de consideraciones de índole práctica realizadas al elegirlo.

3.2. Discusión sobre el modelo del sistema

Dentro de los objetivos planteados por esta tesis se encuentra el dotar de flexibilidad en las fases de diseño y ejecución a sistemas de tiempo real distribuidos, basándose en conceptos asociados habitualmente a arquitecturas orientadas a servicios, como puede ser el propio concepto de servicio o el de aplicación como composición de servicios. Tal y como ya se expuso, se pretende que el sistema tenga la posibilidad de cambiar en tiempo de ejecución las implementaciones de los servicios, perfiles, que emplea una determinada aplicación, sin que tenga efectos en las demás aplicaciones existentes en el sistema. Este objetivo determinado obliga a decidir, antes de definir el modelo de servicio o de aplicación, la aproximación que se realizará a la hora de diseñar el sistema completo.

Cuando se diseña un sistema distribuido se han de tener en cuenta diferentes niveles de abstracción. Si nos fijamos en los niveles definidos por OSI, deberemos definir para cada nivel el modelo utilizado. Por ejemplo, los protocolos y paradigmas de comunicaciones de tiempo real expuestos en el apartado 2.3 podríamos encuadrarlos en el nivel 2 de OSI (y parte del 3, por ejemplo, AFDX al definir un protocolo de enrutamiento usando circuitos virtuales define parte de este nivel). En esta tesis, sobre la abstracción ofrecida por un protocolo de comunicaciones de tiempo real, se definirá el modelo de sistema. Este modelo se construye directamente sobre el nivel 2 de OSI. En principio, se parte de un nivel de abstracción elevado: una aplicación como un conjunto de servicios que proporcionan una determinada funcionalidad. Se

disminuye el nivel de abstracción al definir un servicio como un conjunto de perfiles que lo implementan disponibles dentro del sistema. Estos perfiles, si se disminuye aún más el nivel de abstracción, serán instanciados en cada nodo físico en forma de tareas. Aunque se utilice el mismo perfil en distintas aplicaciones o varias veces en la misma aplicación, se considerarán tareas diferentes. Así, nos acercamos a la abstracción comúnmente utilizada en el diseño de sistemas distribuidos de tiempo real: una aplicación como un conjunto de tareas que se ejecutarán en distintos procesadores que intercambian mensajes sobre una red, es decir, una transacción (véase apartado 2.2.1.1).

A la hora de diseñar un sistema de tiempo real distribuido basándose en transacciones, una aproximación utilizada es el diseño y análisis holístico, definido por Tindell en [223]. Esta aproximación tiene dos objetivos: por una parte, comprobar si se cumplen los requisitos extremo a extremo de la transacción dado un conjunto de atributos locales de las tareas involucradas en una transacción, como pueden ser el plazo, el tiempo de ejecución en el peor caso o los desplazamientos temporales; y, por otra, encontrar esos parámetros locales para asegurar que se cumplan dichos requisitos. Además, asociado al análisis de planificación holístico, pueden emplearse herramientas que evalúen la solución a la que llega el análisis y, en el caso de que no sea una solución adecuada, emplear algún procedimiento de optimización que, modificando los parámetros de las tareas, permita buscar una nueva solución que cumpla los requisitos funcionales y no funcionales especificados.

En esta tesis, se empleará un modelo holístico del sistema, pues es necesario considerar la aplicación en conjunto, teniendo en cuenta todos los servicios y perfiles utilizados y las tareas que instanciaremos en cada nodo, para poder establecer sus parámetros.

Por otra parte, ya elegido el modelo para el diseño, se ha de decidir la aproximación seguida: sistema gobernado por eventos o sistema gobernado por tiempo. En el apartado 2.2.2.2 se presenta una comparativa general de las dos aproximaciones. La aplicación de éstas a un modelo basado en transacciones genera distintos tipos de análisis, cada uno de ellos con un conjunto diferente de características:

- Aproximación gobernada por eventos:
  - Las transacciones son gobernadas por eventos.
  - Cada una de las entidades internas se activan en secuencia.
  - Los tiempos exactos de activación no son conocidos en tiempo de diseño

  Dadas estas características, en un sistema gobernado por eventos, existe una dependencia muy alta entre las acciones (tareas o mensajes) de una misma transacción, de tal forma que cualquier cambio en una acción tiene un impacto directo en las demás. Además, aparece la problemática del retraso en las activaciones o *release jitter*, véase apartado 2.2.1.2, que obliga a aplicar un análisis de planificabilidad iterativo [221], o a emplear técnicas que, usando

desplazamientos temporales, capturen las relaciones de precedencia entre las acciones de una misma transacción [173, 174]. Nótese que un cambio en los parámetros temporales de las acciones de una transacción obliga a un nuevo cálculo de su planificabilidad.

- Aproximación gobernada por tiempo:

  - Las transacciones están constituidas por entidades internas, que comparten el mismo período y que son independientes entre sí.
  - Los tiempos exactos de activación, tanto de transacciones como de las acciones que las componen, sí son conocidos en tiempo de diseño. Además, los eventos de activación son todos periódicos e independientes.

  Estas características implican, en primer lugar, la necesidad de una base de tiempos global, de tal forma que los nodos y la red estén sincronizados. Como se puede intuir, en un sistema gobernado por tiempo, se escoge el desplazamiento temporal de cada acción de tal forma que sea mayor que los tiempos de respuesta en el peor caso de sus predecesoras. Esto implica que no se produce el efecto de retraso en las activaciones y, además, que la planificación de cada recurso (nodo o red) puede ser independiente, sin necesidad de realizar un análisis de planificabilidad iterativo.

En esta tesis se sigue una aproximación híbrida. Las aplicaciones presentes en el sistema estarán gobernadas por tiempo, y las acciones involucradas tendrán instantes de activación predefinidos y siempre mayores que los tiempos de respuesta en el peor caso de las acciones precedentes. Por otra parte, será necesaria la inclusión de soporte para la comunicación gobernada por eventos, necesaria para dotar de flexibilidad al sistema completo y que éste proporcione facilidades para la inclusión de nuevas aplicaciones, nuevos servicios y nodos físicos. Así, la definición del modelo temporal de perfil de servicio permitirá que las tareas asociadas a éste puedan ser activadas por tiempo, pertenecientes a una transacción, o activadas por eventos del sistema.

Por otra parte, a la hora de realizar la planificación de una acción en un recurso, el cálculo del tiempo de respuesta en el peor caso se hará suponiendo que en el instante crítico todas las tareas están preparadas para activarse. El resultado de la aplicación de esta suposición ofrece los tiempos de respuesta en el peor caso más pesimistas, pues no tiene en cuenta ni los desplazamientos temporales, ni las relaciones de precedencia entre las acciones de una misma transacción. Sin embargo, aumenta la flexibilidad del sistema.

Cuando, en una transacción ya existente en el sistema, se modifica algún parámetro temporal de una acción intermedia, sólo será necesario realizar el análisis temporal en el recurso en el que ésta se ejecute, calcular su nuevo tiempo de respuesta en el peor caso y, si es planificable, recalcular los desplazamientos temporales de

las subsiguientes acciones. La verificación de la planificabilidad de la transacción completa se podrá reducir entonces a la verificación de que el tiempo de respuesta de la última transacción añadido a su nuevo desplamiento temporal es menor que el plazo global extremo a extremo.

Las demás transacciones del sistema sólo se verán afectadas por este cambio si alguna acción perteneciente a éstas está ubicada en el mismo recurso de la acción modificada y, además, cambia su tiempo de respuesta. Si estos tiempos de respuesta disminuyen, puede obviarse la comprobación de la planificabilidad de estas transacciones, al igual que si siguen siendo menores que el desplazamiento temporal de la acción inmediatamente posterior en su correspondiente transacción. En el caso de que los tiempos de respuesta en el peor caso aumenten, la comprobación de la planificabilidad del sistema completo se reducirá a comprobar, para aquellas transacciones afectadas, el cumplimiento de su plazo temporal de la misma manera que se realizó para la transacción que incluía la acción modificada.

Si a la hora de realizar el análisis de planificabilidad del sistema y el cálculo de los tiempos de respuesta en el peor caso, tenemos en cuenta los desplazamientos temporales, esta flexibilidad se perdería, pues un cambio en los parámetros temporales de una acción implicaría no sólo recalcular su nuevo tiempo de respuesta, sino también realizar un nuevo análisis de planificabilidad para toda la transacción y para todas aquellas transacciones con tareas afectadas por dicha modificación.

3.3. Modelo de servicio

El modelo de ejecución del sistema consiste en un conjunto de servicios. Cada servicio es una entidad *software* autocontenida, definida por su funcionalidad (respuesta de sensores y actuadores, filtros para sistemas multimedia, codificadores, decodificadores, etc.). Dicha funcionalidad puede ser implementada por diferentes entidades, *perfiles de servicio*, no necesariamente ubicadas en la misma localización. Sea $\mathcal{S}$ la representación del sistema consistente en un conjunto de n servicios:

$$\mathcal{S} = \{S_1, S_2, \ldots, S_n\} \tag{3.3}$$

Cada servicio, S_i, será implementado por un conjunto de m perfiles:

$$\mathcal{P}_i = \{P_i^1, P_i^2, \ldots, P_i^{m_i}\} \tag{3.4}$$

Cuando se desee utilizar un determinado servicio, se seleccionará (ya sea manual o automáticamente) de entre todos los perfiles de éste, aquél que más se ajuste a los requisitos establecidos para su elección por el usuario que lo demanda o por el propio sistema. Esta decisión deberá tener en cuenta el estado del sistema en ese momento. Por ejemplo, en el caso de un filtro multimedia, si distintos perfiles ofrecen diferentes calidades de servicio, el usuario podría elegir aquél que, ofreciendo una calidad de

salida determinada, minimice un determinado parámetro, por ejemplo el consumo de memoria.

La aproximación de servicio implementado por diferentes perfiles pueden ser usada para proveer de tolerancia a fallos a nivel de aplicación, en un entorno donde se permitan reconfiguraciones, es decir composición dinámica. Si una vez elegido un perfil del servicio S_i, P_i^j, el sistema detecta que, durante su ejecución, se produce un fallo, podría elegir otro perfil, P_i^k, que, implementando la misma funcionalidad, permita que el sistema sobreviva.

Si el sistema admite la inclusión en tiempo de ejecución de nuevas entidades *software*, el conjunto de perfiles asociados a un determinado servicio variará dinámicamente, permitiendo realizar la actualización dinámica de versiones. Así, si un nuevo perfil, P_i^l, llega al sistema, es posible ofrecer la posibilidad de comprobar, para las aplicaciones en ejecución hagan uso del servicio S_i, si la utilización del nuevo perfil en lugar del que están utilizando mejora sus prestaciones, de acuerdo con una determinada métrica. En ese caso, el sistema podría realizar el intercambio. Además, esta funcionalidad añade un grado más de libertad a los usuarios al proporcionar la posibilidad de elegir al proveedor que más le convenga en cada ocasión.

Asimismo, como las prestaciones de un perfil variarán en el tiempo debido a la carga que soporta, el uso de perfiles permite realizar equilibrado de carga, repartiendo la misma entre nodos desocupados.

3.3.1. Perfiles de servicio

La flexibilidad que introduce el uso de servicios y perfiles requiere información adicional que irá asociada a los distintos perfiles para permitir, así, seleccionar el más adecuado. Esta información adicional incluirá desde características generales del servicio, como puede ser su funcionalidad o métodos asociados, hasta las más específicas de cada perfil, como puede ser su calidad de salida, la cantidad de clientes que puede soportar en cada momento o la calidad de entrada que necesita. Toda esta información estará presente en el modelo de perfil de servicio que se presenta en este apartado.

El modelo de perfil de servicio incluye la siguiente información:

- Información dependiente del servicio:
  - *Funcionales*: define la funcionalidad del servicio, S_i. Indica el servicio del que se trata y los métodos o funciones que provee.
- Información dependiente del perfil:
  - *Tipo*: indica si el perfil de servicio se puede descargar al nodo donde posteriormente se ejecutará, o si es un servicio remoto. Esta diferenciación posibilita la composición estática y, aunque es una modificación del concepto de servicio, pues en el caso de servicios descargables, el proveedor

lo único que ofrece es una implementación que, posteriormente, el cliente utilizará en su propio sistema, puede ser necesario para determinadas aplicaciones que no soporten distribución.

- *De ubicación y hardware*: localización y, si es necesario, plataforma y *hardware* que lo soporta.
- *Requisitos de calidad de servicio de entrada*: requisitos de calidad de los datos recibidos, como pueden ser el tipo de datos, o el formato.
- *Parámetros de calidad de servicio de salida*: características de los datos de salida, como pueden ser su tipo o formato.
- *Políticas de calidad de servicio de salida*: posibles relaciones entre la entrada y la salida.
- *Políticas de parametrización*: se proporciona la posibilidad de degradar la calidad de salida indicada por las políticas de calidad de servicio si el cliente así lo indica. Esta degradación sobre la calidad ofrecida a un cliente determinado siempre estará sujeta a no afectar negativamente a otros clientes del servicio. Esta característica permite a los clientes llegar a un compromiso entre parámetros de calidad de salida, por ejemplo, si un cliente que requiere menor calidad de salida podría conseguirla con un menor tiempo de ejecución en el peor caso. Las interfaces ofrecidas por cada perfil posibilitarán la inclusión de parámetros de reconfiguración.
- *Comportamiento temporal*: incluye requisitos *hardware* del servicio, como puede ser el consumo de memoria. Nótese que el tiempo de ejecución en el peor caso se incluye dentro de los parámetros de calidad de salida dinámicos.

Cada perfil de servicio tendrá uno o varios puertos de entrada así como uno o varios puertos de salida. Cada puerto tendrá asociado un búfer de datos en el que el perfil de servicio escribirá o leerá los datos que necesite. La transmisión de los datos será dependiente de la aproximación que utilicemos: en el modelo distribuido, será gobernada por la propia red, mientras que en el modelo centralizado, la comunicación se realizará a través de estos búferes de comunicación entre perfiles.

Formalmente, un perfil de servicio queda completamente caracterizado por:

$$P = \{F, t, U, \vec{\theta}^{in}, \vec{\lambda}_{in}, \vec{\theta}^{out}, \vec{\lambda}_{out}, \vec{\Psi}_{out}, \vec{\zeta}_{out}\} \tag{3.5}$$

Donde,

- F es su funcionalidad.
- t el tipo de servicio.
- U su ubicación

- $\vec{\theta}^{in}$ las características de la entrada. Cada elemento es una característica que puede estar representada por un valor discreto único, un conjunto de valores discretos o un rango de valores continuos.
- $\vec{\lambda}_{in}$ conjunto de reglas que representan relaciones de dependencia entre parámetros de entrada.
- $\vec{\theta}^{out}$ las características de la salida. Este vector de características presenta la misma estructura que el de características de la entrada.
- $\vec{\lambda}_{out}$ conjunto de reglas que representan relaciones de dependencia entre parámetros de salida.
- $\vec{\Psi}_{out}$ sus políticas de calidad de servicio de salida. Éstas pueden estar representadas por conjuntos de valores discretos, o por una función. Relacionan características de la entrada con características de la salida.
- $\vec{\zeta}_{out}$ sus políticas de parametrización. Cada política de parametrización modifica las políticas de calidad de servicio de salida en función de la entrada así como las características de la entrada y salida estáticas, y sólo es aplicable una cada vez.

3.3.2. Elección de perfiles

La composición de servicios requiere, tal y como veremos en el siguiente apartado, que los perfiles usados de cada servicio sean compatibles. La elección de un determinado perfil para implementar una funcionalidad dependerá tanto de las especificaciones del cliente como de las propias características del perfil de servicio. En este apartado se analizarán en primer lugar las demandas de un determinado cliente respecto a la elección de un perfil determinado y, posteriormente cuáles son los requisitos de compatibilidad entre dos perfiles para poder componer una aplicación.

Los requisitos establecidos por el cliente se caracterizan completamente por:

$$\Upsilon_{pet} = \{F_{pet}, t_{pet}, \vec{\delta}^{in}_{pet}, \vec{\lambda}^{in}_{pet}, \vec{\delta}^{out}_{pet}, \vec{\lambda}^{out}_{pet}, \vec{\gamma}_{pet}\} \tag{3.6}$$

donde,

- F_{pet} es la funcionalidad requerida por el cliente.
- t_{pet} es el tipo de servicio deseado
- $\vec{\delta}^{in}_{pet}$ es el conjunto de características estáticas de los datos de entrada del cliente.
- $\vec{\lambda}^{in}_{pet}$ conjunto de reglas que representan relaciones de dependencia entre características de entrada.

- $\vec{\delta}_{pet}^{out}$ es el conjunto de características estáticas deseadas de la salida.

- $\vec{\lambda}_{pet}^{out}$ conjunto de reglas que representan relaciones de dependencia entre características de salida.

- $\vec{\gamma}_{pet}$ relaciona las características de la entrada deseadas con las de salida.

Tanto las características de los datos de entrada como el conjunto de características deseadas de la salida pueden estar representadas por un valor discreto único, un conjunto de valores discretos o un rango de valores continuos. En el caso de parámetros que presenten dependencias entre ellos, éstas se declaran en reglas incluídas en $\vec{\lambda}_{pet}^{in}$ y $\vec{\lambda}_{pet}^{out}$. Las funciones de dependencia entre características dinámicas de la entrada con las de la salida pueden representarse como funciones, conjuntos de valores discretos o estructuras condicionales incluídas dentro de $\vec{\gamma}_{pet}$.

El análisis de compatibilidad entre las características deseadas por el cliente y un perfil determinado se realiza de la siguiente manera:

1. Se seleccionan los perfiles de un servicio determinado que coincidan su funcionalidad y tipo con el deseado por el cliente.

2. Se estudia la compatibilidad de los datos de entrada del cliente, $\vec{\delta}_{pet}^{in}$, con las características de entrada del perfil $\vec{\theta}^{in}$. Las características de entrada válidas serán la intersección de los parámetros definidos por ambos vectores, $\vec{\Delta}_{in}$.

3. Se aplica a las características de entrada válidas para perfil y cliente, las dependencias impuestas por el cliente entre entrada y salida deseadas, $\vec{\gamma}_{pet}$, para obtener el conjunto de datos de salida posibles y se añaden aquellas características enunciadas en $\vec{\delta}_{pet}^{out}$ que no tengan relaciones de dependencia con la entrada, resultando el conjunto, $\vec{\Delta}_{in}^{pet}$.

4. Se aplica a las características de entrada válidas para perfil y cliente, las dependencias impuestas por el perfil de servicio entre entrada y salida deseadas, $\vec{\Psi}_{out}$, para obtener el conjunto de datos de salida posibles y se añaden aquellas características enunciadas en $\vec{\delta}^{out}$ que no tengan relaciones de dependencia con la entrada resultando el conjunto, Δ_{in}^{perfil}.

5. Se estudia la compatibilidad del conjunto de datos obtenidos en los pasos 3 y 4, obteniendo el conjunto de datos válidos de salida para perfil y cliente.

6. En el caso de que los pasos 2 o 5 no produzcan un resultado satisfactorio, se aplican las políticas de parametrización del perfil, ζ_{out} a Δ^{in}, Δ^{out} y Ψ_{out}, volviendo a repetir los pasos 2, 3, 4 y 5

Definición 3.1. Se definen las características de entrada válidas, $\vec{\Delta}_{in}$, como la intersección de las características de entrada deseadas por el cliente, $\vec{\delta}^{in}_{pet}$, con las ofrecidas por el perfil, $\vec{\theta}^{in}$, una vez aplicadas a cada una de ellas sus respectivas relaciones de dependencia de parámetros:

$$\vec{\Delta}_{in} = \vec{\lambda}^{in}_{pet}(\vec{\delta}^{in}_{pet}) \cap \vec{\lambda}_{in}(\vec{\theta}^{in}) \tag{3.7}$$

Definición 3.2. Se define una relación de dependencia de entrada-salida de parámetros como una aplicación del conjunto potencia de M y del conjunto potencia de N en el conjunto potencia de N $f : 2^M \times 2^N \rightarrow 2^N$ que verifica que dados $m \in M$ y $n \in N$, donde m es un subconjunto de parámetros de entrada posibles y n el conjunto de todos los parámetros de salida posibles. $f(m, n) \in 2^N$ es el subconjunto de n posible para el conjunto definido por m.

Nótese que si un parámetro de salida existente en n, no mantiene relaciones de dependencia con parámetros de entrada, permanecerá inmutable.

Definición 3.3. Se definen las características de salida válidas para el cliente dado un determinado perfil, $\vec{\Delta}^{pet}_{out}$, como la aplicación al conjunto de entradas válidos para ambos, $\vec{\Delta}_{in}$, y al conjunto de salidas válidas para el cliente de las relaciones de dependencia de entrada-salida impuestas por el cliente, restringidas a las relaciones de dependencia de la salida del cliente:

$$\vec{\Delta}^{pet}_{out} = \vec{\lambda}^{out}_{pet}\left(\vec{\gamma}_{pet}(\vec{\Delta}_{in}, \vec{\delta}^{out}_{pet})\right) \tag{3.8}$$

Definición 3.4. Se definen las características de salida válidas para el perfil dado un determinado cliente, $\vec{\Delta}^{perfil}_{out}$, como la aplicación al conjunto de entradas válidos para ambos, $\vec{\Delta}_{in}$, y al conjunto de salidas válidas para el perfil de las relaciones de dependencia de entrada-salida impuestas por el perfil, es decir, las políticas de calidad de servicio, restringidas a las relaciones de dependencia de la salida del perfil:

$$\vec{\Delta}^{perfil}_{out} = \vec{\lambda}_{out}\left(\vec{\Psi}_{out}(\vec{\Delta}_{in}, \vec{\theta}^{out})\right) \tag{3.9}$$

Definición 3.5. Se definen las características de salida válidas, $\vec{\Delta}_{out}$, como la intersección de las características de salida válidas para el cliente dado un determinado perfil y las características de salida válidas para el perfil dado un determinado cliente, aplicando las restricciones de ambos para la salida:

$$\vec{\Delta}_{out} = \vec{\lambda}_{out}\left(\vec{\lambda}^{pet}_{out}\left(\vec{\Delta}^{perfil}_{out} \cap \vec{\Delta}^{pet}_{out}\right)\right) \tag{3.10}$$

Definición 3.6. Se define la aplicación de una parametrización a un determinado perfil como:

$$\zeta^{i}_{out}\,(P) \tag{3.11}$$

donde los parámetros que se modifican son:

- Los requisitos de entrada y sus dependencias: $\zeta^i_{out}(\vec{\theta}^{in})$ y $\zeta^i_{out}(\vec{\lambda}_{in})$.
- Los requisitos de salida y sus dependencias: $\zeta^i_{out}(\vec{\theta}^{out})$ y $\zeta^i_{out}(\vec{\lambda}_{out})$.
- Las políticas de calidad de servicio de salida $\zeta^i_{out}(\vec{\Psi}_{out})$.

3.3.2.1. Ejemplo de estudio de compatibilidad de perfil

Un cliente necesita un servicio remoto que proporcione la funcionalidad de un filtro conversor de imágenes *jpeg* y *png*. Los requisitos impuestos por el cliente son:

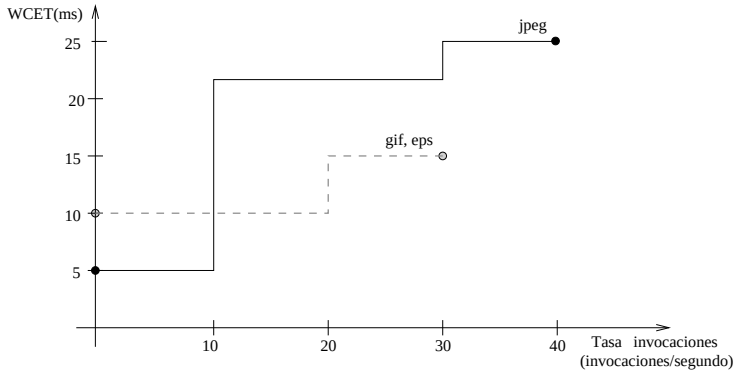
- Para la entrada:
  - que admita como formato de entrada *jpeg* o *png*, siendo indispensable que admita *jpeg*.
  - que la resolución en pixels esté incluída dentro de las siguientes escalas $[640 \times 480, 1024 \times 768, 1536 \times 2048]$, y
  - que la tasa de invocaciones por segundo que desea realizar el cliente va a estar en el intervalo $[10, 40]$.
- Por otra parte, impone como restricción a la entrada:
  - Si el formato es *jpeg*, la tasa de invocaciones nunca será menor de $20\ invocaciones/segundo$.
- Con respecto a la salida:
  - que los formatos que soporte sean *jpeg*, *gif* y *eps*.
  - Que el tiempo máximo de ejecución en el peor caso sea como mucho $20ms$.
- Y las dependencias de la salida con respecto a la entrada son:
  - Si el formato de entrada es *jpeg* y el número de invocaciones superior a $30\ invocaciones/segundo$, el tiempo máximo de ejecución sea como mucho $15ms$.

Uno de los perfiles encontrados que cumple dicha funcionalidad presenta las siguientes características:

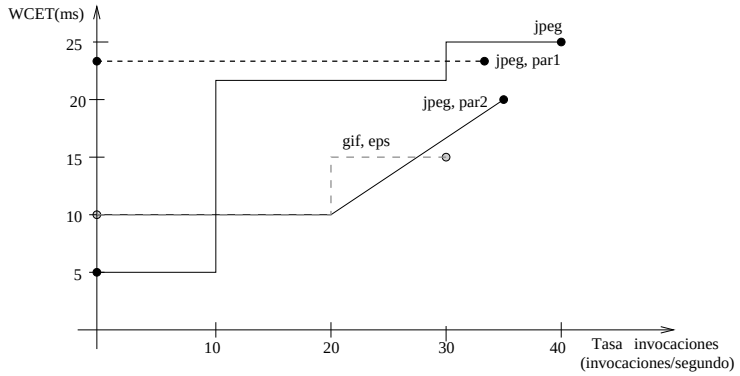
- Para la entrada:
  - Los formatos que admite son *jpeg*, *bmp* y *tiff*.
  - La resolución en pixeles admitida son $[640 \times 480, 1024 \times 768]$.

- No existen dependencias entre parámetros de entrada.
- La salida presenta las siguientes características:
  - Los formatos que soporta son *jpeg*, *bmp*, *gif* y *eps*.
- Las políticas de calidad de salida son:
  - El número de invocaciones por segundo con respecto al tiempo de ejecución en el peor caso en función del tipo de datos de la salida depende de las funciones representadas en la figura 3.3.1.

Las políticas de parametrización modifican las políticas de calidad de salida, tal y como se puede observar en la figura 3.3.2.



3.3.1: Políticas de calidad de salida $\vec{\Psi}_{out}$



3.3.2: Políticas de calidad de salida $\vec{\Psi}_{out}$ modificadas por las políticas de parametrización

Figura 3.3: Características del perfil del ejemplo del apartado 3.3.2.1

En primer lugar se haya la intersección del conjunto de datos de entrada posibles para el cliente con los del perfil, $\vec{\Delta}_{in}$:

- Formato de entrada: *jpeg*.
- Resolución en pixels: $[640 \times 480, 1024 \times 768]$
- Tasa de invocaciones por segundo $[20, 40]$.

Se aplica a $\vec{\Delta}_{in}$ y $\vec{\delta}_{pet}^{out}$ las relaciones de dependencia impuestas por el cliente, $\vec{\gamma}_{pet}$:

- Formato de salida: *jpeg*, *gif* y *eps*.
- Tiempo máximo de ejecución para salida en formato *gif* y *eps*: como mucho $20ms$.
- Tiempo máximo de ejecución para salida en formato *jpeg* con tasa de invocaciones en el intervalor $[20, 30)$: como mucho $20ms$.
- Tiempo máximo de ejecución para salida en formato *jpeg* con tasa de invocaciones en el intervalo $[30, 40]$: como mucho $15ms$.

El conjunto resultante $\vec{\Delta}_{out}^{pet}$ está representado en la figura 3.4.

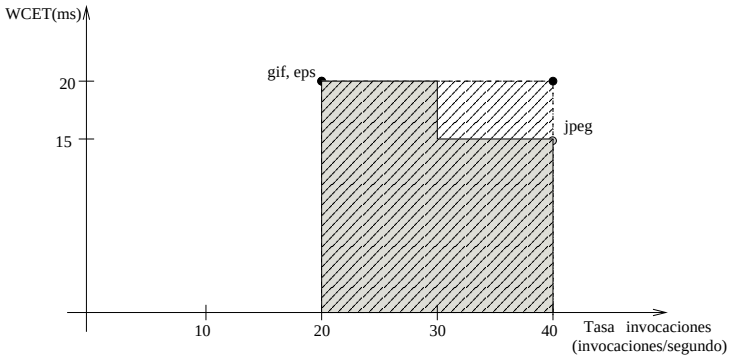


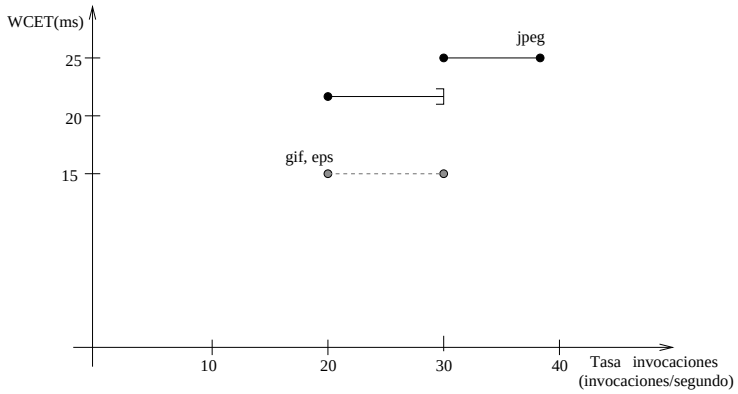
Figura 3.4: Conjunto resultante $\vec{\Delta}_{out}^{pet}$ para el ejemplo del apartado 3.3.2.1

Se aplica a $\vec{\Delta}_{in}$ y $\vec{\theta}^{out}$ las políticas de calidad de servicio impuestas por el perfil, $\vec{\Psi}_{out}$:

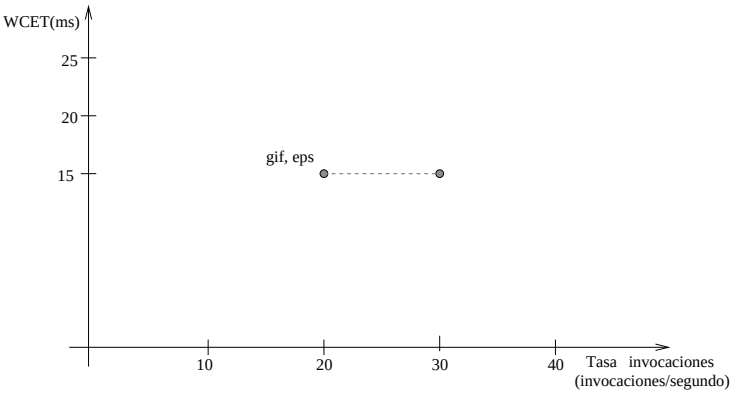
- Formato de salida: *jpeg*, *gif* y *eps*.
- Tiempo máximo de ejecución para salida en formato *gif* y *eps* para tasa de invocación en el intervalo (20,30]: $15ms$

- Tiempo máximo de ejecución para salida en formato *jpeg* con tasa de invocaciones en el intervalor $[20, 30)$: $20ms$.
- Tiempo máximo de ejecución para salida en formato *jpeg* con tasa de invocaciones en el intervalo $[30, 40]$: $25ms$.

El conjunto resultante $\vec{\Delta}_{out}^{perfil}$ está representado en la figura 3.5.1. La intersección de ambos conjuntos está representada en la figura 3.5.2, se puede observar que al aplicar $\vec{\lambda}_{out}^{pet}$ a dicha intersección $\vec{\Delta}_{out} = \emptyset$, pues no cumple las restricciones impuestas por el cliente, que exigía que debía soportar *jpeg*.



3.5.1: Conjunto resultante $\vec{\Delta}_{out}^{perfil}$

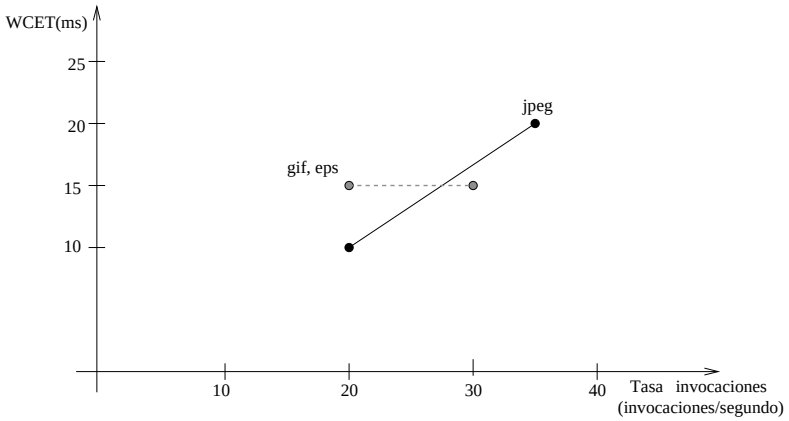


3.5.2: $\vec{\Delta}_{out}^{pet} \cap \vec{\Delta}_{out}^{perfil}$

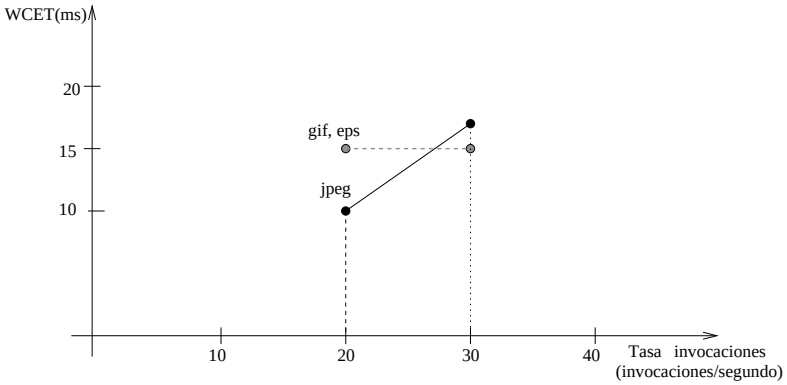
Figura 3.5: Cálculo de $\vec{\Delta}_{out}^{perfil}$ y $\vec{\Delta}_{out}^{pet} \cap \vec{\Delta}_{out}^{perfil}$ del ejemplo del apartado 3.3.2.1

En este punto se aplican las políticas de parametrización del perfil, para ver si alguna cumple los requisitos. En el caso de la política de parametrización ζ_{out}^{1} se puede observar que no va a cumplirlos, pues para un formato de entrada *jpeg* el tiempo

de ejecución en el peor caso es constante de valor $21{,}5$, sin embargo la política de parametrización ζ^2_{out} puede cumplir los requisitos. Como dicha política únicamente modifica la política de calidad de servicio, se debe volver a calcular $\vec{\Delta}^{perfil}_{out}$, véase figura 3.6.1. Cuya intersección con $\vec{\Delta}^{perfil}_{out}$ se puede ver en la figura 3.6.2



3.6.1: Conjunto resultante $\vec{\Delta}^{perfil}_{out}$



3.6.2: $\vec{\Delta}^{pet}_{out} \cap \vec{\Delta}^{perfil}_{out}$

Figura 3.6: Cálculo de $\vec{\Delta}^{perfil}_{out}$ y $\vec{\Delta}^{pet}_{out} \cap \vec{\Delta}^{perfil}_{out}$ una vez aplicada ζ^2_{out} del ejemplo del apartado 3.3.2.1

Así, el conjunto de datos de salida válidos para ambos, $\vec{\Delta}_{out}$, será:

- Formatos de salida válidos: *gif*, *eps* y *jpeg*
- Tiempo máximo de ejecución para salida en formato *gif* y *eps* para tasa de invocación en el intervalo $(20, 30]$: $15ms$.

- Tiempo máximo de ejecución para salida en formato *jpeg* para tasa de invocación en el intervalo $(20, 30]$ dado por la función $f(t) = \frac{2}{3}(t - 20) + 10$

Dentro de estos datos de salida, las relaciones de dependencia condicional entre entrada y salida si se establece una relación obligan a ambas partes, cliente y perfil, a cumplirlas.

3.3.3. Caracterización temporal de un perfil de servicio

Para simplificar la caracterización temporal de un perfil de servicio, en este apartado se considera que tiene asociado únicamente un método público. Cada una de las invocaciones a un perfil se materializa en una única tarea. Para cada una de las parametrizaciones de los perfiles que afectan al tiempo de ejecución en el peor caso se considerará una tarea diferente. El número de tareas vinculadas a un perfil de servicio está acotado por el número máximo de clientes que éste soporta y por la tasa de invocaciones de cada cliente. Las tareas en el sistema son periódicas, con período T_i^t, o esporádicas, con tiempo mínimo de activación entre tareas $T_{min,i}^t$. Para caracterizar las tareas de ambos tipos nos basaremos en el modelo comúnmente empleado de caracterización de tareas(ver apartado 2.1.1). Así, se caracterizará cada invocación periódica del perfil P_i como:

$$\tau_{i,j}^t = (C_{i,j}^t, D_{i,j}^t, T_{i,j}^t, \phi_{i,j}^t, p_{i,j}^t) \tag{3.12}$$

y cada invocación esporádica como:

$$\sigma_{i,j} = (C_{i,j}^t, D_{i,j}^t, T_{min,i,j}^t, p_{i,j}^t) \tag{3.13}$$

Por otra parte, la comunicación con las tareas se realiza a través del intercambio de mensajes, los cuales son leídos al principio de su ejecución y generados al final. La activación de una tarea puede producirse debido a la recepción de un mensaje en uno de sus búferes de entrada, o al vencimiento de su período.

Se clasifican las tareas en cuatro grupos dependiendo de su interacción con el entorno. Una tarea que necesita datos es un *consumidor*; una tarea que genera datos es un *productor*; una tarea puede ser ambos a la vez, es decir, un *consumidor/productor*; mientras que las tareas que no intercambian mensajes, se denominarán *independientes*. Las tareas que pertenecen a los primeros tres grupos, son generalmente llamadas *interactivas* y sus interacciones se soportan a través del paso de mensajes a través de la red, en el caso del modelo distribuido, o a través de los búferes, en el caso del modelo monoprocesador.

En el modelo distribuido, no hay flujo de control entre tareas y mensajes. Su interfaz está basada en búferes de datos los cuales son leídos o escritos por las tareas. Los mensajes son considerados entidades independientes, cuya transmisión será gobernada por la red. Cada uno de los mensajes tendrá un tiempo de transmisión en el peor caso, C_i^m, un plazo relativo, D_i^m, un período T_i^m y un desplazamiento ϕ_i^m.

En el caso de ser un mensaje correspondiente a una tarea esporádica, el período se corresponderá con el tiempo mínimo de activación entre tareas $T_{min,i}$ de la tarea que lo genera.

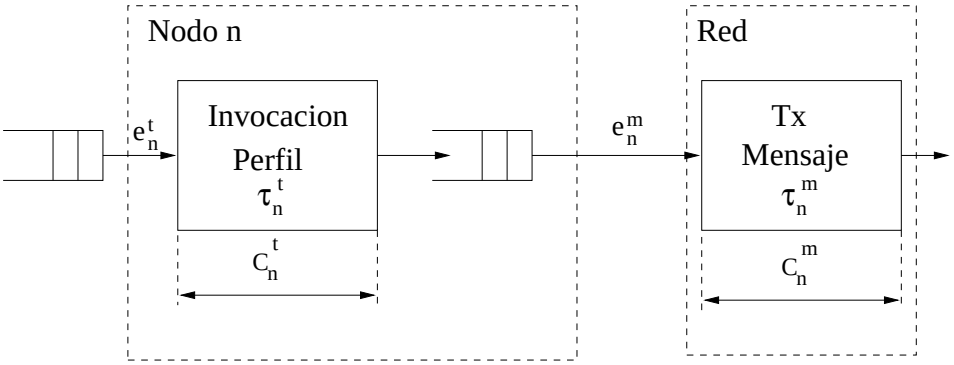


Figura 3.7: Caso general invocación

El caso más general se presenta en la figura 3.7, donde e_n^t se corresponde a un evento que activa la realización de una tarea y e_n^m a un evento que activa la transmisión de un mensaje. En el caso de la figura, el evento e_n^t activa la tarea τ_n. Este evento puede ser la recepción de un mensaje o el vencimiento de un período. La tarea τ_n^t, con tiempo de ejecución en el peor caso C_n^t, se ejecuta y escribe en su búfer de salida el resultado. Posteriormente, el evento e_n^m activa la tarea de transmisión del mensaje resultante, τ_n^m, que llegará a su destino después de un tiempo de transmisión en el peor caso de C_n^m.

La elección de los parámetros temporales de τ_n^m está relacionada íntimamente con los parámetros de τ_n^t. Si τ_n^t es una tarea periódica, activada por el vencimiento de su período y caracterizada como $\tau_n^t = (C_n^t, D_n^t, T_n^t, \phi_n^t, p_n^t)$, entonces, $\phi_n^m \geq \phi_n^t + J_n^t + R_n^t$, siendo J_n^t el retraso en la activación de la tarea τ_n^t y R_n^t su tiempo de respuesta. En el caso de que τ_n^t sea una tarea activada por la recepción de un mensaje, la elección de los parámetros de τ_n^m dependerán de los de la tarea periódica que active la transmisión de ese mensaje, τ_{n-1}^m, con lo que $\phi_n^m \geq \phi_{n-1}^m + J_{n-1}^m + R_{n-1}^m + J_n^t + R_n^t$.

3.4. Aplicaciones basadas en servicios

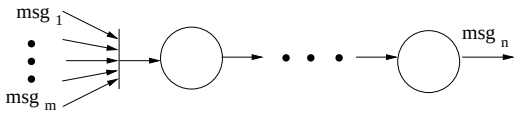
El modelo de aplicación utilizado en esta tesis se basa en la composición de servicios heterogéneos dispersos en una red de tiempo real. Los servicios se invocarán para cada aplicación en una determinada secuencia y se comunicarán entre ellos a través de mensajes. Una aplicación en ejecución quedará definida por los perfiles seleccionados para cada servicio y los mensajes intercambiados entre ellos.

En el siguiente apartado, se presenta la clasificación de las posibles aplicaciones atendiendo a su estructura y características temporales.

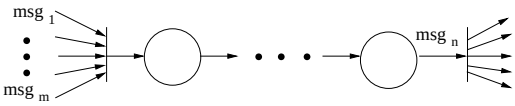
3.4.1. Clasificación de las aplicaciones

3.8.1: Aplicación uno-a-uno

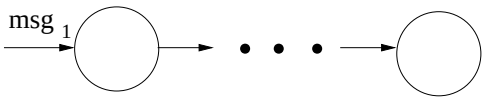
3.8.2: Aplicación uno-a-muchos



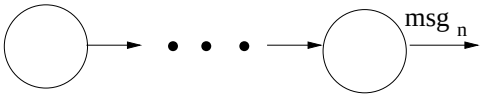
3.8.3: Aplicación muchos-a-uno



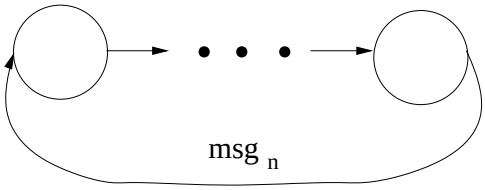
3.8.4: Aplicación muchos-a-muchos



3.8.5: Aplicación a petición de un cliente



3.8.6: Aplicación gobernada por el sistema



3.8.7: Aplicación en bucle

Figura 3.8: Clasificación aplicaciones

Se pueden diferenciar según su estructura diferentes tipos de aplicaciones (véase figura 3.8):

- *Uno-a-uno*: este tipo de aplicación se ejecuta cuando uno de los nodos del sistema recibe un dato, éste se procesa a través de todos los servicios de la aplicación y se entrega el resultado a un único nodo del sistema. Un ejemplo de este tipo de aplicaciones pueden ser las cadenas de procesado multimedia, donde se recibe un archivo multimedia que debe ser procesado por un conjunto de filtros y entregado a un único cliente.
- *Uno-a-muchos*: generalización del caso anterior, donde los datos finales son entregados a múltiples receptores. Puede ser el caso de sistemas multimedia *multicast*.

- *Muchos-a-uno*: en este tipo de aplicación son necesarios datos de múltiples emisores, que deberán ser procesados para proporcionar una determinada funcionalidad. Por ejemplo, un sistema de control que necesita datos de múltiples sensores para establecer el estado del sistema y tomar las decisiones adecuadas respecto al mismo.

- *Muchos-a-muchos*: generalización del caso anterior, donde la información, resultado de la ejecución de la aplicación, se necesita en múltiples ubicaciones del sistema.

- *A petición de un cliente*: su inicio lo marca la recepción de una orden de activación de un cliente y no hay datos de salida de la aplicación, tan sólo una serie de órdenes a ejecutar. Puede ser el caso de aplicaciones de control rutinario en un sistema invocadas por un usuario humano.

- *Gobernadas por el sistema*: Por ejemplo, aplicaciones de control automáticas iniciadas por el sistema, que pueden o no, generar datos de salida.

- *En bucle*: en este caso, la aplicación se realimenta, siendo su entrada su propia salida en la invocación anterior. Una aplicación ejemplo, podría ser un bucle de control de un sistema.

Atendiendo a sus características temporales:

- *Periódicas*: se ejecutan cada cierto tiempo especificado previamente.

- *Esporádicas*: se ejecutan ante un evento en el sistema, por ejemplo la recepción de un mensaje o el cambio en un sensor externo que lo genera.

3.4.2. Modelo funcional

En esta tesis se distingue entre el concepto de *aplicación deseada*, aquella dada por los requisitos de un cliente, y el de *aplicación en ejecución*.

Una aplicación deseada, $\mathcal{A}_D$, consiste en un grafo acíclico de servicios principal, un conjunto de subgrafos acíclicos opcionales con sus respectivas relaciones conmutativas con respecto al grafo principal y un conjunto de requisitos de calidad de servicio impuestos por el usuario, tal y como se muestra en la figura 3.9 que representa gráficamente la estructura de una aplicación deseada y su especificación.

La salida de un servicio será la entrada del siguiente en el grafo, estableciéndose así una relación de precedencia, ya que un servicio no puede comenzar hasta que termine el anterior. Cuando un servicio genera un mensaje *multicast*, éste generará ramas paralelas que se ejecutarán concurrentemente, en la figura 3.9 este hecho se representa por múltiples arcos procedentes del mismo vértice. Asimismo, un servicio puede necesitar datos de múltiples servicios precedentes antes de iniciar su

ejecución, en la figura 3.9 aparece representado a través de múltiples arcos que se unen en el mismo punto. Ramas o servicios concretos del grafo principal se pueden substituir por los subgrafos acíclicos proporcionados en la especificación, ya sea para proporcionar una calidad de servicio menor, permitiendo la reconfiguración *on-line* de la aplicación, ya sea por la imposibilidad de componer la aplicación según las especificaciones dadas. Por simplicidad se considera que se reemplazan o servicios, ramas o conjuntos enteros del grafo, entendiendo por conjunto entero, aquél que tiene un único nodo de entrada y un único nodo de salida. En la figura 3.9 se representan los subgrafos opcionales y el subconjunto del que son equivalentes mediante cajas conectadas por una flecha a la que está asociada su respectiva condición de reemplazo.

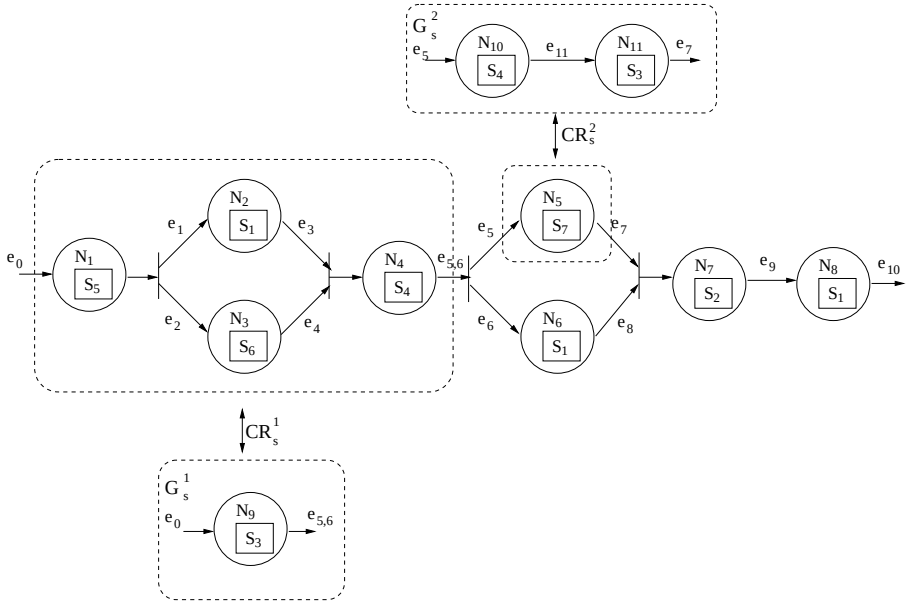


Figura 3.9: Especificación de una aplicación deseada

Definición 3.7. Se define la función $S : N \rightarrow \mathcal{S}$ que verifica para todo $n \in N$ que $S(n) \in \mathcal{S}$. Donde N es el conjunto de nodos de un grafo de servicios y $\mathcal{S}$ el conjunto de servicios existentes en el sistema.

Es decir, todo nodo de un grafo de servicios hace referencia a un servicio del sistema. En la figura 3.9, todos los nodos del grafo hacen referencia a servicios y distintos nodos de un grafo pueden hacer referencia al mismo servicio, como puede ser el caso de los nodos N_2 y N_3.

Un grafo de servicios evoluciona a través de transiciones de un nodo a otro, que representan relaciones de precedencia entre un nodo origen y otro destino alcanzable directamente a partir de él. Estas transiciones llevan aparejada la transmisión de un mensaje.

Definición 3.8. Se define una *transición* de un grafo de servicios como la terna $(n, m, n') \in N \times M \times N = E$. Donde $n \xrightarrow{m} n'$, y m representa el mensaje que debe ser intercambiado para evolucionar de n a n'.

Definición 3.9. Un *grafo de servicios* se define como $G = \langle N, E \rangle$, $N = \{n_i \mid S(n_i) = s_j\}$, $E = \{e_k \mid e_k = (n_i, m_{i,j}, n_j)\}$, donde los nodos del grafo contienen los servicios, y las aristas representan las relaciones de precedencia existentes entre ellos.

Así, un grafo de servicios es un conjunto de nodos que hacen referencia a servicios del sistema y de aristas, que representan relaciones de precedencia entre ellos.

Definición 3.10. Las *condiciones de reemplazo* de un subgrafo opcional, dado un grafo principal $G_p = \langle N, E \rangle$, se definen como $CR_s^i = \{(n_i, n_j, CS_{ree}) \mid n_i, n_j \in N, i \prec j\}$, donde n_i y n_j indican los nodos de inicio y fin del conjunto entero del grafo principal que se puede substituir y CS_{ree} las condiciones de reemplazo o implicaciones funcionales y/o en la calidad de servicio que tiene realizar dicho cambio.

Las condiciones de reemplazo pueden indicar que el conjunto entero y el subgrafo opcional son equivalentes; referirse a la calidad de salida, donde el uso de un subgrafo opcional en vez del conjunto entero a substituir en el grafo implique un degradamiento en la calidad de salida; o a implicaciones funcionales. Obviamente, el sistema deberá tener en cuenta a la hora de componer y reconfigurar las aplicaciones las condiciones de reemplazo de cada subgrafo, los requisitos funcionales y de calidad de servicio de cada aplicación y las condiciones globales del sistema.

Dentro de los grafos de servicios pueden especificarse estructuras condicionales de forma explícita e implícita. Explícitamente mediante el uso de etiquetas en las transiciones e implícitamente mediante el uso de los subgrafos de reemplazo. Dado que el modelo permite al sistema reconfigurar las aplicaciones dinámicamente, los subgrafos de reemplazo pueden usarse de forma análoga a estructuras condicionales del tipo `if-then-else` o `switch` donde, si se dan ciertas condiciones especificadas en CR_i, se pueda decidir dinámicamente el camino a seguir por el proceso. Nótese que ambas soluciones son equivalentes. Por otra parte, el modelo expuesto en este capítulo se abstrae de realizar consideraciones sobre la información o los datos sobre los que se evalúan las condiciones.

Definición 3.11. Una *aplicación deseada* vendrá definida por $\mathcal{A}_D = (G_p, \{G_s^i, CR_s^i\}, CS_{req})$. Siendo $G_p = \langle N, E \rangle$, el grafo principal, $\{G_s^i, CR_s^i\}$ el conjunto de los subgrafos opcionales y sus condiciones de reemplazo respectivamente, y CS_r el conjunto de requisitos de calidad de servicio impuestos por el usuario.

El sistema eligirá de acuerdo a unos determinados parámetros y dentro de la libertad que le permite la especificación de la aplicación deseada, el conjunto de

perfiles que compondrán la aplicación en ejecución. La aplicación en ejecución deberá tener asociada información acerca de las condiciones de calidad de servicio específicas en cada momento.

Definición 3.12. Se define la función $P : V \rightarrow \mathcal{P}$ que verifica para todo $v \in V$ que $P(v) \in \mathcal{P}$. Donde V es el conjunto de nodos de un grafo de perfiles y $\mathcal{P}$ el conjunto de perfiles existentes en el sistema.

Definición 3.13. Se define una *transición* de un grafo de perfiles como la terna $(v, m, v') \in V \times \mathcal{M} \times V = \mathcal{E}$. Donde $v \xrightarrow{m} v'$, y m representa el mensaje que debe ser intercambiado para evolucionar de v a v'.

Definición 3.14. Un grafo de perfiles se define como $G_e = \langle V, \mathcal{E} \rangle$, donde $V = \{v_k \mid P(v_k) = P_j^k\}$ y $\mathcal{E} = \{e_k \mid e_k = (v_j, m_{j,l}, v_l)\}$

Definición 3.15. Dado un sistema que contiene un conjunto de perfiles $\mathcal{P}$, se define una aplicación en ejecución como $\mathcal{A}_e = (G_e, CS_e)$, donde $G_e = \langle V, \mathcal{M} \rangle$ representa el grafo acíclico formado por los perfiles y los mensajes que intercambian y CS_e las condiciones de calidad de servicio en ejecución en cada momento.

En la figura 3.10 se representa gráficamente la estructura del proceso de una aplicación en ejecución. Puede observarse que, si bien los nodos de la aplicación son distintos, en una aplicación pueden usarse perfiles distintos del mismo servicio, como en los casos de los vértices V_4 y V_6, o el mismo perfil que recibe diferentes datos de entrada, como en el caso de los vértices V_1 y V_3.

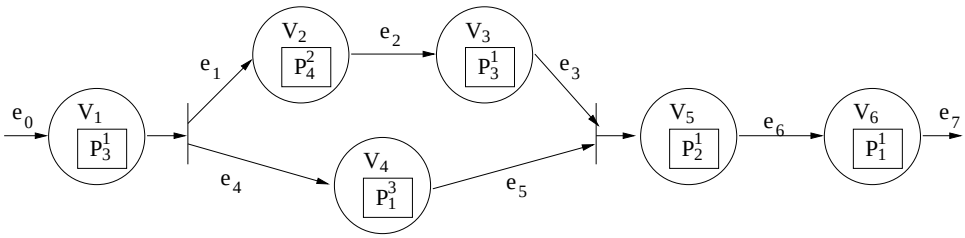


Figura 3.10: Estructura del grafo de una aplicación en ejecución

3.4.3. Modelo temporal

En este apartado se presenta el modelo temporal de una aplicación en ejecución. Se considera un sistema distribuido ejecutando aplicaciones, posiblemente periódicas, cuya ejecución puede extenderse a través de diferentes nodos en la red. Cada aplicación está compuesta por un conjunto de perfiles de servicio que pueden residir en nodos diferentes y que son invocados en una determinada secuencia.

Tal y como se expuso en el apartado 3.3.3, cada perfil de servicio se materializa en una tarea, cada una de ellas con un plazo de ejecución en el peor caso C_i^t, y

un plazo relativo D_i^t. Por otra parte, las tareas se comunican unas con otras a través del envío de mensajes, en este apartado se asume que el momento de envío de estos mensajes lo determinará el sistema, de forma autónoma. Asimismo, el tiempo de transmisión en el peor caso de estos mensajes y su plazo relativo son fijados previamente por el sistema.

En este modelo se considerarán las aplicaciones periódicas, compuestas por un conjunto de perfiles que se ejecutan según un determinado grafo de ejecución. Las ramas paralelas, debidas a la transmisión de mensajes multicast, de este grafo de ejecución se ejecutarán concurrentemente. Cada entidad, mensaje o tarea, del grafo de ejecución tendrá el mismo período que la aplicación a la que pertenece y las aplicaciones, tal y como se presentó en el apartado 3.2, son gobernadas por tiempo.

Se aplicará parte del modelo desarrollado en su tesis por Mario Calha [47, 46] para la planificación holística de sistemas distribuidos flexibles gobernados por tiempo que se explica en el siguiente apartado.

3.4.3.1. Planificación holística usando flujos de datos

En la tesis de Calha [47] se modela la interacción entre las tareas como un flujo de datos siguiendo el modelo de tareas productor-consumidor. Los datos fluyen de un productor a uno o más consumidores, que, a su vez, pueden ser productores de otros datos, especificándose sólo los posibles conjuntos de mensajes a transmitir, permaneciendo encapsulados los aspectos computacionales de las tareas. La planificación holística se plantea como una planificación en dos fases, en primer lugar se determinan los parámetros, según restricciones impuestas por la interacción entre tareas y el uso de recursos y, posteriormente, se realiza la planificación en cada nodo y en la propia red. La tesis de Calha se centra en la primera de estas dos fases. Dentro de la determinación de parámetros distingue los siguientes escenarios:

- *Centrado en la red*: para sistemas donde el recurso escaso es la red. En esta aproximación, los mensajes son los que imponen restricciones a las tareas, realizándose la especificación completa de los mensajes previamente a la planificación de las tareas.

- *Centrado en el nodo*: en esta aproximación, las restricciones se encuentran en el nodo.

- *Superpuesto*: en sistemas donde la ejecución de una tarea depende directamente del tiempo de transmisión del mensaje que la inicia, es decir, que el momento de inicio de una tarea depende del momento de inicio del mensaje en cuestión. Esta aproximación es la adecuada para el modelo de aplicación que se desarrolla en la presente tesis, debido a las relaciones de precedencia existentes entre las tareas.

3.4.3.2. Planificación holística superpuesta de mensajes y tareas

En su modelo [1], se realizan las siguientes suposiciones:

- Las tareas son periódicas e interactivas. El flujo de datos también será periódico con período T_{DS}.
- Las tareas tienen asociado un tiempo de computación acotado por un valor menor o igual a su período.
- Ninguna tarea puede suspenderse a sí misma.
- Las tareas pueden sufrir apropiación.
- Las tareas se inician al principio de cada período.
- Los mensajes son consumidos al principio de la ejecución de la tarea y producidos al terminar su ejecución.
- Las tareas pueden tener relaciones de precedencia.
- Los períodos y sus fases iniciales tienen que ser múltiplos de un valor predefinido, la duración del ciclo elemental, T_{EC}. Este modelo se centra en sistemas donde el inicio de cada ciclo está marcado por la recepción de un mensaje de disparo y dentro de cada ciclo no pueden establecerse relaciones de precedencia ni control entre mensajes y tareas. Así, el modelo considera, a efectos de planificación, simultáneas las acciones que ocurren en el mismo ciclo elemental.
- Del anterior punto se colige que dos entidades que presenten relaciones de precedencia deben ocurrir en diferentes ciclos elementales.

La determinación de los parámetros dentro de esta aproximación, la realiza en función del tipo de tarea: *productora*, *consumidora* o *productora-consumidora*.

Tareas productoras Se considera una tarea que produce un mensaje, tal y como se muestra en la figura 3.11. Los parámetros previamente especificados son el tiempo de ejecución en el peor caso de la tarea, C_t, y el tiempo de transmisión en el peor caso del mensaje, C_{msgP}.

Se impone la restricción de que los períodos de las tareas y sus plazos relativos tengan el mismo valor, $T_t = T_{msgP} = D_t = D_{msgP} = T_{DS}$. Así, el único parámetro que queda por determinar es el desplazamiento del mensaje respecto al de la tarea que lo genera, como se realiza la suposición de que el mensaje es enviado al término de la tarea, su valor será:

[1] Ver capítulo 4: *Information flow and Holistic Scheduling* de [47]

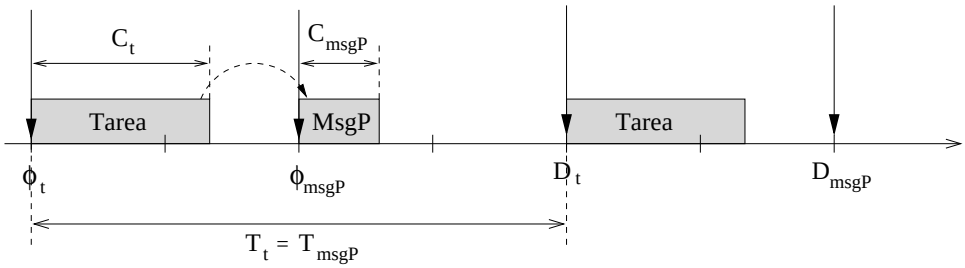


Figura 3.11: Tarea productora

$$\phi_{msgP} = \phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{3.14}$$

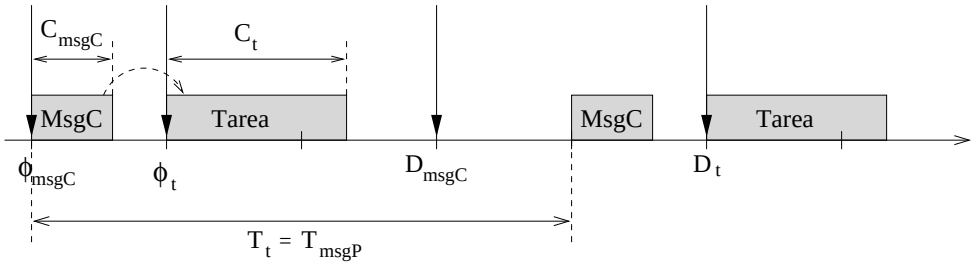


Figura 3.12: Tarea consumidora

Tareas consumidoras Se considera una tarea, véase figura 3.12, que consume un mensaje, *MsgC*. Las restricciones con respecto a los parámetros para esta tarea consumidora serán:

$$T_t = T_{msgC} = T_{DS} \tag{3.15}$$

$$\phi_t = \phi_{msgC} + \left\lceil \frac{C_{msgC}}{T_{EC}} \right\rceil T_{EC} \tag{3.16}$$

$$D_{msgC} = T_{DS} - \left\lceil \frac{C_{msgC}}{T_{EC}} \right\rceil T_{EC} \tag{3.17}$$

$$D_t = T_{DS} \tag{3.18}$$

Tareas productoras-consumidoras La figura 3.13 muestra una tarea productora-consumidora, que consume un mensaje *MsgC* y produce al llegar a término un mensaje *MsgP*. Las restricciones generales para esta tarea vendrán dadas por:

$$T_t = T_{msgC} = T_{msgP} = T_{DS} \tag{3.19}$$

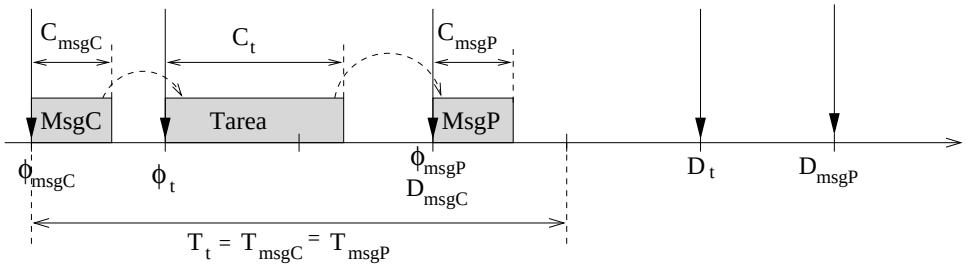


Figura 3.13: Tarea productora-consumidora

$$\phi_t = \phi_{msgC} + \left\lceil \frac{C_{msgC}}{T_{EC}} \right\rceil T_{EC} \tag{3.20}$$

$$\phi_{msgP} = \phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{3.21}$$

$$D_{msgC} = T_{DS} - \left\lceil \frac{C_{msgC}}{T_{EC}} \right\rceil T_{EC} \tag{3.22}$$

$$D_t = T_{msgP} = T_{DS} \tag{3.23}$$

3.4.4. Extensión a la planificación holística

La única extensión que se necesita definir para la aplicación del modelo anterior consiste en analizar dentro de un flujo de datos, los puntos de sincronización. Es decir, aquellos puntos dentro de un grafo de ejecución donde confluyen varias ramas que se ejecutan concurrentemente. En estos puntos, datos de varios productores llegan al mismo consumidor (o productor-consumidor) y la determinación de los parámetros de éste dependerá de los valores máximos de las distintas ramas.

Así, para una tarea consumidora de n mensajes,$\{msgC\}_{i=1}^{n}$, suponiendo que su ejecución empieza al recibir el último de los mensajes que esperaba, las ecuaciónes 3.16 y 3.20 quedan modificadas de la siguiente manera:

$$\phi_t = \max_{\forall i=[1...n]} \left\{ \phi^i_{msgC} + \left\lceil \frac{C^i_{msgC}}{T_{EC}} \right\rceil T_{EC} \right\} \tag{3.24}$$

Por otra parte, la determinación de los parámetros que realiza busca la maximización de las ventanas de transmisión y ejecución de mensajes y tareas, pero se puede permitir un grado de mayor de flexibilidad a la hora de asignar los parámetros de una aplicación, siendo los valores que se hallan a partir de estas ecuaciones cotas superiores al valor que se les puede asignar.

3.5. Conclusiones

En este capítulo se han presentado los modelos en los que se basará el resto del trabajo desarrollado en esta tesis. En primer lugar, se estableció el modelo de sistema, en el que las aplicaciones son gobernadas por tiempo y que ofrece la posibilidad de intercambiar mensajes asíncronos entre entidades.

Este modelo no es el único posible para el desarrollo de aplicaciones basadas en servicios, siendo cuestión de trabajos futuros el desarrollo de modelos para otros tipos de sistemas. Una de las características de la orientación a servicios es su flexibilidad, la que permite intercambiar servicios o implementaciones durante la ejecución de una aplicación. En aras de permitir de manera sencilla esta característica, se estableció que el análisis de planificabilidad sería realizado de manera independiente para cada recurso, posibilidad que ofrece la aproximación gobernada por tiempo, lo que da como resultado unos tiempos de respuesta en el peor caso pesimistas, pero no influenciados por los cambios en los desplazamientos de las tareas.

En base a dicho modelo, se definió un modelo de servicio y de perfil de servicio. Cada instanciación de un perfil de servicio genera una tarea en el nodo físico en el que se encuentre. Basándose en esta premisa, se definió un modelo temporal de perfil de servicio.

Se caracterizaron las peticiones de los clientes, y se propuso un análisis de compatibilidad entre petición y perfil. Existen otros modelos para la definición y comprobación de compatibilidades entre los requisitos de comportamiento deseados por el usuario de un determinado servicio que aporta un conjunto de datos de entrada y las interfaces, tanto de entrada como de salida, de un servicio o componente. Entre estos destacan, en el campo de la orientación a servicios, el definido por Gu en su tesis [90], aunque éste no aporta, como resulta lógico dado su campo de aplicación, un modelo temporal ni ofrece la posibilidad de definir características temporales. Tampoco contempla las relaciones de dependencia entre datos de entrada ni las relaciones de dependencia entre datos de salida de un servicio, y entiende la adaptación como un proceso que se realiza tanto en el cliente como en el servicio.

Por otra parte, existen trabajos que tratan de aplicar *Network Calculus* a sistemas de tiempo real. *Network Calculus* [36] complementa las aproximaciones probabilísticas de teoría de colas, basándose en realizar las especificaciones del tráfico de entrada en una red y las garantías de servicio que ésta ofrece, a través de la definición de cotas superiores e inferiores de ambos, lo que permite deducir las características del tráfico saliente. En concreto, el trabajo de Chakraborty et al [53] aplica estos conceptos a la composición de componentes basada en interfaces. Para comprobar la compatibilidad entre componentes, cada interfaz lleva asociada una tasa de entrada, una tasa de salida y un conjunto de variables de entrada y salida, así como predicados asociados a éstas. El álgebra de interfaces que proponen para comprobar su compatibilidad está basada en principios tomados de *Network Calculus*. La aproximación realizada por este trabajo es conceptualmente similar la presentada, la

diferencia radica en cómo se modelan los componentes o servicios y en el álgebra utilizada.

Posteriormente, se presentó el modelo de aplicación, caracterizando tanto las aplicaciones deseadas por los clientes como las instanciadas en el sistema y se elegió como modelo temporal de las aplicaciones, el definido por Calha [47], para sistemas gobernados por tiempo. Dado que en el modelo se han definido las aplicaciones como entidades gobernadas por tiempo, otros modelos de aplicación, como el de Tindell [223] o el de Gutiérrez [95], no no son aplicables, debido a que en éstos las aplicaciones son gobernadas por eventos. Finalmente, se propuso una pequeña extensión no contemplada en el modelo de Calha pero necesaria para permitir la composición de aplicaciones, dado el modelo de aplicación que se ha propuesto.

Capítulo 4

Soporte arquitectónico para la composición

Este capítulo está dedicado a los requisitos que debería cumplir una arquitectura de tiempo real que soporte composición de aplicaciones basadas en servicios. En primer lugar, se presenta una descripción funcional de dicha arquitectura, analizando los procesos principales que se ejecutarían en la misma. Posteriormente, se analizan los requisitos arquitectónicos para la composición, diferenciando entre composición dinámica o estática, para finalizar proponiendo para ambas aproximaciones, una arquitectura. Finalmente, se exponen las conclusiones.

4.1. Introducción

La presente tesis trata de demostrar la viabilidad de la aplicación de conceptos vinculados con el paradigma de orientación a servicios a arquitecturas de tiempo real como una manera de ofrecer una mayor flexibilidad en estos entornos. Como resulta lógico, las entidades y conceptos presentes en la orientación a servicios, deben adaptarse a dichos entornos, siendo necesarios una redefinición y establecimiento de los requisitos que debería cumplir una arquitectura de tiempo real que los aplique. En este capítulo se tratará de establecer los elementos básicos de dicha arquitectura, atendiendo tanto a la funcionalidad que deberá soportar, como a los requisitos de tiempo real necesarios en una arquitectura de estas características.

Por otra parte, en este documento, se diferencia entre composición estática y dinámica de aplicaciones. En la composición estática [71], tanto los servicios como todas sus implementaciones, perfiles, que estarán presentes en el sistema se conocen previamente a la composición de la aplicación, siendo esta composición, una vez realizada, estática, es decir, que no se modificará en tiempo de ejecución de la aplicación. Esta composición estática puede realizarse en tiempo de ejecución del sistema, o previamente a la ejecución del mismo. Por otra parte, en la composición dinámica [69], se permite que los nodos físicos, servicios y los perfiles de servicio

puedan aparecer y desaparecer de forma dinámica, de tal forma que las aplicaciones puedan ser reconfiguradas en tiempo de ejecución.

Como resulta evidente, ambas aproximaciones presentan requisitos arquitectónicos distintos. En este capítulo, se analizarán, en primer lugar, las entidades genéricas que deberán estar presentes en ambas arquitecturas. Posteriormente, se expondrán los requisitos arquitectónicos para cada aproximación y se propondrá para cada una de las aproximaciones realizadas una posible arquitectura.

4.2. Descripción funcional de la arquitectura

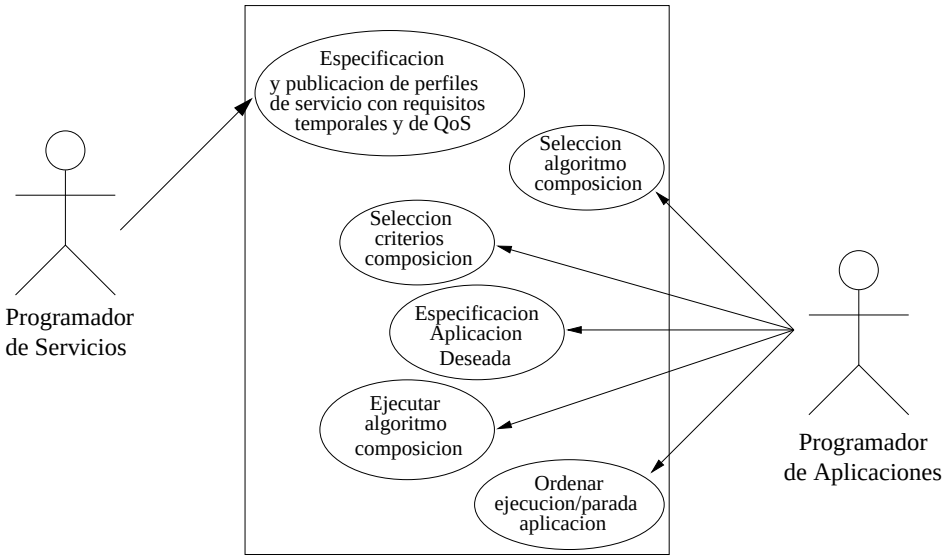


Figura 4.1: Funcionalidad básica ofrecida por la arquitectura

Desde un punto de vista funcional, una arquitectura de tiempo real que soporte composición de servicios deberá proporcionar, véase figura 4.1, las siguientes facilidades a los actores externos a la misma [80]:

- A los programadores de perfiles de servicios: para la especificación y publicación de perfiles de acuerdo con lo expuesto en el apartado 3.3. De tal manera que expliciten no sólo las características funcionales y temporales de cada perfil implementado, sino también, en general, los requisitos de calidad de servicio de cada uno de ellos.

- A los programadores de aplicaciones: para la descripción de la aplicación deseada (véase apartado 3.4.2), para la selección de un determinado criterio de selección de perfiles y de un determinado algoritmo de composición. En base a su petición, el entorno deberá descubrir perfiles que implementen una determinada funcionalidad, aplicar dicho algoritmo de composición y, si es posible,

lanzar la ejecución de la aplicación compuesta. Nótese que este actor puede ser tanto un usuario humano como otra aplicación.

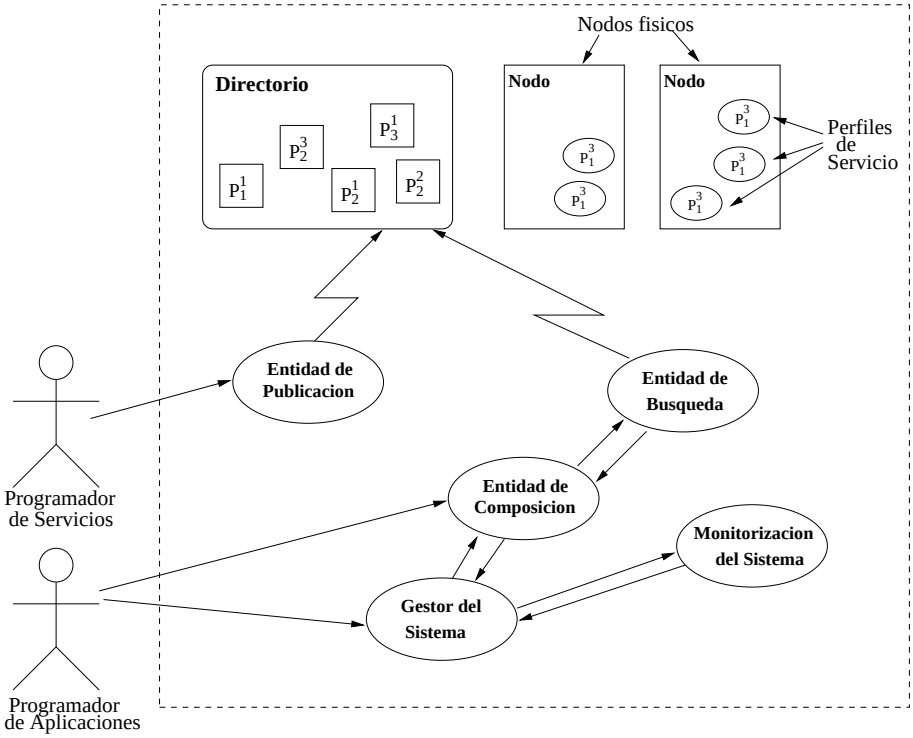


Figura 4.2: Entidades presentes en la arquitectura

Así, podemos distinguir las siguientes entidades [73, 72], véase figura 4.2:

- Perfiles de servicio: entidades *software* que implementan una determinada funcionalidad. Creados por los programadores de servicios y ubicados en nodos físicos dispersos en el sistema. Cuando son publicados, la arquitectura deberá soportar la declaración de su funcionalidad y de sus características.

- Entidad de publicación: entidad que interacciona con los programadores de perfiles de servicio para hacer públicos éstos, indicando su ubicación, funcionalidad y características. Esta entidad será la encargada de publicarlo en un directorio.

- Entidad de búsqueda: encargada del descubrimiento de perfiles de servicio, en base a la funcionalidad y requisitos expresados por los programadores de aplicaciones.

- Entidad compositora: traduce la petición del programador de aplicaciones en una estructura de datos basada en grafos e interacciona con la entidad de bús-

queda para obtener la información de los perfiles disponibles. Con esta información y los requisitos acerca de la composición aportados por el programador de aplicaciones, selecciona el subconjunto de perfiles más adecuados para la composición de la aplicación y realiza la composición. El proceso seguido por el algoritmo de composición se estudiará en más detalle en el capítulo 5.

- Gestor del sistema: el encargado de configurar el sistema, y ensamblar y desplegar una aplicación compuesta, dado un conjunto de perfiles y requisitos especificados por la entidad compositora.

- Monitor del sistema: es necesario realizar una monitorización del sistema para asegurar que las prestaciones ofrecidas a las aplicaciones existentes no se degraden y, en caso de fallo de algún componente, notificárselo al gestor, para que éste tome las medidas necesarias para evitar un fallo de todo el sistema.

Por otra parte, esta arquitectura deberá disponer de facilidades tanto para la composición de aplicaciones distribuidas, incluyendo su despliegue, gestión y monitorización, como para la publicación y descubrimiento de servicios. Además, precisa de una capa subyacente que proporcione garantías temporales en la comunicación, que podrá ser un *software* intermedio de tiempo real o un protocolo de comunicaciones de tiempo real.

4.2.1. Fases en la composición de servicios

Uno de los núcleos básicos de esta arquitectura es proporcionar facilidades para la composición de aplicaciones basadas en servicios. Dentro del proceso de composición de aplicaciones se pueden distinguir cuatro fases diferentes [70]:

- *Fase de Publicación*: fase previa en la que los perfiles de los servicios son publicados en los servicios de directorio, especificando su funcionalidad, características y requisitos de calidad de servicio. La figura 4.3 muestra un esquema de esta fase.

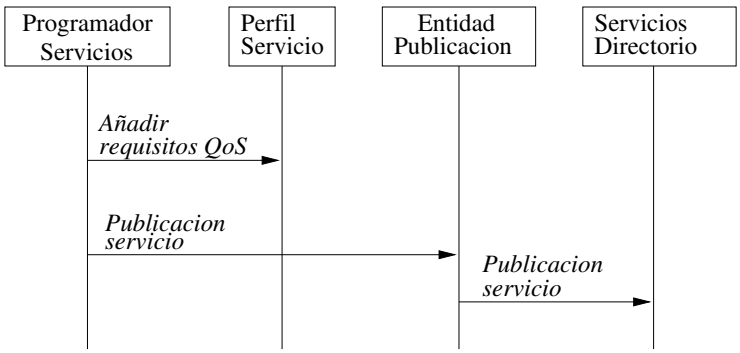


Figura 4.3: Pasos en la fase de publicación

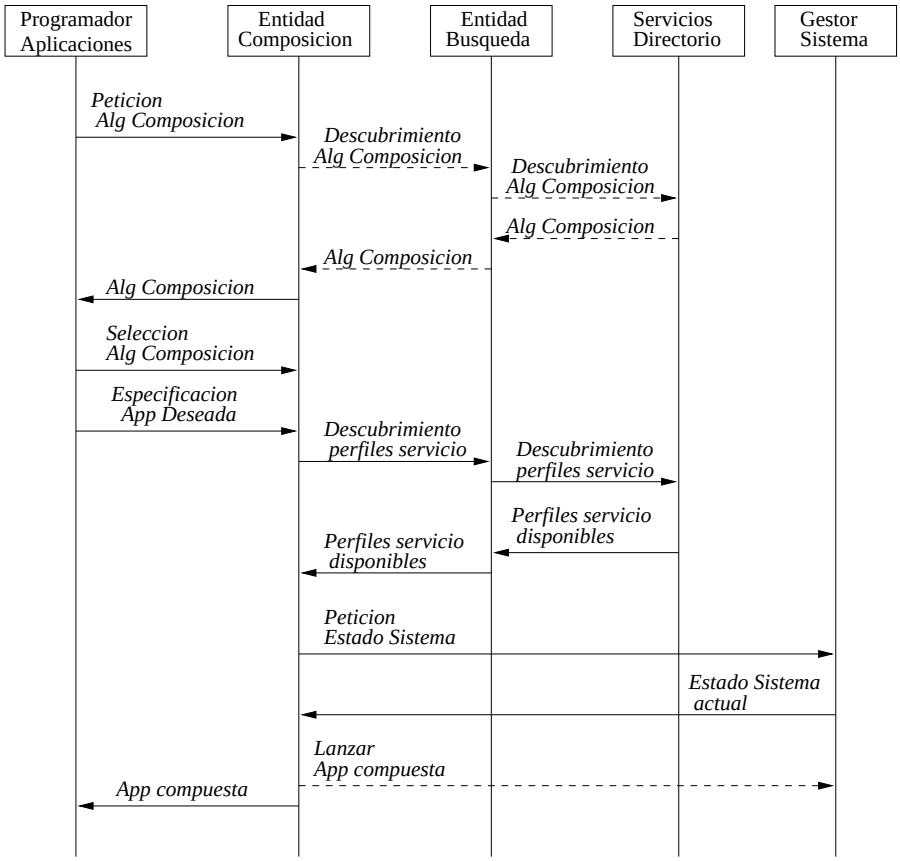


Figura 4.4: Pasos en la fases de Especificación, Descubrimiento y Composición

- *Fase de Especificación de Requisitos*: donde el programador de aplicaciones especifica la descripción de la aplicación deseada y los requisitos de *QoS* que ésta debe cumplir. Esta fase puede incluir el descubrimiento de los algoritmos de composición, en el caso de que éstos estén publicados en el servicio de directorio. En caso contrario, la entidad de composición deberá tener conocimiento de los algoritmos de composición presentes en el sistema, o que éstos sean locales a la entidad de composicón.

- *Fase de Descubrimiento*: donde se encuentran todas las posibles implementaciones, perfiles, de los servicios. Este descubrimiento estará basado en la funcionalidad del servicio requerido.

- *Fase de Composición*: en esta fase se realiza la composición propiamente dicha de la aplicación deseada. La elección está basada en los requisitos impuestos por el usuario, tanto temporales como de calidad de servicio, y en las propias características de los perfiles. Los algoritmos de composición utilizados en esta fase se explican con más detalle en el capítulo 5. En la figura 4.4 se muestra un esquema de estas tres últimas fases.

4.3. Requisitos arquitectónicos para la composición dinámica

En esta tesis se entiende por composición dinámica aquélla que se realiza en tiempo de ejecución del sistema y que, además, permite cambios en las acciones que componen una aplicación mientras ésta se está ejecutando. Permitiendo la aparición y desaparición de nuevos servicios e implementaciones de éstos, perfiles, en tiempo de ejecución del sistema. Es decir, el sistema se auto-reconfigura, ya sea por la inclusión o desaparición de una aplicación, de un conjunto de perfiles de servicio o, incluso, de un nodo físico.

Así, se perfila un escenario cambiante, donde las condiciones de carga tanto en los nodos como en la red evolucionan a lo largo del tiempo, donde es necesaria una gestión de los servicios y perfiles existentes, del ancho de banda y de la planificabilidad del sistema, una monitorización del mismo y una gestión de la calidad de servicio. Además, todos estos mecanismos deberán ser implementados de tal forma que introduzcan baja latencia, en términos de la granularidad de la escala temporal del sistema.

Por otra parte, la composición de aplicaciones de tiempo real, que se propondrá en el siguiente capítulo, exige un conocimiento prácticamente completo de todas las entidades existentes en el sistema a todos los niveles, desde el conjunto de perfiles que implementan un determinado servicio, hasta la planificación de todos los recursos involucrados. Sólo el hecho de permitir la composición de aplicaciones en tiempo de ejecución y que las acciones involucradas en las mismas varíen sus parámetros dinámicamente, implica que el proceso de composición, planificación y gestión de aplicaciones necesita acceder a toda la información del sistema con la seguridad de que está actualizada en todo momento. La posibilidad de añadir entidades, como perfiles y servicios, en tiempo de ejecución, la gestión de los perfiles como un conjunto de réplicas y la reconfiguración de aplicaciones justifica aún más este requisito.

Por lo tanto, en el entorno que se presenta, todos los mecanismos de control que se implementen (incluyendo la composición) deberán poder acceder a toda la información del sistema, de tal forma que la información a la que accedan cada uno de ellos en cada momento sea la misma. En otras palabras, existe la necesidad de una base de datos centralizada o distribuida con toda la información sobre el estado y los requisitos del sistema.

Esta base de datos de requisitos del sistema aporta:

- La posibilidad de realizar cambios en tiempo de ejecución con bajas latencias tanto en el conjunto de acciones del sistema, como en la propia política de planificación del mismo.

- Facilidades para realizar el control de admisión en tiempo de ejecución y la gestión de ancho de banda, también con baja latencia.

- Soporte para gestionar la replicación de entidades, ya sean servicios, en forma de perfiles de servicio, ya sean nodos físicos.

Por otra parte, en una arquitectura distribuida, los mecanismos de control pueden estar centralizados o distribuidos. Por ejemplo, en la automoción los mecanismos de control de ABS, ESP y suspensión activa podrían ser tratados como subsistemas independientes, o integrarlos en un único sistema que aumente la estabilidad del coche. La aplicación de un control jerarquizado puede simplificar en gran medida la gestión de niveles de control más elevados y facilita la integración funcional de componentes. Ateniéndonos al ejemplo, una mayor integración puede aumentar las prestaciones del sistema, su eficiencia en el uso de recursos, reducir el número de componentes activos y el coste del sistema completo, y también puede ayudar a manejar la complejidad del mismo.

En un entorno que permita la composición de aplicaciones distribuidas están presentes múltiples entidades que manejan prácticamente la misma información aunque sus cometidos sean diferentes. Estas entidades, como por ejemplo la entidad compositora, el gestor de perfiles o el planificador del sistema, podrían estar ubicadas en localizaciones diferentes, siempre y cuando la información que compartan sea la misma. Sin embargo, si se permite que estas entidades estén ubicadas en el mismo nodo, existiendo una única base de datos que compartan entre ellas, se elimina, en primer lugar, la complejidad asociada al mantenimiento de la base de datos distribuida, y, como existe interacción entre las distintas entidades (por ejemplo, la entidad compositora debe asignar los desplazamientos temporales a las acciones involucradas en una aplicación en función de la planificación del sistema; el gestor de perfiles deberá notificar la existencia de un nuevo perfil y replanificar, si es posible, el sistema, etc.), las prestaciones del sistema completo aumentan si se integran en una única entidad.

4.3.1. Arquitectura propuesta

En este apartado se propone una arquitectura para la composición de aplicaciones de tiempo real basadas en servicios. La implementación de esta arquitectura sobre un protocolo concreto de comunicaciones de tiempo real se expondrá en el capítulo 6.

En el modelo de sistema, discutido en el apartado 3.2, se contempla una aproximación gobernada por tiempo para las aplicaciones del sistema, donde las acciones involucradas en la ejecución de éstas tienen instantes de activación predefinidos. En un entorno dinámico estos instantes de activación podrán variar en tiempo de ejecución, siendo responsabilidad de una entidad que gestione el sistema el indicar cuándo y cómo varían estos instantes. Esta situación puede ser tratada de manera eficiente si la gestión se realiza de manera centralizada, por ejemplo siguiendo el modelo maestro-esclavo, donde el maestro indica a los nodos físicos cuándo ejecutar una tarea o transmitir un mensaje a través de mensajes específicos de control.

En esta arquitectura propuesta se seguirá el modelo maestro-esclavo (productor-distribuidor-consumidor, véase apartado 2.2.2.3), donde un nodo maestro actuará de gestor del sistema distribuido, controlando la ejecución de las tareas y las transmisiones de los mensajes, y realizando la gestión de las múltiples implementaciones, perfiles, de los servicios. Así, en este nodo estará ubicada la base de datos de requisitos del sistema (BDRS), e integrará las funciones de las entidades de composición, gestión de perfiles, planificación y gestión de calidad de servicio.

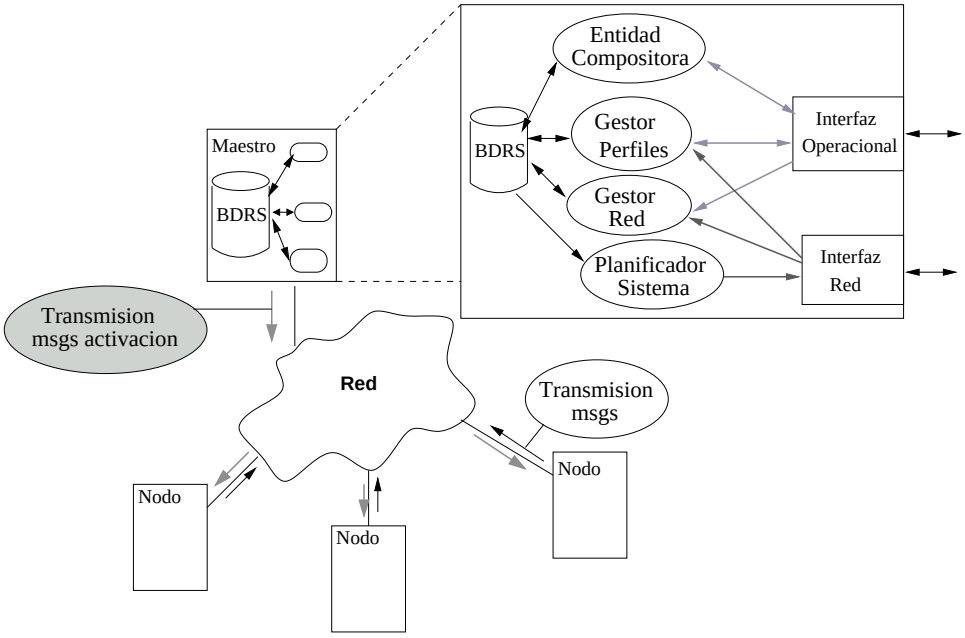


Figura 4.5: Arquitectura propuesta para la composición dinámica

En la figura 4.5 se muestra un esquema de esta arquitectura propuesta. En el nodo maestro se ubicará la base de datos de requisitos del sistema, que debería incluir:

- Para cada aplicación existente en el sistema:
  - Su configuración actual, incluyendo los parámetros temporales de todas las tareas y mensajes intercambiados entre éstas.
  - La especificación realizada por el usuario en el momento de su composición. Si se produjese una reconfiguración de la aplicación, esta información podría ser necesaria para comprobar que sus requisitos se sigan cumpliendo. Esta reconfiguración puede deberse a la substitución de un perfil por otro que implemente la misma funcionalidad, a la replanificación de los instantes de activación de las acciones de la aplicación para permitir la instanciación de una nueva, o a la necesidad de substituir un conjunto de perfiles de la aplicación por otro equivalente (definido en los grafos de substitución en la especificación).

- Para cada servicio existente en el sistema:
  - Su funcionalidad.
  - Conjunto de perfiles que lo implementan, incluyendo su ubicación y características, tanto temporales como de calidad de servicio.
- Para cada nodo físico del sistema:
  - Conjunto de tareas activas en el nodo con sus parámetros temporales. Esta información es necesaria para comprobar si se puede instanciar una nueva tarea y calcular su tiempo de respuesta.
  - Conjunto de mensajes transmitidos y recibidos.
  - Conjunto de perfiles de servicio, activos o no, presentes en dicho nodo.

Es decir, la base de datos no sólo incluirá el estado y configuración de las aplicaciones, tareas y mensajes activos en cada momento, sino que contendrá información adicional acerca de todas las entidades existentes en el sistema, aplicaciones, perfiles no activos y servicios, asumiendo la funcionalidad adicional del servicio de directorio necesario para la implementación de una arquitectura basada en servicios.

Por otra parte, estarán presentes en el nodo maestro, las siguientes entidades:

- Un gestor de la red que distribuirá el ancho de banda disponible de acuerdo con una política determinada, forzando el cumplimiento de las garantías temporales a través del uso de planificación en la red. Este gestor recibirá a través de la interfaz de red información sobre el estado del sistema y a través de la interfaz operacional las instrucciones adecuadas sobre la configuración y planificación de la red. Teniendo en cuenta ambas fuentes de información, modificará convenientemente la BDRS, especificando un determinado conjunto de mensajes a transmitir que será planificado junto al conjunto de tareas por el planificador del sistema.
- Un planificador del sistema que construirá las planificaciones del sistema, para que se puedan transmitir, posteriormente, los mensajes de activación de las tareas que se ejecutarán en los nodos esclavos y de los mensajes a transmitir por éstos.
- Un gestor de perfiles que mantendrá actualizados los datos de los perfiles de cada servicio existente en el sistema. Será el encargado de incluir los nuevos perfiles que aparezcan en tiempo de ejecución y de eliminar de la base de datos los perfiles que desaparezcan. En caso de que se implemente tolerancia a fallos a nivel de aplicación, también será el encargado de gestionar las réplicas (posibles perfiles substitutos). Nótese que el gestor de perfiles asume las funciones de la entidad de publicación.

- Una entidad compositora que, teniendo conocimiento del estado actual del sistema, compondrá las nuevas aplicaciones de acuerdo con los requisitos establecidos por los usuarios. Dentro de las funciones que implementa esta entidad compositora se incluirá el control de admisión de las nuevas aplicaciones en el sistema. Los algoritmos de composición disponibles, así como los criterios que pueden utilizarse en la composición pueden formar parte de la BDRS o ser locales a la entidad compositora. Nótese que en esta arquitectura propuesta, la entidad compositora asume las funciones de la entidad de búsqueda.

Para realizar la gestión de los perfiles, es necesario incluir en la arquitectura un mecanismo de comunicación asíncrono que informe al maestro de la aparición de nuevos servicios, perfiles o nodos en el sistema.

Por otra parte, sería importante que la arquitectura propuesta ofrezca un sistema de replicación del maestro para eliminar así este punto de fallo. Sería necesaria la replicación de la base de datos en otra localización del sistema, de tal forma que se mantuviese la coherencia entre las bases de datos frente a cambios en éstas. Asimismo, debería ofrecer un modelo de detección de fallos tanto a nivel de maestro, como a nivel de nodo físico y aplicación. La implementación de los mecanismos de detección y de tolerancia a fallos cae fuera del ámbito de la presente tesis, siendo cuestiones a tratar en cada arquitectura y protocolo de comunicaciones en concreto en la que se implemente el mecanismo de composición.

4.4. Requisitos arquitectónicos para la composición estática

La composición estática en esta tesis se refiere a aquélla que, una vez realizada, no se modifica en tiempo de ejecución, existiendo un conocimiento *a priori* de todos los servicios y perfiles existentes en el sistema en el momento de la composición, con la seguridad de que, una vez compuesta la aplicación, éstos no desaparecerán del sistema.

Si todas las aplicaciones y los perfiles de servicio que estarán presentes en el sistema son conocidos antes de la ejecución del mismo, la composición se convierte en una planificación *off-line* donde el grado de complejidad reside en la selección de un determinado conjunto de perfiles para una aplicación en concreto. Esta selección puede realizarse haciendo uso de los algoritmos de composición que serán propuestos en el capítulo 5.

La composición estática puede realizarse de acuerdo con el esquema propuesto para la composición dinámica, sin más que no permitir los cambios dinámicos en los perfiles, nodos y servicios disponibles en el sistema. Sin embargo, en este apartado se esbozarán los requisitos arquitectónicos que tendría una arquitectura que realizase una composición estática sin otro requisito que el de la existencia de la pila de protocolos adecuada en el nodo en el que se realice la petición.

Este apartado se centra en aquellos entornos donde la composición se realizará de manera centralizada, desde el nodo que realiza la petición. Los perfiles de servicio podrán estar ubicados en cualquier nodo físico del sistema, pero deberán ser o bien descargados al nodo donde se realiza la composición, o bien cada perfil de servicio involucrado dispondrá de un substituto (*proxy*) que se descargará al nodo físico en cuestión. Desde el punto de vista del programador de aplicaciones, estas dos aproximaciones no presentan diferencias. Sin embargo, tienen implicaciones muy distintas:

- Si todos los perfiles se deben descargar al nodo físico donde se realiza la composición, el problema de la composición distribuida se convierte en únicamente la elección para un único recurso, de los perfiles más adecuados, incluyendo su planificación en dicho nodo. Esta elección se basará en el cumplimiento de los requisitos de calidad de servicio de dichos perfiles y en el cumplimiento del plazo deseado.

  Para asegurar ambas condiciones, se deberá tener conocimiento del comportamiento del perfil descargado en el *hardware* concreto en el que se ejecutará la aplicación, así como de la variación de sus parámetros temporales. Este conocimiento podrá ser obtenido a través de información del perfil que de forma explícita asocie comportamiento con plataforma, o de forma indirecta a través de modelos temporales de las plataformas y anotación del código descargable.

  En este último caso, y únicamente con respecto al comportamiento temporal, pueden aplicarse técnicas como la expuesta en [102] , para el cálculo del tiempo de respuesta. Este trabajo, desarrollado por el grupo de tiempo real de la universidad de York, basándose en resultados del campo del análisis de tiempo de ejecución en el peor caso [183], propone para XRTJ (*eXtensible high-integrity Real–Time Java*) técnicas de análisis de las plataformas *hardware* para extraer un modelo temporal de las mismas, que, combinado con anotaciones en los ficheros de clase java y en los propios *bytecodes*, permite obtener cotas superiores del tiempo de ejecución en el peor caso de un código. Esta aproximación exige al programador de perfiles de servicio la introducción de información en el código acerca de caminos no realizables, frecuencias de ejecución, secuencias de ejecución de trozos de código y la dependencia entre ellos.

- Si cada perfil tiene un substituto que se descargará en el nodo físico donde se ha realizado la petición, en primer lugar se ha de contar con un protocolo de comunicaciones de tiempo real o de un *software* intermedio que ofrezca garantías temporales a la comunicación entre cliente y servidor.

  Por otra parte, la composición de aplicaciones es estática, pero esta composición puede producirse en tiempo de ejecución del sistema. Así que los nodos

físicos donde se encuentren los perfiles deberán realizar un análisis de planificabilidad para determinar si pueden admitir una o varias tareas periódicas nuevas. Estas tareas periódicas son la materialización de las invocaciones que se realizarán al perfil o perfiles requeridos por la Entidad Compositora en dicho nodo físico. Además, deberá determinar si las prestaciones ofrecidas a las demás tareas presentes en dicho nodo (pertenecientes a aplicaciones ya existentes en el sistema) se ven afectadas, pues en ese caso, deberá rechazar la instanciación de una nueva tarea. En el caso de que esta nueva tarea pueda ser aceptada, calculará su tiempo de respuesta en el peor caso, que comunicará junto a los parámetros de calidad de servicio ofrecidos en dicho instante a la Entidad Compositora. Esta aceptación supone un contrato tácito con la Entidad Compositora en la que el nodo se compromete, si ésta lo elige, a mantener las condiciones de calidad de servicio que le ha comunicado. Es decir, el proceso de composición en este caso se convierte en un proceso distribuido.

En el siguiente apartado se propone una arquitectura para la composición estática.

4.4.1. Arquitectura propuesta

En [71, 80] se propuso una arquitectura para la composición estática, CoSeRT (*Composition of Service-Based Real-Time Applications*), la cual se expone en 4.6. Esta arquitectura se centraba en la composición estática de las aplicaciones con descarga de los perfiles de servicio al nodo que realizaba la composición.

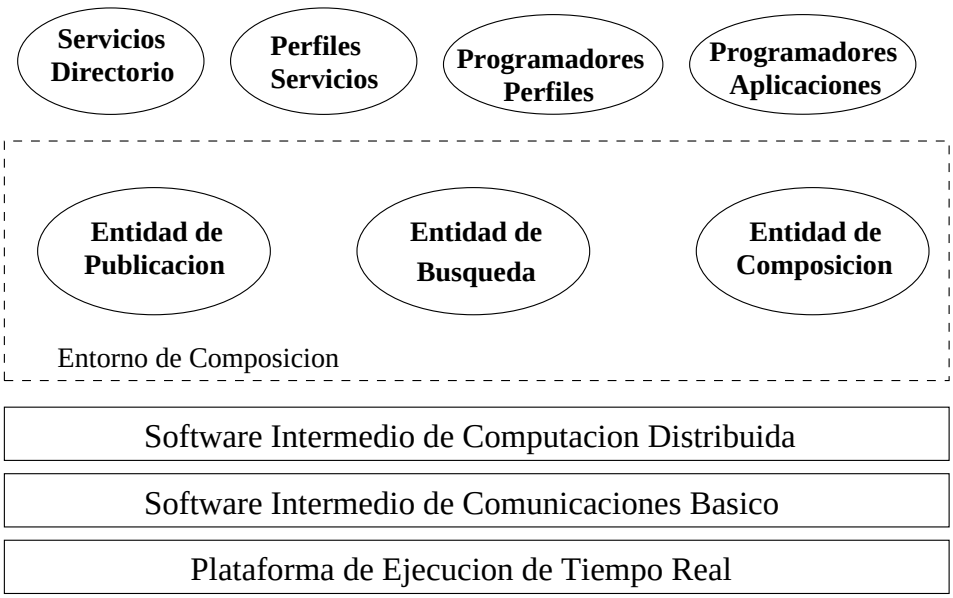


Figura 4.6: Arquitectura propuesta para la composición estática

Para demostrar la validez de la composición estática de aplicaciones de tiempo real, se implementó sobre tecnología Java, un prototipo de dicha arquitectura. Este prototipo no se llegó a probar en una red de tiempo real, limitándose a demostrar la viabilidad de la aplicación de esta idea mediante simulaciones de un sistema distribuido embarcado de tiempo real donde las aplicaciones pudieran ser compuestas a partir de servicios descargables (todavía no se había desarrollado el concepto de perfil de servicio en este trabajo). Las aplicaciones a las que se enfocaba esta arquitectura eran, sobre todo, aplicaciones multimedia, como reproductores de vídeo, audio, etc. que dependiendo de la plataforma en la que fueran a ser ejecutadas y de los requisitos de calidad impuestos, ya por el usuario, ya por la propia plataforma, podrían requerir el uso de un perfil de uso de recursos (incluyendo memoria, CPU, batería...) u otro.

En este trabajo, el entorno de composición se basaba en la existencia de un *software* intermedio de comunicaciones básico, como RMI [210] o RT-CORBA, implementándose el prototipo sobre Jini [212]. Los servicios fueron programados en lenguaje Java y serializados en los servicios de directorio.

En CoSeRT, se permitió en el momento de la publicación y descubrimiento de servicios, la descripción de los requisitos de calidad de servicio como un atributo adicional, donde se indicaban para cada servicio:

- Su funcionalidad.
- El conjunto de tareas que forman parte de un servicio.
- Los requisitos de precedencia con respecto a otros servicios, es decir, si necesitaba de la ejecución de otro servicio antes de su invocación.
- La media de consumo de memoria en su ejecución.
- Requisitos de memoria y procesador.

El descubrimiento de servicios, se basa únicamente en la funcionalidad de éstos, habiéndose implementado una entidad de búsqueda, que permite a los clientes realizar una búsqueda configurable de las implementaciones de cada servicio y obtener los parámetros de calidad de servicio de cada uno de ellos. En esta arquitectura, los algoritmos de composición también son publicados en los servicios de directorio, ofreciendo la posibilidad al cliente de elegir entre los disponibles en el sistema.

En este trabajo, los algoritmos de composición desarrollados son:

- FIFO: la primera combinación válida.
- Mejor tiempo de respuesta extremo a extremo.

Asimismo se desarrolló una interfaz de usuario a través de la cual el cliente podía realizar peticiones de composición.

4.5. Conclusiones

En este capítulo se han presentado los elementos básicos de una arquitectura de tiempo real que dé soporte a la composición de aplicaciones basadas en servicios. En primer lugar, se realiza una descripción funcional que sirve de base para la identificación de las entidades que deberían estar presentes.

En una arquitectura que soporte SOA existen tres elementos principales [103]: un proveedor, un consumidor y un directorio. Los proveedores publican o anuncian sus servicios en directorios, donde los consumidores pueden encontrarlos y, posteriormente, invocarlos. Por otra parte, los programadores de aplicaciones usarán estos servicios a través de la composición.

En este trabajo a cada implementación concreta de un servicio, se le denomina perfil, programado por un programador de servicios, que será el encargado de publicarlo en un directorio. Por otra parte, los consumidores, programadores de aplicaciones, especifican una aplicación en términos de los servicios que utilizará. Una entidad compositora será la encargada de realizar la búsqueda, usando la entidad de búsqueda, y de elegir los más adecuados. La *adecuación* no sólo dependerá de los requisitos establecidos por el cliente sino también de los requisitos del sistema: se ha de asegurar que la instanciación de una aplicación no pone en peligro las prestaciones del sistema completo ni la planificabilidad de ninguna aplicación ya instanciada. Así, la entidad compositora deberá interaccionar con el gestor del sistema quien conoce el estado del sistema a través de la monitorización.

La inclusión de requisitos temporales en las aplicaciones, implica que se deberá realizar un control más exhaustivo tanto en la red, como en los nodos involucrados a fin de asegurar no sólo la calidad de servicio de la nueva aplicación, sino también del resto de elementos presentes en el sistema.

La aproximación seguida para la composición de aplicaciones de tiempo genera distintos requisitos arquitectónicos. En función de estos requisitos y de la aproximación se realizada se han clasificado las posibles arquitecturas en:

- Arquitecturas para la composición dinámica: permiten reconfiguraciones de las aplicaciones en tiempo de ejecución, así como la inclusión/desaparición de nuevos elementos en tiempo de ejecución.

- Arquitecturas para la composición estática: una vez realizada la composición, ésta no cambiará. Esto implica que se asegura que los servicios y perfiles de servicio no desaparecerán durante la ejecución. Por otra parte, atendiendo al momento en el que se realice la composición, se puede clasificar la composición estática en:

    - En tiempo de ejecución del sistema: se permite que se realice mientras que el sistema está ejecutándose.
    - Previa a la ejecución del sistema: totalmente *off-line*.

Apenas existen trabajos que propongan una arquitectura de tiempo real que soporte SOA. El más cercano [228], identifica elementos de la arquitectura abstrayéndose o soslayando las restricciones temporales que debería contemplar, limitándose a enunciar entidades genéricas presentes en la orientación a servicios, sin ahondar en las diferentes arquitecturas a las que dan lugar las decisiones de diseño tomadas (como el permitir reconfiguraciones, o que la composición se realice en tiempo de ejecución).

La aportación principal de este capítulo no son sólo las propuestas realizadas, sino el hecho de desarrollarlas con una fuerte orientación a comunicación de tiempo real, es decir, que todo el análisis y las arquitecturas propuestas no se aíslan de los requisitos temporales de la red subyacente, sino que son influenciados por su existencia. Siendo ésta una constante en todo el documento.

En concreto en este capítulo, las aportaciones realizadas son: en primer lugar, la identificación de las funcionalidades que debe ofrecer una arquitectura de tiempo real que ofrezca soporte a orientación a servicios; en segundo lugar, el análisis de las implicaciones que tiene en dicha arquitectura determinadas decisiones de diseño; y, por último, el análisis de requisitos y la realización de propuestas de arquitecturas para cada tipo de composición identificada.

Capítulo 5

Algoritmos para la Composición de Aplicaciones

En este capítulo se presenta, en primer lugar, la problemática de la composición inicial de aplicaciones distribuidas de tiempo real, proponiéndose un algoritmo exhaustivo para realizar la búsqueda de una combinación inicial óptima en términos de una figura de mérito. Sin embargo, la excesiva complejidad computacional de un algoritmo de estas características, hace inviable su aplicación a contextos donde la composición inicial se realiza en tiempo de ejecución del sistema. Para dichos entornos, se propone un algoritmo mejorado que ofrece una solución subóptima en un tiempo acotado, dado un número de combinaciones posibles. Finalmente, se presenta la validación de estos algoritmos.

5.1. Introducción

La composición de entidades *software* a partir de módulos, de forma eficiente y cumpliendo unas determinadas restricciones, es uno de los campos de investigación abiertos en múltiples ramas de la ingeniería del *software*. En el campo de las arquitecturas orientadas a servicios, la construcción de servicios compuestos, ha sido tratada, tal y como se presentó en el apartado 2.6.1 del estado del arte, en numerosos trabajos, siendo de especial relevancia el del grupo de Klara Narhstedt [91, 143], que propone la composición de servicios como un problema similar al enrutado en una red con calidad de servicio. Sin embargo, en ningún trabajo de los propuestos en este campo se plantean soluciones para la composición de aplicaciones con restricciones temporales.

Por otra parte, en el mundo de la investigación de los sistemas de tiempo real, la composición de servicios puede asimilarse a la elección de una ubicación para las tareas que formarán parte de una aplicación distribuida, permitiendo que las tareas cambien sus parámetros temporales en función de su ubicación, y restringiendo el número de ubicaciones posibles para cada tarea. En el campo de los sistemas de

tiempo real, se han propuesto soluciones para la elección de parámetros de tareas, tanto en tiempo de ejecución [47] como *off-line* [221, 173, 174], problema similar al planteado, restringiendo cada tarea a una única ubicación, y marcando como única restricción para la composición el cumplimiento del plazo. Si se aumenta el número de ubicaciones posibles para una determinada tarea y se ofrece la posibilidad de realizar la composición de aplicaciones distribuidas en función de múltiples restricciones, hasta donde se tenía conocimiento en el momento de la elaboración de esta tesis, no se han propuesto soluciones al problema planteado.

En el campo de las aplicaciones que ofrecen calidad de servicio, se han propuesto soluciones para, una vez compuestas las aplicaciones de un sistema distribuido, realizar reservas y asignación de recursos [185, 184, 139], pero siempre partiendo de una aplicación ya planificada y donde la asignación de recursos se realiza en términos de maximización de la utilidad global del sistema.

En este capítulo se estudia cómo realizar composición de aplicaciones de tiempo real distribuidas basadas en servicios. En primer lugar, se plantea la problemática existente en la composición de aplicaciones de tiempo real a partir de servicios previamente existentes. Ésta tiene dos vertientes: en primer lugar, la elección de los servicios adecuados, en función de unas determinadas restricciones, que compondrán la aplicación distribuida y, en segundo lugar, la asignación de parámetros temporales a dichos servicios, asegurando que no se ponen en peligro ni la planificación ni las prestaciones del sistema completo. Dado que, como se puede intuir, la elección de los parámetros temporales y la selección de los perfiles de servicio adecuados son problemas codependientes, la solución no puede estar basada en el abordaje de cada una de las vertientes por separado, sino que ha de plantearse de forma global. Se propone un algoritmo de composición exhaustivo para la composición de aplicaciones basado en la búsqueda de una combinación de perfiles que minimice una determinada figura de mérito global que refleje los deseos del usuario y las restricciones impuestas por el propio sistema. Sin embargo, el uso de algoritmos exhaustivos cuando el número de combinaciones a explorar crece, es excesivamente costoso para poder ser realizado en tiempo de ejecución. Para poder realizar la composición en tiempo de ejecución del sistema, es necesario un algoritmo mejorado que, ofreciendo una combinación subóptima aunque suficientemente cercana a la del exhaustivo, explore un menor número de combinaciones. Este algoritmo mejorado se basa en el uso de una figura de mérito relativa que refleje la figura de mérito global que se emplee en el exhaustivo, y, además, en el uso de heurísticos que acotan el número de combinaciones a explorar.

Es importante señalar que, en este capítulo, se plantea solución al problema de la búsqueda de una solución a la composición inicial de aplicaciones, que se aplicará tanto en la composición dinámica como estática, no tratando la reconfiguración de aplicaciones.

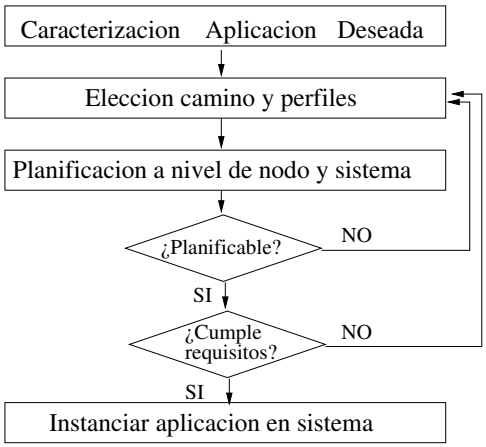


Figura 5.1: Presentación de la problemática

5.2. Planteamiento del problema

El sistema recibe una petición de un cliente, una aplicación o un usuario humano, que define una aplicación deseada, $\mathcal{A}_D = (G_p, \{G_s^i, CR_s^i\}, CS_{req})$, y un conjunto de restricciones y características, $\vec{\Theta}$, temporales y de calidad de servicio que deberá cumplir dicha aplicación. Estas restricciones pueden referirse a toda la aplicación en conjunto, $\vec{\Theta}_{e2e}$, como el plazo temporal extremo a extremo que deberá cumplir, la calidad final deseada y las características de calidad de servicio de entrada; o ser relativas a un servicio del grafo en concreto, $\vec{\Theta}_{intra}^i$, como los requisitos presentados en el apartado 3.3.2. A partir de la especificación realizada por el cliente, el sistema deberá elegir aquel conjunto y configuración de perfiles que, respetando la planificabilidad y prestaciones del sistema, cumpla las especificaciones del usuario.

Como puede observarse en la figura 5.1, este problema tiene varias dimensiones: en primer lugar se ha de escoger, en función de los requisitos funcionales y no funcionales impuestos por el usuario, un conjunto de perfiles concreto, comprobando la compatibilidad entre ellos. Posteriormente, se han de asignar parámetros temporales a las tareas de dichos perfiles y a los mensajes intercambiados entre ellas. Entre estos parámetros temporales se encontrará *siempre* el desplazamiento temporal u *offset*. Para cada tarea instanciada y para cada mensaje involucrado en la aplicación, deberá comprobarse su planificabilidad, a nivel de nodo y de red, comprobando que su instanciación no merma las prestaciones ofrecidas a las demás tareas y aplicaciones previamente existentes en el sistema. Finalmente, deberá comprobarse si los requisitos impuestos por el usuario se cumplen.

Con sólo una combinación posible de perfiles, y si sólo se debe cumplir como restricción impuesta por el usuario el plazo extremo a extremo, el problema se reduce a realizar un análisis de planificabilidad distribuido asignando desplamientos temporales, ya sea considerando las tareas independientes [222, 221], ya sea considerando sus relaciones de precedencia [173, 174]. El escenario se complica si se

desea ofrecer al usuario la mejor combinación en función de una métrica, como *La combinación que tenga el menor (o mayor) tiempo de respuesta extremo a extremo* o *La combinación que equilibre mejor la carga entre los nodos involucrados*, pues, idealmente, deberán explorarse todas las combinaciones posibles de perfiles antes de decantarse por una en concreto.

Anteriormente, en el apartado 3.4.3, se presentó el modelo temporal de una aplicación, que aplicaremos en este capítulo. En este modelo se asumía que el protocolo de comunicaciones estructuraba la comunicación en ciclos, de tal forma que en cada ciclo no pueden darse relaciones de precedencia pues, a efectos de temporales, dos mensajes enviados en el mismo ciclo son simultáneos.

Por otra parte, en este capítulo se asume la existencia de una entidad compositora, como la presentada en el capítulo 4, que conoce el estado actual del sistema. Esta entidad elegirá la combinación y configuración de perfiles que, cumpliendo los requisitos impuestos por el usuario, mejor se adapten ya sea a una cierta política del sistema, por ejemplo equilibrado de carga en los nodos, ya sea a un deseo expreso del usuario, por ejemplo minimización de la utilización total en los nodos involucrados o, simplemente, la primera combinación válida que se encuentre.

Para poder elegir un camino determinado, la entidad compositora deberá, en algún momento, construir el grafo de todos los posibles caminos de la aplicación. Éste es un grafo dirigido multivaluado, pues cada uno de los perfiles (vértices o nodos del grafo) tendrán asociadas características temporales y no temporales, y los arcos que los enlazan, mensajes, tendrán a su vez, sus propias características. En la elección de un camino determinado, siempre estará presente como restricción el cumplimiento del plazo de la aplicación deseada. Además, pueden contemplarse otras restricciones como el cumplimiento de una determinada calidad de servicio o la maximización de la utilidad del sistema.

5.3. Algoritmo de Composición

En este apartado, como ya se comentó en la presentación de la problemática, se parte de la premisa de que la entidad encargada de componer las aplicaciones posee toda la información relativa al estado actual del sistema, es decir, los servicios existentes y sus perfiles asociados, la carga en cada nodo y la carga total del sistema.

El modelo de sistema empleado y la aproximación al diseño del mismo son las presentadas en el apartado 3.2. Se empleará una aproximación holística para la composición de aplicaciones distribuidas, donde la planificación se realizará de forma independiente en cada recurso, suponiendo independencia entre acciones y calculando sus tiempos de respuesta en el peor caso sin tener en cuenta sus relaciones de precedencia.

El modelo temporal de una aplicación que se empleará es el presentado en el apartado 3.4.3, donde la comunicación se estructura en ciclos y no existen relaciones de precedencia temporal entre mensajes que se transmiten en distintos ciclos, pues

a efectos temporales son simultáneos. Como la aceptación de un nuevo mensaje en la red viene condicionada por el resultado positivo de un test de planificabilidad, la utilización de este modelo no supone una pérdida de generalidad, pues el algoritmo desarrollado puede aplicarse en redes de tiempo real que presenten otras características, sin más que emplear el análisis de planificabilidad pertinente en la red para el cálculo de su planificabilidad y sus tiempos de respuesta de transmisión en el peor caso.

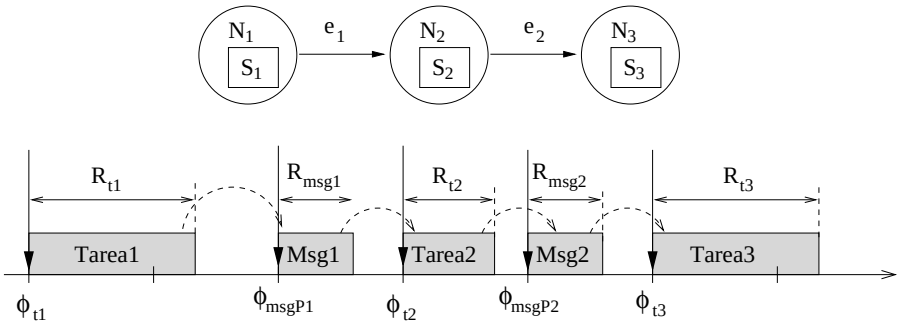


Figura 5.2: Aplicación ejemplo y su posible cronograma temporal

En la figura 5.2 se presenta el grafo de servicios de una aplicación sencilla. Una vez elegidos los perfiles de servicio y establecidos los parámetros temporales de las tareas a las que darán lugar, su cronograma temporal será similar al presentado en la figura.

De cada perfil se conoce su tiempo de ejecución en el peor caso, pero para poder comprobar su planificabilidad en cada nodo físico se deberá calcular su tiempo de respuesta. Es labor de la entidad compositora no sólo seleccionar los perfiles adecuados, sino también establecer los parámetros temporales, tanto de las tareas que se generarán al seleccionar un perfil, como de los mensajes intercambiados. Estos parámetros temporales serán su plazo relativo y su desplazamiento temporal. El período será el mismo para todas las entidades que compongan una aplicación e igual al período de la misma.

Cada perfil está ubicado en un nodo, pudiendo coexistir varios perfiles de la misma aplicación en el mismo nodo y, a su vez, varias tareas vinculadas al mismo perfil. Este hecho implica que, *a priori*, sin saber exactamente cuáles son todos los perfiles que se instanciarán en cada nodo físico, no se puede calcular el tiempo de respuesta en el peor caso para cada uno de ellos.

En la figura 5.3 se presenta una aplicación ejemplo, donde los nodos físicos están representados por cajas en las que se ubican perfiles P_i^α. Estos perfiles pueden estar activos (en la figura, aquellos rodeados por un círculo), es decir, pertenecientes a una aplicación y planificados con una o más tareas en ejecución vinculadas al mismo, o ser residentes en ese nodo sin consumir recursos en él (en la figura, aquellos que no están rodeados). Para calcular en el ejemplo de la figura el tiempo de respuesta en el peor caso de P_1^A, es necesario saber cuáles son todas las tareas instanciadas en el

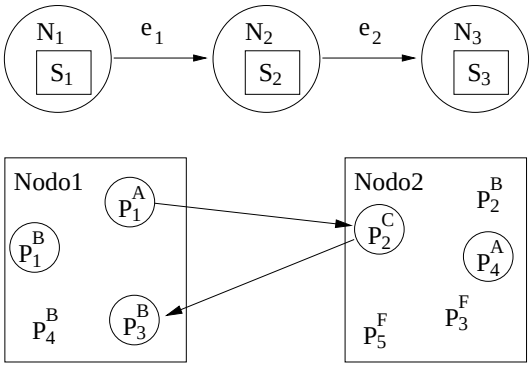


Figura 5.3: Aplicación ejemplo y posibles ubicaciones de perfiles

$Nodo1$, incluyendo P_3^B. Aún suponiendo que P_3^B es una tarea de menor prioridad, puede influir en el cálculo del tiempo de respuesta de P_1^A, ya sea por activaciones retrasadas, ya sea por el incremento en el término de bloqueo debido a la compartición de recursos.

Dentro del diseño de sistemas distribuidos, la asignación de prioridades a tareas y mensajes dentro de una aplicación distribuida es un problema en sí mismo, pues no existe una asignación de prioridades óptima, debiendo emplearse heurísticos para su asignación. El algoritmo de composición presentado en este apartado no tiene en cuenta el esquema de asignación de prioridades a las tareas ni a las aplicaciones, suponiendo que, en principio, son asignadas por una entidad externa, ya sea el propio sistema, ya sea el desarrollador de aplicaciones, según un esquema de prioridades determinado, que puede ser basado en bandas de prioridad [61], o mediante el uso de heurísticos desarrollados a tal efecto [94, 14].

La red se considerará un recurso único que también deberá ser planificado, calculando los tiempos de respuesta en el peor caso de transmisión de cada mensaje.

Aunque el algoritmo presentado en este apartado está pensado para ser ejecutado previamente a la instanciación de la aplicación concreta en el sistema, esta composición debería poder realizarse en tiempo de ejecución del sistema, así que los cálculos relativos a la composición de la aplicación, incluyendo la planificación de mensajes o de tareas deberán ser computacionalmente sencillos para evitar, en primer lugar, una carga excesiva en la entidad compositora y, en segundo lugar, reducir los tiempos necesarios para la aceptación de una nueva aplicación en el sistema.

La entidad compositora elegirá aquella combinación que maximice una determinada figura de mérito que tendrá en cuenta requisitos extremo a extremo funcionales y no funcionales impuestos por el usuario o por el propio sistema. Las diferentes figuras de mérito definidas se presentarán en el apartado 5.5. La búsqueda de la mejor combinación se hará en profundidad, eligiendo una determinada combinación de perfiles antes de estudiar su planificabilidad y obtener su figura de mérito.

Algoritmo 5.1 Algoritmo de composición exhaustivo

$[exito, P_{new}, Comb_{best}, Fig_{best}] = Compongo(n, i, P_{Actual}, Comb_{now}, \mathcal{A}_D, \mathcal{P}_f, \mathcal{F}_i)$
{
  if $n <> 0$
  /* Incluye su indice en la combinación que se está evaluando */
    $Comb_{now}(n) = i$
  if $n = ultimo$ **then**
  /* Si es el último nivel, es decir, una hoja del grafo. */
    let $Comb_{best} = Comb_{now}$
    /* Calcula la figura de mérito global de la rama completa */
    /* y la nueva planificación */
    $[exito, Fig_{best}, P_{new}, nodosInv] := calcPlanyFigM(1, Comb_{best}, \mathcal{A}_D, \mathcal{F}_i, P_{Actual})$
    if exito
      /* Comprueba planificabilidad en los nodos involucrados */
      foreach nodo in nodosInv
        $[AppsModificadas, P_{new}] := CompruebоPlanificabilidad(nodo)$
      end foreach
      /* Comprueba planificación y restricciones aplicaciones afectadas */
      foreach app in $AppsModificadas$
        $exito := ReplanificoAplicacion(app, P_{new})$
      end foreach
    end if
  else
    if $n = 0$
      $\mathcal{P}_f := EliminoPerfilesNoAdecuados(\mathcal{P}_f, \mathcal{A}_D)$
    let $Comb_{best} := [\,]$; $Fig_{best} := \infty$; exito:=**false**
    /* Para cada perfil posible del siguiente nivel */
    foreach p in $\mathcal{P}_f(n+1)$
      /* Compruebo su compatibilidad con el perfil actual */
      if Compatibles(p,i)
        $[e, P_{calc}, comb, f] = Compongo(n+1, p, P_{Actual}, Comb_{now}, \mathcal{A}_D, \mathcal{P}_f, \mathcal{F}_i)$
        if e **and** $f < Fig_{best}$
          /* Si hubo éxito y la figura de mérito calculada es mejor que la actual */
          let $Fig_{best} = f$; $Comb_{best} = comb$; $P_{new} = P_{calc}$
      end if
    end foreach
  end if
  return $(exito, P_{new}, Comb_{best}, Fig_{best})$
}

Una petición de un cliente al algoritmo de composición consiste en la definición de una aplicación deseada, $\mathcal{A}_D$, y la selección de una determinada figura de mérito $\mathcal{F}_i$ de entre las disponibles en el sistema.

La definición de la aplicación deseada, $\mathcal{A}_D = (G_p, \{G_s^i, CR_s^i\}, CS_{req})$, incluye un grafo principal y un conjunto de subgrafos opcionales. Las ramas paralelas de estos grafos serán ejecutadas concurrentemente, producto de la transmisión de un mensaje en *multicast* o *broadcast*. A efectos de la composición, en un nodo don-

de pueden elegirse dos caminos: seguir el grafo principal o un subgrafo opcional, la elección se hará en base a las condiciones de reemplazo especificadas en el respectivo CR_s^i. Nótese que esta notación es una forma de expresar una condición `if-then-else` en el grafo principal.

El algoritmo que sigue la entidad compositora, dada una petición de un cliente y el estado actual del sistema, es el mostrado en el algoritmo 5.1, y que se explica a continuación:

1. Define el nivel 0 de la aplicación como aquel anterior al primer servicio involucrado.

2. De entre todos los posibles perfiles de los servicios involucrados descarta aquéllos cuya C_i, tiempo de ejecución en el peor caso, es mayor que el plazo deseado por la aplicación o los que la suma de su utilización, $U_i = \frac{C_i}{T}$, con la utilización actual del nodo en el que reside sea mayor que 1.

3. Enumera los nodos, servicios, del grafo completo de la aplicación, incluyendo los nodos de los subgrafos opcionales, desde el nivel 1 hasta el nivel N, que será la profundidad máxima de este grafo.

4. Partiendo del nivel 0 hasta llegar hasta el último nivel, se selecciona un camino dentro del grafo de la aplicación:

   a) Si no es el último nivel, se escoge siguiendo el grafo de la aplicación, el primer perfil del siguiente nivel. Si es el último, se sigue en el paso 5. Nótese que, aunque dos servicios se ejecuten concurrentemente, a efectos de este paso en la composición, se considerarán niveles diferentes. Si existe la posibilidad de elegir entre una rama del grafo principal y un subgrafo, se tratarán como si pertenecieran las dos posibilidades al mismo nivel, aunque tengan diferente numeración.

   b) Se comprueba su compatibilidad con el perfil escogido del nivel actual. En el caso de que el nivel actual sea el nivel 0, se comprueba su compatibilidad con las características de la entrada de la aplicación.

   c) Si son compatibles, se desciende al siguiente nivel (Paso 4*a*).

5. Una vez seleccionada una rama del grafo, se calculan los parámetros de la aplicación y, si es planificable, la figura de mérito de la rama completa.

6. Se verifica que el resto de las aplicaciones en el sistema no se vean afectadas. Para ello se comprueba si los tiempos de respuesta en el peor caso de las tareas preexistentes en los nodos involucrados se modifican. En el caso de que lo hagan, se comprueba que este cambio no merma las prestaciones ofrecidas a las aplicaciones afectadas.

Algoritmo 5.2 Algoritmo para el cálculo de la figura de mérito y los parámetros de las tareas y mensajes de la aplicación

$[exito, Fig, P_{new}, nodosInv] := calcPlanyFigM(primera, Comb_{best}, \mathcal{A}_D, \mathcal{F}_i, P_{Actual})$
{
  if primera **then**
    $P_{new} := AsignoPrioridades(Comb_{best}, \mathcal{A}_D, P_{Actual})$
    $[exito, P_{new}] = CalculoPlanificacionRed(P_{new}, Comb_{now}, \mathcal{A}_D)$
    if exito **then**
      $[exito, P_{new}, nodosInv] = CalculoPlanificacionTareas(P_{new}, Comb_{now}, \mathcal{A}_D)$
    end if
    if not exito
      /* Ya sea por la no planificabilidad de la red, de algún nodo del sistema */
      /* o porque alguna tarea tiene un WCRT superior al Plazo */
      return (**false**, ∞, P_{Actual}, [])
  end if
  foreach accion in $Comb_{best}$
    $P_{new} = CalculoOffset(accion, Comb_{best})$
    if Offset >Plazo Aplicacion
      return (**false**, ∞, P_{Actual}, [])
    if accion.siguiente=paralelos **then**
      let offsetSig=Offset
      foreach paralelo in accion.siguiente.paralelos
        $[exito, f, P_{new}] := calcPlanyFigM(0, paralelo, \mathcal{A}_D, \mathcal{F}_i, P_{new})$
        if not exito
          return (**false**, ∞, P_{Actual}, [])
        elseif offsetParalelo>offsetSig
          offsetSig=offsetParalelo
      end foreach
      $AsignoOffsetSiguiente(P_{new}, offsetSig)$
    end if
  end foreach
  $Fig := CalculoFigMerito(\mathcal{F}_i, Comb_{best}, P_{new})$
  return (**true**, $Fig, P_{new}, nodosInv$)
}

7. Se devuelve al nivel padre de la hoja la nueva planificación, la figura de mérito y la combinación.

8. El nivel padre seguirá iterando sobre todos sus hijos (Paso 4*a*), seleccionando en cada iteración la mejor combinación con respecto a la figura de mérito. Esta figura de mérito y combinación, será la que devuelva a su nivel padre, a excepción del nivel 0. Cuando el nivel 0 termina de iterar sobre todos sus hijos, ya ha obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura de mérito dada y la nueva planificación del sistema.

El algoritmo seguido para el cálculo de la figura de mérito y los parámetros de las tareas y mensajes se muestra en el algoritmo 5.2. El algoritmo aplica una

planificación holística basada en una aproximación gobernada por tiempo.

La asignación de los parámetros temporales se basa, en primer lugar, en el cálculo del tiempo de respuesta en el peor caso de las tareas y mensajes involucrados en la aplicación. Si la red o alguno de los nodos físicos no es planificable, la combinación que se está evaluando no lo será. Si son planificables, se realiza la comprobación de que la suma de los tiempos de respuesta en el peor caso de todas las acciones involucradas sea menor que el plazo deseado por el usuario.

Una vez calculados los tiempos de respuesta en el peor caso, se asignan los desplazamientos temporales a las acciones involucradas, siguiendo la aproximación presentada en el apartado 3.4.3, donde se supone que el sistema se estructura en ciclos, y el desplazamiento temporal de cada acción es un múltiplo entero de la duración de dicho ciclo. Para calcular el desplazamiento temporal en un punto de sincronización, donde llegan mensajes de distintas ramas que se ejecutan concurrentemente, se aplicará la extensión al modelo definida en el apartado 3.4.4.

Como se puede intuir, al tratarse de un algoritmo exhaustivo de búsqueda en profundidad, su complejidad es elevada, pues deberá evaluar todas las posibles combinaciones de perfiles existentes en el sistema. A continuación, se analizan las distintas fuentes de complejidad de este algoritmo:

- *Número de combinaciones a evaluar*: si cada servicio involucrado en una aplicación, S_i, tiene un número de perfiles asociados $\mathcal{N}(S_i)$ y el grafo a evaluar tiene una profundidad máxima de n niveles, es decir, pueden componer la aplicación hasta n servicios diferentes, el algoritmo deberá evaluar hasta un máximo de $\mathcal{N}(\mathcal{C}) = \prod_{i=1}^{n} \mathcal{N}(S_i)$ combinaciones.

  Así, el número de combinaciones a evaluar crece exponencialmente con el número de niveles que posea una aplicación. Por ejemplo, si el número de perfiles de cada servicio disponibles en el sistema es 2, el encontrar la combinación óptima, dada una figura de mérito, para una aplicación compuesta por 10 niveles, implica evaluar 1024 combinaciones. Si el número de perfiles de cada servicio disponibles es 3, deberán evaluarse 59049 combinaciones, y si es 4, 1048576.

- *Tipo de test de planificabilidad*: como se deben calcular los tiempos de respuesta tanto de las acciones de la aplicación a componer, como de las acciones presentes en los recursos compartidos por éstas, el tipo de test de planificabilidad que se presentó como más adecuado es el basado en tiempos de respuesta. En comparación con los tests basados en utilización, este tipo de tests es menos pesimista pero suele ser de mayor complejidad. Nótese, sin embargo, que en el paso 2 del algoritmo de composición se realiza una comprobación de la planificabilidad basada en utilizaciones para descartar perfiles no adecuados, lo que reduce el número de combinaciones a evaluar.

- *Necesidad de recalcular la planificabilidad de todos los nodos físicos involucrados*: relacionado con el punto anterior. Si las combinaciones a evaluar

pueden involucrar como máximo a m nodos físicos, el número máximo de invocaciones al test de planificabilidad será $m * \mathcal{N}(\mathcal{C})$ veces.

- *Recálculo para las aplicaciones implicadas de sus tiempos de respuesta*: no introduce tanta complejidad como los puntos anteriores. Al ser cada replanificación una suma iterativa con comprobaciones sucesivas, la invocación de la replanificación es de complejidad $O(m{\cdot}n)$, siendo n el número de aplicaciones afectadas y m el número de acciones de cada una de ellas. Esta replanificación no se efectúa en todas las ocasiones, pero en el peor caso, se ejecutaría $\mathcal{N}(\mathcal{C})$ veces, tantas como combinaciones a evaluar.

Si la composición se realiza antes de la ejecución del sistema, la complejidad computacional del algoritmo puede ser asumible. Sin embargo, la composición también deberá poder realizarse en tiempo de ejecución, por lo que disponer de un algoritmo de composición computacionalmente más sencillo es deseable, aunque las combinaciones que ofrezca sean subóptimas, siempre que emplee un tiempo razonable y acotado en obtenerlas.

5.4. Algoritmo de Composición Mejorado

Tal y como se analizó en el apartado anterior, el algoritmo de composición tiene una complejidad muy elevada, dependiente del número de combinaciones que, a su vez, aumenta exponencialmente con el número de niveles de la aplicación. El número de niveles de una aplicación deseada no se puede modificar, pero sí se puede acotar el número de combinaciones evaluadas. En este apartado, se presenta una mejora del algoritmo de composición basada en la disminución del número de ramas a evaluar mediante el uso de heurísticos de poda.

El algoritmo de composición mejorado aplicará una figura de mérito relativa a los perfiles de cada nivel de la aplicación, en función de la cual ordenará los perfiles de ese nivel de mejor a peor. Acto seguido, dividirá dicho conjunto ordenado en subconjuntos, para, en cada nivel en el que se aplique la poda, seleccionar un número determinado de subconjuntos de perfiles, de tal manera que sólo evaluará un número determinado de ramas del grafo total. La elección del número de perfiles de cada nivel a partir del cual se aplica la poda, el tamaño de los subconjuntos y el número de subconjuntos que se evaluarán depende del heurístico empleado. Por otra parte, la figura de mérito relativa empleada está fuertemente vinculada a la figura de mérito global. En el siguiente apartado se se presentarán distintas figuras de mérito globales y sus figuras de mérito relativas para la aplicación del algoritmo mejorado.

En el algoritmo 5.3 se presenta la fase de preparación, ejecutada por el nivel 0:

1. En primer lugar, como en el algoritmo exhaustivo, se eliminan los perfiles no adecuados mediante la comprobación de que su tiempo de ejecución en el peor caso sea menor al plazo deseado de la aplicación y que su instanciación

Algoritmo 5.3 Fase de preparación previa del algoritmo de composición mejorado

```
[P_f, nPerfSubc, nMaxSubc] = PreparoPoda(P_f, A_D, F_i)
{
  P_f := EliminoPerfilesNoAdecuados(P_f, A_D)
  /* Calcula las figs relativas de todos los perfiles y */
  /* la media y desviación típica de los perfiles de cada nivel */
  [P_f, μ_s, σ_s] := CalculoFigMeritoRelativas(P_f)
  foreach nivel in A_D
    let nMaxSubc(nivel):=1
    let nPerfSubc(nivel) := size(P_f(nivel))
    if nPerfSubc(nivel) ≥ nivelPoda and
            (σ_s(nivel)/μ_s(nivel) > valorDesvio or nPerfSubc(nivel) ≥ limNumPerf)
    /* Hay poda si el num de Perfiles es mayor que una cota y */
    /* sus figs de mérito relativas tienen una dispersión mayor que un valor dado */
        /* Num perfiles en cada bloque */
        nPerfSubc(nivel):=calculoCantidad()
        /* Num max bloques a evaluar */
        nMaxSubc(nivel):=calculoNumMaxSubcEval()
    end if
  end foreach
}
```

en el nodo físico al que pertenecen no sobrepase una utilización superior a la unidad. Nótese que éste es un test de planificabilidad adecuado si el nodo físico emplea como política de planificación EDF, siendo una cota superior si la política de planificación es RM. Si se supiera exactamente la política de planificación empleada por el nodo, podría emplearse una cota basada en utilizaciones más ajustada para poder eliminar los perfiles no adecuados previamente a realizar el test de planificabilidad basado en tiempos de respuesta.

2. Para todos los posibles perfiles se calcula la figura de mérito relativa. Nótese que el cálculo de la figura de mérito relativa es independiente para cada perfil del sistema. Es más, si se utilizase siempre la misma figura de mérito global para todo el sistema, podría realizarse este cálculo previamente a la ejecución del algoritmo, ya sea al inicio de la ejecución del sistema, ya sea, en un sistema dinámico donde se permita la inclusión de nuevos perfiles en tiempo de ejecución, cuando aparezca/desaparezca un nuevo perfil en el sistema.

3. Se calcula para cada nivel la media (μ_s) y la desviación típica (σ_s) de las figuras de mérito relativas de los perfiles disponibles. La relación entre la desviación típica y la media da una idea de la dispersión de los valores. En un sistema en el que los valores estén poco dispersos, es difícil decidir cuáles son los más adecuados para realizar la composición. En este sentido, se decidió

que frente a un conjunto de perfiles poco dispersos no se realizase la poda. Sin embargo, cuando el número de niveles o el número de perfiles es elevado, podría forzarse a una poda, aunque los perfiles estén poco dispersos, quedando este punto a criterio del gestor del sistema.

4. Posteriormente, el heurístico concreto que se utilice dividirá el conjunto total de los perfiles del sistema en subconjuntos o bloques, que pueden ser de tamaño variable. Dentro de cada bloque, los perfiles estarán ordenados de mejor a peor, al igual que cada bloque con respecto al conjunto de bloques total.

5. En el algoritmo mejorado se evaluarán en primer lugar los perfiles del primer bloque y, en el caso de no encontrar una combinación válida, se evaluarían los perfiles del siguiente bloque. Si se permitiese realizar la búsqueda en todos los bloques, cuando no existe una combinación posible, este algoritmo se comportaría como el exhaustivo. Para evitarlo, se acota el número de bloques que se evaluarán a un valor máximo, $nMaxSubc$.

La figura de mérito relativa deberá ser escogida cuidadosamente respecto a la figura de mérito global empleada para minimizar el error respecto a la combinación hallada por el exhaustivo.

En el algoritmo 5.4 se presenta el algoritmo de composición mejorado. Este algoritmo sigue la misma estructura que el exhaustivo. A continuación se explica el algoritmo:

1. En el nivel 0 prepara la poda realizando una llamada al algoritmo 5.3.

2. En cada nivel:

   a) Se inicializa el número de bloques evaluados del siguiente nivel a 1 y el número máximo de bloques a evaluar a $nMaxSubc(n+1)$. Si un nivel hermano ya ha encontrado una combinación válida (si Fig_{now} tiene un valor distinto de infinito), sólo se evaluará un bloque del siguiente nivel.

   b) Se evaluarán bloques mientras el número de bloques evaluados sea menor que el número máximo de bloques a evaluar, o mientras no se haya encontrado una combinación válida.

   c) Se extraerán los perfiles correspondientes al bloque a evaluar.

   d) Para cada perfil, se comprobará su compatibilidad con el anterior y si son compatibles se descenderá al siguiente nivel, paso 2*a*.

3. Una vez seleccionada una rama del grafo (último nivel), se calcularán los parámetros temporales de la aplicación y, si es planificable, la figura de mérito completa. La función $CalculoPlanificacionTareas$ se debe modificar para que devuelva el nivel exacto donde se produce el error.

Algoritmo 5.4 Algoritmo de composición mejorado

$[exito, err, P_{new}, Comb_{best}, Fig_{best}] = CompOpt(n,i, P_{Actual}, Comb_{now}, \mathcal{A}_D, \mathcal{P}_f, \mathcal{F}_i, Fig_{now}, nPerfSubc, nMaxSubc)$

```
{
  if n <> 0 then Comb_now(n) = i
  if n = ultimo then
    let Comb_best = Comb_now
    /* En errDonde indicará si el error fue por pérdida de plazo y dónde */
    [exito,errDonde,Fig_best,P_new, nodosInv]=calcPlanyFigM(1,Comb_best,
                                                  A_D, F_i, P_Actual)
    if exito
      foreach nodo in nodosInv
        [exito, AppsModificadas, P_new] := CompruebоPlanificabilidad(nodo)
      end foreach
      if AppsModificadas > 0
        if Fig_now = ∞ and exito then
        /* Sólo si alguna hoja hermana anterior no encontró una combinación válida */
          foreach app in AppsModificadas
            exito := ReplanificoAplicacion(app, P_new)
          end foreach
        else Fig_best = ∞
    end if ; end if ; end if
  else
    if n = 0 then
      [P_f, nPerfSubc, nMaxSubc] = PreparoPoda(P_f, F_i)
    let Comb_best := [ ]; Fig_best := ∞; exito:=false
    /* Inicializa el num de bloque en que está y el num max de bloques que evaluará */
    let numBloque:=1; numMaxBloques:=nMaxSubc(n+1)
    if Fig_now < ∞ then
      /* Si algún subconjunto hermano encontró una combinación válida, */
      /* sólo evalúa el primer bloque */
      numMaxBloques=1
    while Fig_best = ∞ or numBloque < numMaxBloques
      let SubP = Extraigo(numBloque, nPerfSubc(n + 1), P_f(n + 1))
      foreach p in SubP
        if Compatibles(p,i)
          [e, errDonde, P_calc, comb, f] = CompOpt(n+1, p, P_Actual, Comb_now,
                                                   A_D, P_f, F_i, Fig_best,
                                                   nPerfSubc, nMaxSubc)
          if (not e) and errDonde ≤ n then
            /* Si se pierde el deadline en este nivel o en uno superior */
            return (false, errDonde, P_actual, [ ], ∞)
          if e and f < Fig_best
            let Fig_best = f; Comb_best = comb; P_new = P_calc
        end if
      end foreach
      LET numBloque++
    endwhile
  end if
  return (exito, errDonde, P_new, Comb_best, Fig_best)
}
```

En el caso de que no sea planificable la aplicación completa, no se puede asegurar cuál de todas las acciones es la responsable, ya que como pueden estar en el mismo recurso la falta de planificabilidad no es culpa de una tarea en concreto, sino de la combinación. En este caso, se devuelve como nivel en el que se produce el error el último. Sin embargo, en el caso de que se pierda el plazo en un perfil de un nivel en concreto, se puede asegurar que ninguna combinación a partir de ese nivel será adecuada, devolviendo la función el nivel donde se produce el error.

4. Si la aplicación es planificable, se comprueba la planificabilidad de las tareas preexistentes en todos los nodos involucrados.

5. Si alguna aplicación ya existente en el sistema se ve afectada, sólo se realiza la replanificación si ninguna hoja hermana ha encontrado una figura de mérito válida.

6. Devuelve a su nivel padre la figura de mérito encontrada para dicha combinación.

7. En el nivel padre, si una rama hija devuelve error, determinando que éste se ha producido en un nivel superior o igual al del nivel padre, no se siguen evaluando más combinaciones y se devuelve el error a su padre respectivo.

8. En el caso de que no haya error, o de que el error se produzca en un nivel inferior, se seguirán evaluando perfiles (paso 2*b*) del bloque actual.

9. Una vez se terminan de evaluar los perfiles de un bloque, si se han encontrado una o varias combinaciones posibles, se devuelve la mejor combinación encontrada. Si todavía quedan bloques a evaluar, se vuelve al paso 2*b*. En otro caso, si se ha llegado al número máximo de bloques a evaluar, se devuelve al nivel superior que no se ha encontrado una combinación válida.

Esta mejora al algoritmo de composición exhaustivo reduce el número de combinaciones, en un factor determinado por el heurístico que se emplee. La proximidad de la solución obtenida mediante este método a la solución óptima que ofrece el algoritmo exhaustivo, depende del heurístico empleado y de la adecuación de la figura de mérito relativa a la figura de mérito global.

5.4.1. Heurísticos desarrollados

En este apartado se presentan como ejemplo distintos heurísticos desarrollados durante la elaboración de la presente tesis. Posteriormente, en la sección de evaluación se analizarán los comportamientos de cada uno de ellos para las distintas figuras de mérito globales y relativas que se presentarán en el siguiente apartado.

5.4.1.1. Primera combinación válida

Este heurístico refleja el deseo por parte de un cliente de que el sistema escoja la primera combinación válida que encuentre. El nivel de poda en este caso será igual a 2. En este caso la constante `valorDesvio` deberá forzar a que se haga poda en cualquier caso, así que el heurístico asignará un valor de desvio nulo. El número de perfiles de cada bloque será igual a 1, y el número máximo de bloques a evaluar la mitad del número de perfiles en dicho nivel más 1:

$$nMaxSubc(n) = \left\lfloor \frac{\mathcal{N}(S_i)}{2} \right\rfloor + 1 \tag{5.1}$$

Los perfiles de cada nivel estarán en ordenados en orden creciente de su valor de figura de mérito relativa, así que, en realidad, este heurístico realiza la composición, en primer lugar de los mínimos de las figuras de mérito relativas para cada nivel.

En el mejor caso, el heurístico sólo evaluará una combinación de todas las posibles, $\mathcal{N}_{evaluo}^{mejor} = 1$, mientras que, en el peor caso, evaluará:

$$\mathcal{N}_{evaluo}^{peor} = \prod_{i=1}^{n} \left\{ \left\lfloor \frac{\mathcal{N}(S_i)}{2} \right\rfloor + 1 \right\} \tag{5.2}$$

Este número puede reducirse disminuyendo el número de bloques a evaluar.

5.4.1.2. Tamaño de bloque fijo

Dado un tamaño de bloque fijo, especificado por la variable `cantidad`, este heurístico realiza la poda siempre que el número de perfiles sea mayor que el tamaño de bloque especificado sin tener en cuenta el valor de la dispersión entre perfiles del mismo nivel. En este caso se evaluarán bloques de tamaño fijo para cada nivel, $cantidad = c$ perfiles en cada nivel. El número máximo de bloques a evaluar para cada nivel será la mitad de los bloques en los que se divide el nivel más 1:

$$nMaxSubc(n) = \left\lfloor \frac{\left\lceil \frac{\mathcal{N}(S_i)}{c} \right\rceil}{2} \right\rfloor + 1 \tag{5.3}$$

En el mejor caso, el número de combinaciones a evaluar será:

$$\mathcal{N}_{evaluo}^{mejor} = \prod_{i=1}^{n} \text{mín}\{\mathcal{N}(S_i), c\} \tag{5.4}$$

mientras que, en el peor caso el número de combinaciones a evaluar será:

$$\mathcal{N}_{evaluo}^{peor} = \prod_{i=1}^{n} \{c \cdot nMaxSubc(n)\} \tag{5.5}$$

5.4.1.3. Uso de la dispersión como medida para realizar o no poda

En muchas ocasiones, la dispersión entre los valores de la figura de mérito relativa no es lo suficientemente grande como para discernir entre cuáles son los mejores perfiles en cada nivel. Se ha introducido en el algoritmo mejorado una medida de la dispersión mediante la inspección de la relación entre la desviación típica y la media para cada nivel. Si dicho valor es superior a una cota dada por `valorDesvio` y, además, se supera el número de perfiles en cada nivel marcado por `nivelPoda`, se realiza la poda. Aunque no se alcance este valor de desvío, si el número de perfiles en el nivel supera un determinado valor, `limNumPerf`, se fuerza a que se realice la poda, pues, si no se incluyese esta condición, en el peor caso pueden tener que evaluarse todas las combinaciones.

5.4.1.4. Tamaño de bloque variable entre niveles

Dado que no todos los niveles presentarán la misma dispersión entre valores, puede darse el caso de que, para un determinado nivel sea necesario evaluar un número elevado de perfiles, debido a que la dispersión es baja, mientras que en otro, con dispersión elevada, un número más reducido es suficiente. En este caso, para cada nivel se calculará el tamaño de bloque de tal forma que el tamaño se deduzca a partir de la cercanía de los valores más bajos de las figuras de mérito a la media:

$$c_i = size(\mathcal{P}_s); \mathcal{P}_s = \{p \in \mathcal{P}(S_i) \mid f(p) \leq \mu_f - 0{,}5\sigma_f\} \tag{5.6}$$

Si la cantidad calculada es menor a un determinado valor, por ejemplo $c_i = 1$ ó $c_i = 2$, puede forzarse a que el tamaño de bloque alcance un mínimo predeterminado, para permitir la evaluación de un número superior de valores. Si la cantidad calculada es excesivamente grande, por ejemplo, mayor que la mitad de los perfiles en un determinado nivel, siendo el número de perfiles superior a una cota, también deberá forzarse a que el tamaño de bloque sea inferior, porque se corre el riesgo de que el algoritmo mejorado alcance la misma complejidad que el exhaustivo.

Nótese que, en este caso, el tamaño de bloque es fijo para cada nivel, siendo variable entre estos.

5.5. Figuras de Mérito

Las figuras de mérito aplicadas a la composición de una aplicación deseada pueden reflejar:

- Deseos del cliente sobre el comportamiento de la aplicación.
- Requisitos impuestos por parte del administrador del sistema sobre el comportamiento general de éste y de las aplicaciones.

- Ser una combinación de los dos anteriores.

Por ejemplo, pueden preferirse aquellas combinaciones que minimicen o maximicen el número de nodos involucrados, aquéllas que minimicen la utilización global del sistema o que equilibren la carga.

Cuando existe una determinada política con respecto al sistema, impuesta por el administrador del mismo, ésta ha de estar contemplada en todas las figuras de mérito que se apliquen. Por ejemplo, una política de sistema puede ser el equilibrado y la minimización de la utilización en todos los nodos. Las figuras de mérito que se utilicen para realizar la composición, aunque incluyan otros términos como pueden ser la minimización o maximización del plazo deseado,o la calidad de servicio medida por un determinado parámetro ofrecida por los perfiles (por ejemplo, la calidad de imagen ofrecida o el número de filtros aplicados por un determinado servicio) deben reflejar siempre este requisito del administrador, pues la instanciación de una aplicación cuya composición que no lo contemple puede provocar que el sistema se desequilibre, es decir, que ya no responda a la política del sistema.

Nótese que las replanificaciones contempladas en el algoritmo de composición, tanto en la versión exhaustiva como en la mejorada, al poder modificar los tiempos de respuesta de las acciones, pueden provocar que las aplicaciones dejen de ajustarse a la política empleada en el sistema. Si la política responde a un deseo de minimización de la utilización en los nodos y/o a un equilibrado de dicha utilización, como la utilización depende del tiempo de ejecución en el peor caso y no del tiempo de respuesta, no se vería modificada. Sin embargo, si lo que se pretende es que todas las aplicaciones minicen el tiempo de respuesta extremo a extremo, una replanificación de una determinada aplicación puede provocar que ya no se cumpla esta restricción. El administrador del sistema será el encargado de determinar la tolerancia del sistema frente a las replanificaciones.

Los deseos del cliente deberán ajustarse a la política preestablecida del sistema y, si entran en conflicto con ésta, deberá ponderarse si es permisible una relajación en la política del sistema para aceptar a dicho cliente, o si se utiliza una figura de mérito que, ofreciendo una solución subóptima para el cliente, sí refleje dicha política.

Por otra parte, si se emplea el algoritmo de composición mejorado, deberá escogerse una figura de mérito relativa adecuada a la figura de mérito global, de tal forma que la figura de mérito global de la combinación ofrecida por el algoritmo mejorado esté aceptablemente próxima a la figura de mérito de la combinación óptima, que sería la que calcularía el algoritmo exhaustivo.

5.5.1. Figuras de Mérito desarrolladas para el Algoritmo de Composición

En este apartado se presentan distintas figuras de mérito, tanto globales como relativas, que se desarrollaron durante la elaboración de la presente tesis para la

composición de servicios compuestos. El algoritmo de composición escogerá como mejor combinación aquélla cuya figura de mérito sea la de menor valor. Las figuras de mérito globales propuestas pueden combinarse entre ellas, ponderándolas y ajustando la figura de mérito relativa a esa ponderación, en el caso de aplicar el algoritmo mejorado.

5.5.1.1. Minimización del tiempo de respuesta extremo a extremo

La figura de mérito desarrollada para la minimización del tiempo de respuesta se ha relacionado con el plazo deseado de la aplicación. Así, para una combinación $\mathcal{C}$ determinada de perfiles:

$$\mathcal{F}_{minRe2e}(\mathcal{C}) = \frac{R_{e2e}}{D_{deseado}} \tag{5.7}$$

La figura de mérito relativa, como no se pueden relacionar los plazos extremo a extremo, y todavía no se conocen los tiempos de respuesta (por desconocer cuál será exactamente la distribución de las tareas entre los nodos) será la relación entre el tiempo de ejecución en el peor caso de cada perfil y el plazo deseado por la aplicación:

$$\mathcal{F}^r_{minRe2e}(p) = \frac{C_p}{D_{deseado}} \tag{5.8}$$

5.5.1.2. Maximización del tiempo de respuesta extremo a extremo

En algunos entornos se puede buscar la maximización del tiempo de respuesta extremo a extremo. Por ejemplo, para dar tiempo a un servicio en concreto a ejecutarse previamente. En este caso, la figura de mérito desarrollada es similar a la anterior:

$$\mathcal{F}_{maxRe2e}(\mathcal{C}) = 1 - \frac{R_{e2e}}{D_{deseado}} \tag{5.9}$$

La figura de mérito relativa será entonces:

$$\mathcal{F}^r_{maxRe2e}(p) = 1 - \frac{C_p}{D_{deseado}} \tag{5.10}$$

Nótese que el empleo de esta figura de mérito relativa puede provocar que muchas de las combinaciones evaluadas no sean adecuadas por pérdida de plazo extremo a extremo. En este caso, a la hora de preparar la poda, el cálculo de los subconjuntos adecuados del siguiente nivel debería depender de los del conjunto anterior, indicando cuál es el máximo tiempo de respuesta aceptable para evaluar dicho conjunto.

5.5.1.3. Minimización/maximización del número de nodos físicos involucrados

Consiste en una sencilla suma contando el número de nodos físicos involucrados. Como no se puede saber *a priori*, en cada nivel, el número de nodos involucrados en toda la aplicación, esta figura de mérito global no tiene figura de mérito relativa asociada. Puede emplearse en asociación con otras figuras de mérito globales, ponderadas por un factor dependiendo de su importancia, como política de sistema o por deseo expreso del cliente.

5.5.1.4. Minimización de la utilización media

Esta figura de mérito tiene en cuenta la utilización media en los nodos físicos para seleccionar una combinación determinada.

Sea $\mathcal{C}$ una combinación de perfiles de n niveles. Cada perfil p estará ubicado en un nodo físico $N(p)$, pudiendo estar distintos perfiles de la combinación localizados en el mismo nodo. $Ninv(\mathcal{C})$ será el conjunto de los nodos involucrados en la combinación a evaluar, siendo N su número. Denotaremos como $U_i^{pre}(p)$ a la utilización del nodo i previa a la instanciación del perfil p, y $U_i(\mathcal{C})$ a la utilización en el nodo i posterior a la instanciación de la totalidad de la combinación $\mathcal{C}$ en el sistema.

Si se considera que ningún perfil comparte ubicación con otro, la figura de mérito a minimizar será la utilización en los nodos involucrados posterior a la instanciación de la combinación, dividida entre el número de niveles, n, que tenga dicha combinación que, en este caso, coincide con el número de nodos involucrados:

$$\begin{aligned}\mathcal{F} &= \frac{1}{n} \cdot \sum_{\forall\, i \in N_{inv}(\mathcal{C})} U_i(\mathcal{C}) \\ &= \frac{1}{n} \cdot \sum_{\forall\, p \in \mathcal{C}} \left(\frac{C_p}{T_{deseado}} \right) + \frac{1}{N} \sum_{\forall\, i \in Ni(\mathcal{C})} U_i^{pre}(p)\end{aligned} \tag{5.11}$$

El factor $1/n = 1/N$ se añade para poder comparar combinaciones con distinto número de niveles (debido a la presencia de subgrafos opcionales en la especificación de la aplicación deseada).

Cuando se permite que los perfiles compartan ubicación, la minimización de la utilización puede enfocarse desde diferentes perspectivas que, a su vez generan distintas figuras de mérito:

- **Minimización de la utilización en los nodos involucrados**: se considera la media de las utilizaciones en los nodos físicos involucrados una vez instanciada la combinación. La figura de mérito a emplear será:

$$\mathcal{F}_{minUnodos} = \frac{1}{N} \cdot \sum_{\forall\, i \in N_{inv}(\mathcal{C})} U_i(\mathcal{C}) \tag{5.12}$$

Esta figura de mérito penaliza aquellas combinaciones que presenten un número bajo de nodos físicos involucrados.

- **Minimización de la utilización con respecto a los perfiles involucrados**: se realiza la media tomando el valor de la utilización una vez instanciada la combinación en el nodo físico en el que está localizado cada perfil, independientemente de que existan perfiles compartiendo un mismo nodo físico. La figura de mérito a emplear será:

$$\mathcal{F}_{minUperf} = \frac{1}{n} \cdot \sum_{\forall\, p \in \mathcal{C}} U_{N(p)}(\mathcal{C}) \tag{5.13}$$

Con esta figura de mérito se trata en términos de igualdad a combinaciones con distinto número de nodos involucrados, pues por cada perfil de la combinación se suma la utilización final del nodo en el que está ubicado, homogeneizando de esta manera el sistema. Así, se compara en términos de igualdad una combinación cuyos perfiles estén todos ubicados en diferentes localizaciones, con una combinación que posea perfiles ubicados en el mismo nodo físico.

- **Minimización de la utilización de la combinación**: Las dos figuras de mérito anteriores no realizan distinción entre combinaciones cuya media de utilización sea la misma. En el caso en el que se desee realizar esta distinción, se ha definido una figura de mérito que tiene en cuenta las utilizaciones previas a la instanciación en cada nodo, de tal manera que penaliza aquellas combinaciones cuyos perfiles de niveles inferiores tengan un tiempo de ejecución elevado. Esta figura de mérito se elaboró como una medida de *benchmark*, razonable en términos teóricos.

  La utilización previa a la instanciación del perfil, $U_i^{pre}(p)$ que se calculará de la siguiente manera:

$$U_i^{pre}(p) = \begin{cases} U_i^{pre}(\mathcal{C}) & \text{Si } p \text{ es el 1er perfil de } \mathcal{C} \text{ instanciado en el nodo } i \\ U_i^{pre}(\mathcal{C}) + \sum_{\forall j \in m} \frac{C_j}{T_{deseado}} & \text{Si existen } m \text{ predecesores de } p \text{ instanciados en el nodo } i \end{cases} \tag{5.14}$$

  Siendo $U_i^{pre}(\mathcal{C})$, la utilización en el nodo i antes de la instanciación de la combinación $\mathcal{C}$.

  Así, la figura de mérito global será

$$\mathcal{F}_{minUrama} = \frac{1}{n} \sum_{\forall\, p\, in \mathcal{C}} \left(U_i^{pre}(p) + \frac{C_p}{T_{deseado}} \right) \tag{5.15}$$

$\mathcal{C}$	$(N_1; U_1(\mathcal{C}))$	$\frac{C_1}{T}$	$(N_2; U_2(\mathcal{C}))$	$\frac{C_2}{T}$	$(N_3; U_3(\mathcal{C}))$	$\frac{C_3}{T}$	$\mathcal{F}_U^{nodos}$	$\mathcal{F}_U^{per}$	$\mathcal{F}_U^{rama}$
1	$(1; 0{,}25)$	0,25	$(2; 0{,}1)$	0,1	$(3; 0{,}3)$	0,3	$0{,}65/3$	$0{,}65/3$	$0{,}65/3$
2	$(1; 0{,}35)$	0,25	$(1; 0{,}35)$	0,1	$(3; 0{,}3)$	0,3	$0{,}65/2$	$1/3$	$0{,}9/3$
3	$(2; 0{,}25)$	0,25	$(1; 0{,}4)$	0,1	$(1; 0{,}4)$	0,3	$0{,}65/2$	$1{,}05/3$	$0{,}75/3$
4	$(1; 0{,}55)$	0,25	$(2; 0{,}1)$	0,1	$(1; 0{,}55)$	0,3	$0{,}65/2$	$1{,}2/3$	$0{,}9/3$
5	$(1; 0{,}65)$	0,25	$(1; 0{,}65)$	0,1	$(1; 0{,}65)$	0,3	0,65	0,65	$1{,}25/3$
6	$(1; 0{,}1)$	0,1	$(2; 0{,}4)$	0,4	$(3; 0{,}4)$	0,4	0,3	0,3	$0{,}9/3$
7	$(1; 0{,}1)$	0,1	$(2; 0{,}4)$	0,3	$(2; 0{,}4)$	0,1	0,25	0,3	$0{,}8/3$
8	$(1; 0{,}1)$	0,1	$(2; 0{,}4)$	0,2	$(2; 0{,}4)$	0,2	0,25	0,3	$0{,}7/3$
9	$(1; 0{,}1)$	0,1	$(2; 0{,}4)$	0,1	$(2; 0{,}4)$	0,3	0,25	0,3	$0{,}6/3$
10	$(1; 0{,}3)$	0,1	$(1; 0{,}3)$	0,1	$(1; 0{,}3)$	0,1	0,3	0,3	$0{,}6/3$

Cuadro 5.1: Ejemplos comparación entre figuras de mérito basadas en utilización

Estas tres figuras de mérito globales se comportan de manera diferente tal y como se muestra en la tabla 5.1, donde se muestran dos conjuntos de ejemplos de combinaciones de perfiles sobre nodos en vacío ($U_i^{pre}(\mathcal{C}) = 0$). La elección de una figura de mérito u otra dependerá de los deseos del administrador de sistema, pudiéndose combinar entre ellas a través de, por ejemplo, una combinación lineal.

La figura de mérito basada en considerar únicamente la utilización en los nodos no diferencia entre combinaciones que presenten el mismo número de nodos y cuya suma de utilizaciones global sea la misma, aunque los nodos presenten una utilización final diferente (en la tabla, columna 8, combinaciones 2–4 y 7–9). El uso de la figura de mérito basada en considerar las utilizaciones finales de nodo físico para cada perfil distingue en el caso de que la suma de la utilización final sea la misma y las utilizaciones en los nodos sean diferentes (en la tabla, columna 9, combinaciones 2–4). Sin embargo, en el caso de que las utilizaciones finales en los nodos sean la misma, la figura de mérito resultante es idéntica (en la tabla, combinaciones 6–9), o cuando la media de utilizaciones finales da el mismo resultado (combinaciones 6–10).

La figura de mérito basada en la suma acumulativa de utilizaciones en la rama, sí realiza la distinción entre combinaciones con el mismo número de nodos y utilizaciones finales (en la tabla, columna 11, combinaciones 6–9), penalizando las combinaciones en las que los perfiles de mayor C_i son instanciados antes. Sin embargo, también adolece de no distinguir entre una combinación con distinto número de nodos pero misma suma acumulada de utilizaciones (combinaciones 9 y 10: $f_9 = (0{,}1 + 0{,}1 + 0{,}4)/3$ y $f_{10} = (0{,}1 + 0{,}2 + 0{,}3)/3$; y 2 y 4) y, en algunos casos, entre combinaciones con distinto número de nodos (combinación 6, de 3 nodos, con combinaciones 2 y 4 de 2 nodos).

Las tres combinaciones pueden combinarse con dos figuras de mérito relativas diferentes. Como en un nivel superior no se tiene conocimiento de la utilización final en los nodos, pero sí del tiempo de ejecución en el peor caso de los perfiles, una figura de mérito relativa simple es aquella basada en la relación entre este tiempo,

C_i, y el período deseado, $T_{deseado}$:

$$\mathcal{F}_U^{r1}(p) = \frac{C_p}{T_{deseado}} \tag{5.16}$$

Esta figura relativa no tiene en cuenta la utilización en los nodos en los que se instancia la aplicación. Dado que cada perfil tiene asociada unívocamente una ubicación y se puede saber *a priori* la $U_i^{pre}(\mathcal{C})$, es posible desarrollar una figura de mérito que tenga en cuenta la carga previa en cada nodo:

$$\mathcal{F}_U^{r2}(p) = U_p^{pre}(\mathcal{C}) + \frac{C_p}{T_{deseado}} \tag{5.17}$$

Nótese que ambas figuras son equivalentes si todos los nodos físicos tienen la misma carga.

5.5.1.5. Equilibrado de carga entre los nodos involucrados

El uso de las figuras de mérito anteriores basadas en la minimización de la utilización, aseguran que la combinación elegida, en el caso de emplear el algoritmo exhaustivo, tiene asociada una utilización final en los nodos baja. Si se emplea en toda la actividad del sistema dicha figura de mérito, y los servicios involucrados en las aplicaciones están uniformemente distribuidos, el sistema tenderá a una uniformización de la carga. Sin embargo, en determinados casos, se puede desear que, minimizando la utilización, los nodos involucrados tengan una utilización similar. Es decir, que frente a dos combinaciones posibles de utilizaciones finales para una combinación $\mathcal{C}_1 = \{0{,}6; 0{,}1\}$ y $\mathcal{C}_2 = \{0{,}3; 0{,}35\}$, que serían equivalentes si se emplea una figura de mérito basada en utilizaciones, se escoja siempre aquélla cuyos nodos tengan una utilización similar (o la condición inversa, aquélla cuya utilización esté desbalanceada). Una medida de la dispersión de los valores la aporta la utilización de la varianza o de la desviación típica. Si se emplea la desviación típica normalizada por el valor de la media de las utilizaciones, puede compararse la dispersión de distintos conjuntos de valores sin importar el valor de la utilización. El empleo de la desviación típica sin normalizar sirve para, además, dar una medida no sólo de la dispersión, sino también de los valores medios de la utilización.

En la tabla 5.2 se presentan, para el mismo conjunto de ejemplos de la tabla 5.1, los valores de la varianza y de la varianza normalizada asociada a cada figura de mérito basada en utilización. Puede observarse que, en determinados casos en los que la figura de mérito no permite discernir qué combinación es la más adecuada, la inspección de la varianza normalizada puede ayudar a realizar la elección.

En el caso de la figura de mérito basada en las utilizaciones finales de los nodos $\mathcal{F}_{Unodos}$, en los que si la media de las utilizaciones finales era la misma, a efectos de la figura de mérito las combinaciones eran equivalentes (En la tabla combinaciones 2–4, 6 y 10, y 7–9). Si las utilizaciones en los nodos son diferentes (combinaciones 2–4 y 6 y 10), la varianza normalizada puede ayudar a elegir una combinación (ya

$\mathcal{C}$	$\mathcal{F}_U^{nodos}$	σ^2	$\frac{\sigma^2}{\mu^2}$	$\mathcal{F}_U^{per}$	σ^2_{Uper}	$\frac{\sigma^2_{Uper}}{\mu^2_{Uper}}$
1	0,65/3	0,01083	0,23077	0,65/3	0,01083	0,23077
2	0,65/2	0,00125	0,01183	1/3	0,00083	0,0075
3	0,65/2	0,01125	0,10651	1,05/3	0,0075	0,06122
4	0,65/2	0,10125	0,95858	1,2/3	0,0675	0,42187
5	0,65	–	–	0,65	0	0
6	0,3	0,03	0,33333	0,3	0,03	0,33333
7	0,25	0,045	0,72	0,3	0,03	0,33333
8	0,25	0,045	0,72	0,3	0,03	0,33333
9	0,25	0,045	0,72	0,3	0,03	0,33333
10	0,3	–	–	0,3	0	0

	$\mathcal{F}_U^{rama}$	σ^2_{Urama}	$\frac{\sigma^2_{Urama}}{\mu^2_{Urama}}$
1	0,65/3	0,01083	0,23077
2	0,9/3	0,0025	0,02778
3	0,75/3	0,0225	0,36
4	0,9/3	0,0525	0,58333
5	1,25/3	0,04333	0,2496
6	0,9/3	0,03	0,33333
7	0,8/3	0,02333	0,32812
8	0,7/3	0,02333	0,42857
9	0,6/3	0,03	0,75
10	0,6/3	0,01	0,25

Cuadro 5.2: Ejemplos comparación entre figuras de mérito basadas en el uso de la desviación típica entre utilizaciones.

sea minimizando o maximizando la dispersión entre las utilizaciones de los nodos). Sin embargo, si las utilizaciones finales en cada nodo no varían entre combinaciones (en la tabla, combinaciones 7–9), aunque se utilice una figura de mérito combinando utilización y varianza, todas éstas serán equivalentes. Comportamiento similar presenta la figura de mérito basada en utilizaciones finales de nodos para cada perfil ($\mathcal{F}_U^{per}$). Con respecto a la figura de mérito basada en la suma acumulativa de utilizaciones en la rama, el uso de la varianza normalizada puede ayudar a discernir entre combinaciones cuando la suma acumulada es la misma, pero los valores a sumar son diferentes.

El uso de la varianza o varianza normalizada sólo se ha contemplado en esta tesis como parte ponderada de una figura de mérito global compuesta por varios términos, entre ellos un término que minimice la utilización.

5.6. Validación experimental

En este apartado se presenta la validación experimental realizada a través de simulación de los algoritmos propuestos mediante la implementación de un prototipo de la Entidad Compositora, abstrayendo todos los demás elementos de la arquitec-

tura. Dicho prototipo se realizó sobre *Scilab*, entorno de programación matemático similar a *Matlab*, desarrollado por INRIA y otros, y distribuido con licencia de código libre.

5.6.1. Descripción del prototipo

En la figura 5.4 se muestran los módulos implementados en el prototipo.

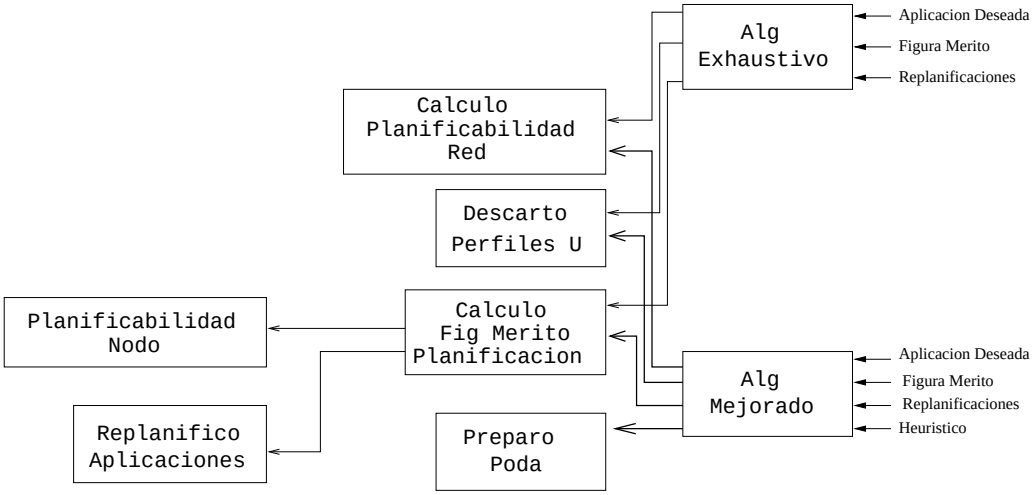


Figura 5.4: Módulos implementados en el prototipo

La funcionalidad implementada por cada uno es la siguiente:

- *Algoritmo Exhaustivo*: implementa el algoritmo explicado en el apartado 5.3, que se muestra en el Algoritmo 5.1. Recibe como parámetros la estructura de la aplicación deseada, así como los nombres de los servicios que la compondrán, su período, su plazo deseado y la figura de mérito que se aplicará y si se admite o no reconfiguraciones de las aplicaciones existentes en el sistema.

- *Algoritmo Mejorado*: implementa el algoritmo explicado en el apartado 5.4, que se muestra en el Algoritmo 5.4. Recibe los mismos parámetros que el algoritmo exhaustivo y además el heurístico concreto que se utilizará.

- *Preparo Poda*: implementa la preparación para el algoritmo mejorado, que se muestra en el Algoritmo 5.3.

- *Descarto Perfiles U*: descarta de los perfiles de los perfiles de los servicios involucrados aquéllos cuya C_i es mayor que el plazo deseado por la aplicación o los que la suma de su utilización $U_i = \frac{C_i}{T_{app}}$, con la utilización actual del nodo en el que reside sea mayor que la unidad, tal y como se explica en el paso 2 del Algoritmo Exhaustivo y en el paso 1 de la fase de preparación del Algoritmo Mejorado.

- *Cálculo Figura de Mérito y Planificación*: implementa el cálculo de la figura de mérito y los parámetros de las tareas, tal y como se muestra en el Algoritmo 5.2.

- *Planificabilidad Nodo*: calcula la nueva planificabilidad en un nodo, dado un conjunto de tareas a instanciar en éste. Si el conjunto de tareas es planificable, devuelve los tiempos de respuesta en el peor caso de cada tarea instanciada. Para calcular la planificabilidad en un nodo, se suponen plazos inferiores a los períodos y los nodos se planifican suponiendo que el instante crítico ocurre en $t = 0$. El algoritmo implementado es el propuesto por Audsley [17, 18], explicado en el capítulo del Estado del Arte (véanse ecuaciones 2.20 y 2.21).

- *Replanifico Aplicaciones*: si se permiten replanificaciones, recalcula los parámetros de las aplicaciones afectadas, tal y como se explica en el paso 6 del Algoritmo Exhaustivo y en el paso 5 del Algoritmo Mejorado.

- *Cálculo Planificabilidad Red*: como se considera la red un recurso único, donde la planificación se realiza de manera semejante a la de los nodos, considerando que el instante crítico se produce en $t = 0$, este cálculo puede realizarse previamente a la ejecución de los algoritmos de composición. En la implementación del prototipo se ha supuesto una red de tiempo real gobernada por tiempo, que sigue el paradigma FTT y se ha implementado un algoritmo de análisis de planificabilidad basado en tiempos de respuesta, suponiendo que se planifica la red con RM. El algoritmo implementado se muestra en el Algoritmo 5.5. Tanto este algoritmo como su demostración pueden encontrarse en [176].

Todos estos módulos consultan la información del sistema que está estructurada de la siguiente manera:

- *Conjunto de nodos*: guarda información sobre todos los nodos físicos del sistema. Para cada nodo del sistema:

  - Conjunto de perfiles que residen en él. Para cada perfil guarda el identificador de servicio y su tiempo de ejecución en el peor caso.
  - Utilización actual del nodo.
  - Tareas instanciadas. Para cada tarea guarda sus parámetros temporales: $\tau_i = (C_i, T_i^{app}, D_i^{app}, p, R_i, D_i^{local})$, donde D_i^{local} es el plazo local que debe cumplir cada tarea para generar el mensaje que tiene asociado y D_{app} es el plazo de la aplicación a la que pertenece.

- *Mensajes en el sistema*: guarda información sobre los mensajes que se intercambian en la red. Para cada mensaje guarda sus parámetros temporales: $\tau_i^m = (C_i^m, T_i^m, D_i^m, R_i^m)$

Algoritmo 5.5 Análisis de planificabilidad de la red implementado [176]

$[R, exito] = CalculoPlanificabilidadRed(ConjuntoMensajes, Deadlines, LSW)$
{
let FinalAlg=max(Deadlines(ConjuntoMensajes)
foreach i in 1:FinalAlg **let** lsw(i)=0
foreach $mensaje$ in ConjuntoMensajes
let R(mensaje):=0 ATxAhora(mensaje):=**true**
end foreach
let n:=1
while $n \leq FinalAlg$ **and** R(numMensajes)=0
foreach $mensaje$ in ConjuntoMensajes
if ATxAhora(mensaje)=**true**
if $lsw(n) + C(mensaje) \leq LSW$
let $lsw(n) = lsw(n) + C(mensaje)$; ATxAhora(mensaje):=**false**
if R(mensaje)=0
let R(mensaje):=n
if R(mensaje) $>$ Deadline(mensaje)
return $(R,$ **false**$)$
end if
end if
end if
end if
if modulo(n,Periodos(mensaje))==0
let ATxAhora(mensaje):=0
end if
end foreach
let n=n+1
endwhile
return $(R,$ **true**$)$
}

- *Aplicaciones existentes*: guarda la información sobre todas aplicaciones existentes en el sistema. Para cada aplicación existente en el sistema:
  - Estructura de su grafo de servicios, incluyendo los paralelos existentes.
  - Período de la aplicación.
  - Plazo extremo a extremo deseado.
  - Tiempo de respuesta extremo a extremo conseguido.
  - Vector con los parámetros temporales de las tareas que la componen, incluyendo dentro de la información asociada los nodos en los que están instanciadas.
  - Las aplicaciones que la instanciación de esta aplicación ha modificado.
- *Servicios existentes*: guarda la información sobre todos los servicios existentes en el sistema. Para cada servicio guarda el conjunto de nodos en los que

residen perfiles que implementan dicho servicio.

5.6.2. Metodología de simulación

El objetivo principal de las simulaciones es la evaluación del comportamiento del algoritmo mejorado con respecto al exhaustivo.

La no exploración de conjuntos de combinaciones posibles implica una mejora en términos del tiempo de ejecución, pero puede implicar que la figura de mérito global de la combinación de perfiles de servicio elegidos esté muy alejada de la figura de la combinación óptima. Cuando en este apartado se emplea el término *óptimo*, siempre se refiere a la solución ofrecida por el algoritmo exhaustivo, pues éste al explorar todas las posibles combinaciones puede, en función de una determinada figura de mérito global, elegir la mejor.

Por otra parte, el comportamiento del algoritmo mejorado depende:

- Del heurístico empleado, pues en función de éste explorará un número de combinaciones u otro.
- De las figuras de mérito relativas empleadas, pues cada una de ellas tendrá, por una parte, un coste en términos de ejecución y, por otra, la figura de mérito relativa determinará qué combinaciones se explorarán, lo que influirá en el posible error cometido.

Así, en la evaluación, se eligió un subconjunto de las distintas figuras de mérito globales y sus respectivas relativas propuestas, y un subconjunto de los heurísticos desarrollados:

- Las figuras de mérito evaluadas fueron:

  1. Una figura de mérito combinada basada en las utilizaciones por rama, la varianza de dichas utilizaciones y la minimización del tiempo de respuesta:

     $$\mathcal{F}(\mathcal{C}) = 0{,}6 \cdot \mathcal{F}_{minRe2e}(\mathcal{C}) + 0{,}3 \cdot \mathcal{F}_{minUrama}(\mathcal{C}) + 0{,}1 \cdot \sigma^2_{Urama}(\mathcal{C}) \quad (5.18)$$

     A esta figura de mérito combinada se le hizo corresponder la siguiente figura de mérito relativa:

     $$\mathcal{F}^r(p) = \mathcal{F}^{r2}_U(p) + \mathcal{F}^{r2}_U(p) + \mathcal{F}^r_{minRe2e}(p) \quad (5.19)$$

  2. Minimización del tiempo de respuesta extremo a extremo:

     $$\mathcal{F}_{minRe2e}(\mathcal{C}) = \frac{R_{e2e}}{D_{deseado}} \quad (5.20)$$

     y la figura de mérito relativa empleada fue:

     $$\mathcal{F}^r_{minRe2e}(p) = \frac{C_p}{D_{deseado}} \quad (5.21)$$

3. Minimización de la media de utilizaciones por perfil:

$$\mathcal{F}_{minUperf} = \frac{1}{n} \cdot \sum_{\forall\, p \in \mathcal{C}} U_{N(p)}(\mathcal{C}) \tag{5.22}$$

y se empleó como figura de mérito relativa:

$$\mathcal{F}_U^{r2}(p) = U_p^{pre}(\mathcal{C}) + \frac{C_p}{T_{deseado}} \tag{5.23}$$

- Los heurísticos que se evaluaron fueron los siguientes:

1. Primera combinación válida.
2. Tamaño de bloque fijo, con tamaños de bloque: 2, 3, 4 y 5.
3. Tamaño de bloque variable.

La evaluación se centró en:

- El comportamiento temporal del algoritmo exhaustivo en función del número de combinaciones.
- El comportamiento temporal del algoritmo mejorado para distintos heurísticos.
- El error cometido en la solución con respecto a la óptima, debido a emplear el algoritmo mejorado.
- La calidad de las combinaciones exploradas por el algoritmo mejorado con respecto a la totalidad de las combinaciones posibles.

Para analizar el comportamiento temporal, se evaluó el caso mejor, donde siempre existe una combinación óptima, a través de la simulación con los nodos completamente descargados, y el peor caso, con los nodos soportando una carga elevada. En este análisis no importa tanto el número exacto de perfiles o de nodos presentes en el sistema, como el número de combinaciones posibles, $\mathcal{N}(C)$, que se deben evaluar. Evidentemente, el número de combinaciones posibles depende directamente del número de perfiles de los servicios que se utilizarán y la planificabilidad de éstos depende de cómo están distribuidos los perfiles en los nodos. Para variar el número de combinaciones posibles se varió el número de nodos presentes en el sistema y de número de perfiles totales existentes.

Las pruebas se estructuraron en función de la figura de mérito global empleada. En cada iteración se realizaba la composición de una aplicación deseada, especificada por su estructura. En primer lugar, se empleaba el algoritmo exhaustivo, midiendo su tiempo de ejecución y almacenando la figura de mérito de la combinación óptima encontrada y las figuras de mérito asociadas a todas las posibles combinaciones. A

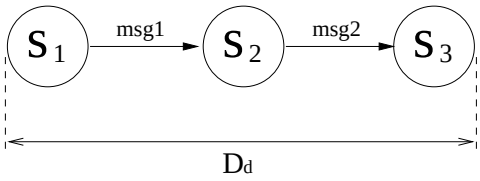


Figura 5.5: Estructura de la aplicación deseada empleada en los experimentos

continuación, para la misma configuración inicial se ejecutaban los algoritmos mejorados a evaluar, almacenando para cada uno de ellos su tiempo de ejecución, la figura de mérito de la combinación encontrada y las figuras de mérito asociadas a todas las combinaciones que exploró durante su ejecución. Este proceso se repetió al menos 10 veces para cada valor de $\mathcal{N}(C)$.

Se barrió para todos los experimentos el rango de combinaciones posibles $\mathcal{N}(C) \in [2, 10000]$, y se analizaron los tiempos y el número de combinaciones exploradas, $\mathcal{N}_{evaluo}$, por tramos. Para obtener valores significativos de los tiempos de ejecución y del número de combinaciones exploradas por los distintos heurísticos del algoritmo mejorado, para cada número de combinaciones posibles se realizaron entre 10 y 20 iteraciones, obteniendo la media y la desviación típica de los valores a medir.

En algunos casos, se realizaron experimentos en los que se permitió $\mathcal{N}(C) > 10000$, con un menor número de iteraciones por valor de $\mathcal{N}(C)$, como comprobación de que el comportamiento temporal para todos los algoritmos se mantenía. Todos estos experimentos son costosos en tiempo de ejecución, pues en el algoritmo exhaustivo, para obtener un solo valor se debían realizar al menos 10 iteraciones y si el número de combinaciones posibles es alto, por ejemplo, 5000, el algoritmo tarda, dependiendo de la máquina donde se ejecute, entre 40 y 50 segundos en encontrar la combinación óptima, lo que multiplicado por, como mínimo, 10 iteraciones representa aproximadamente 9 minutos para hallar un solo valor de un experimento, todo esto sin contar con el tiempo de ejecución de los algoritmos mejorados. Así, se decidió tener más resolución en valores bajos, explorando más combinaciones, y menos en valores elevados, presentando las gráficas de tiempos mediante interpolación.

Las pruebas se realizaron con la estructura de la aplicación deseada que se muestra en la figura 5.5, con 3 servicios en secuencia.

Las pruebas se realizaron en 3 máquinas distintas, una por cada figura de mérito evaluada, para comprobar cómo se comporta el algoritmo en diferentes entornos, con distinta capacidad de procesamiento. Debido a este hecho, los tiempos de ejecución de las diferentes figuras de mérito no pueden ser comparados directamente entre sí, aunque sí el número de combinaciones exploradas, pues éste no depende de las características del *hardware* empleado. Las características de dichas máquinas se muestran en la tabla 5.3. Para las pruebas de la figura de mérito combinada se empleó la máquina C, para las pruebas relativas a la figura de mérito de minimización del

tiempo de respuesta extremo a extremo se empleó la máquina B y para las de la figura de mérito de minimización de la utilización, la máquina A.

Máquina	Modelo	Sistema Operativo	CPU	Memoria RAM
A	Mobile Intel Pentium 4 M	Windows XP	1800MHz	256MB
B	AMD Atholon XP 2400++	Windows XP	795MHz	512MB
C	Intel Pentium M	Linux 2.6.11-686	1600MHz	1024MB

Cuadro 5.3: Características hardware de las máquinas donde se realizaron las simulaciones

5.6.3. Análisis y presentación de resultados

Los resultados se presentarán en función de la figura de mérito global empleada. Para cada una de ellas, se realizó el experimento, en primer lugar, con los nodos descargados y, en segundo lugar, con una carga alta en cada nodo de tal forma que ninguna de las combinaciones posibles fuese planificables y, por lo tanto, no se pudiese encontrar una combinación óptima. Así, se pudo evaluar el comportamiento de los algoritmos en el mejor y en el peor caso. Además se realizó un experimento adicional, consistente en la composición de la aplicación deseada en un sistema con un valor de la media de utilización los nodos medio-bajo: $\bar{U} \simeq 0{,}2$

Cada heurístico explora un subconjunto de todas las combinaciones posibles. El tamaño del subconjunto lo determina el heurístico, mientras que las combinaciones exploradas son determinadas por la figura de mérito relativa. En las tablas 5.4, 5.8 y 5.12 se analiza el comportamiento de las figuras de mérito relativas en función de la calidad de las combinaciones exploradas. Así, para el conjunto de combinaciones posibles que explora el algoritmo exhaustivo, se calcula a qué tramo pertenecen las exploradas por el mejorado. Puede observarse que, para valores de número de combinaciones posibles, $\mathcal{N}(C)$, bajos, la peor de las figuras de mérito exploradas por el mejorado está en algunos casos, hasta en el $95\,\%$ mejor de las exploradas por el exhaustivo. A medida que el número de combinaciones posibles sube, el conjunto de combinaciones exploradas se refina, llegando, por ejemplo en la figura de mérito combinada para el heurístico de la primera combinación válida (tabla 5.4) a ser parte del $0{,}1\,\%$ mejor.

Este resultado refleja que las combinaciones exploradas forman parte de las mejores de todas las posibles, es decir, que la figura de mérito relativa escoge bien el subconjunto de combinaciones a explorar.

Por otra parte, se analizó el error cometido debido al uso de los distintos heurísticos. Este error se definió como la diferencia entre la figura de mérito hallada por el heurístico y la figura de mérito óptima hallada por el exhaustivo:

$$e = 100 * \left(1 - \frac{f_{mej}}{f_{exh}}\right) \tag{5.24}$$

Se analizaron los errores medio ($\bar{e}$), máximo (e_{max}) y la media de error cuando se produce error ($\bar{e}_e$), para las distintas figuras de mérito:

- Para la figura de mérito combinada, tabla 5.5, puede observarse que los valores de error medio y máximo son, en general, bajos lo que indica que la figura de mérito relativa se adapta bien a la figura de mérito global empleada. El error medio cometido en porcentaje por emplear distintos heurísticos se muestra en la figura 5.6. Por otra parte, se analizó el comportamiento de la figura de mérito relativa en un entorno con una utilización media en los nodos, manteniéndose su buen comportamiento con respecto al error medio cometido tal y como se muestra en la tabla 5.6.

- La figura de mérito basada en la minimización del tiempo de respuesta extremo a extremo, tabla 5.9, presenta valores superiores de error para el mejor caso, y tal y como se puede observar en la figura 5.8. La adaptación de la figura de mérito relativa no es de la misma calidad que en el caso anterior. En el estudio del comportamiento de la figura de mérito relativa con una utilización media en los nodos baja, ofreció los mismos resultados. En la tabla 5.10 se muestra el valor del error medio, máximo y condicionado a error (media de error cuando se produce error), para distinto número de combinaciones respecto a la figura de mérito. Esta tabla sólo refleja los valores que llegan a la misma solución (combinación posible o no posible) que el algoritmo exhaustivo. En los experimentos realizados, la única figura de mérito que, en este caso, no encontraba una combinación válida existiendo una posible era ésta: para el heurístico de la primera combinación válida, un $7\,\%$ no obtenía una solución; para el heurístico de tamaño de bloque 2, el porcentaje era de un $0{,}9\,\%$. En el resto de los heurísticos no se producía este error.

- En la figura de mérito basada en utilizaciones se observan errores medios elevados para el heurístico consistente en buscar la primera combinación válida, tanto para el mejor caso (tabla 5.13) como para el escenario de utilización baja 5.14. El error cometido disminuye si se permite emplear heurísticos que exploren en cada nivel un mayor número de perfiles (véase figura 5.10). Aún así, la adaptación de la figura de mérito relativa respecto a la global es mejor que la anterior.

A continuación, se presenta el comportamiento temporal del algoritmo exhaustivo. Como era de esperar, el tiempo de ejecución del algoritmo exhaustivo depende linealmente del número de combinaciones posibles y presenta el mismo comportamiento sea cual fuere el estado de la red, ya que debe explorar el mismo número de combinaciones y, aunque la posibilidad de reconfiguración está implementada en el prototipo, no se han permitido reconfiguraciones en los experimentos realizados. Si se comparan los tiempos de ejecución del algoritmo exhaustivo para las distintas figuras de mérito (figuras 5.7.1, 5.9.1, 5.11.1), puede observarse que la pendiente de la

gráfica cambia, debido principalmente a la distinta capacidad de procesamiento del *hardware* empleado, pero el comportamiento temporal no se ve apenas influenciado por la figura de mérito empleada. Nótese que el tiempo en evaluar una combinación depende directamente del número de niveles que la compongan.

A continuación, se presentan los tiempos de ejecución en el mejor caso, en base logarítmica, de los distintos heurísticos empleados por el algoritmo mejorado con respecto al algoritmo exhaustivo (figuras 5.7.2, 5.9.2 y 5.11.2). En todas las gráficas se evidencia un claro descenso del tiempo de ejecución, dependiente del algoritmo empleado y, a partir de cierto valor (en el que el número de perfiles de cada nivel es igual o superior al valor del tamaño del bloque) que el tiempo empleado permanece constante en las ejecuciones del algoritmo mejorado que emplean como la familia de heurísticos de tamaño de bloque constante.

Este resultado era esperable, pues ya se había llegado a una cota mínima analítica previamente, donde el número de combinaciones a evaluar en el mejor caso de dicho heurístico es el producto del número de perfiles de cada nivel. Así, si el número de perfiles es igual o superior al tamaño del bloque,c_{bloque}, $\mathcal{N}_{evaluo}^{mejor} = (c_{bloque})^n$, siendo n el número de niveles. Este resultado se muestra también en las tablas 5.4, 5.8 y 5.12, que muestran, entre otros datos, el número medio de combinaciones evaluadas en el mejor caso. Nótese que estos datos dependen del heurístico empleado y no de la figura de mérito que se emplee.

Se analizó posteriormente, el número de combinaciones medio que se evaluaban en el peor caso para cada heurístico (véanse tablas 5.7, 5.11 y 5.15), se puede ver que el número de combinaciones que se evalúan es inferior a las evaluadas por el algoritmo exhaustivo, pero puede ser excesivamente elevado para determinados entornos. En dichos entornos, se debe modificar el parámetro $nMaxSubc$ que aceptan los heurísticos para reducir el número de combinaciones evaluadas. Por otra parte, antes de emplear cualquiera de los heurísticos en un entorno, se deberá analizar si el compromiso entre el error cometido y la reducción del tiempo de ejecución es el adecuado, o si podrían aceptarse combinaciones más alejadas de la combinación óptima para conseguir una mayor reducción del tiempo de ejecución.

Intervalo	Tamaño de bloque fijo							
	2		3		4		5	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	7,063	69,369	10,758	81,104	15,455	91,439	18,671	95,426
$(50, 100]$	9,266	45,882	21,307	69,427	40,302	86,559	54,200	91,817
$(100, 250]$	8,815	33,833	26,302	66,957	58,259	85,952	97,377	94,074
$(250, 500]$	8,209	15,789	27	44,964	63,163	71,401	117,557	87,079
$(500, 1000]$	8	9,836	27	35,127	64,000	58,082	123,240	78,330
$(1000, 1500]$	8	6,077	27	20,166	64,000	40,387	124,781	66,240
$(1500, 2000]$	8	2,795	27	15,806	64,000	32,156	125	54,844
$(2000, 2500]$	8	1,155	27	14,112	64,000	23,069	125	46,299
$(2500, 3500]$	8	2,186	27	8,670	64,000	23,815	125	42,641
$(3500, 5000]$	8	1,297	27	4,693	64,000	12,907	125	29,177

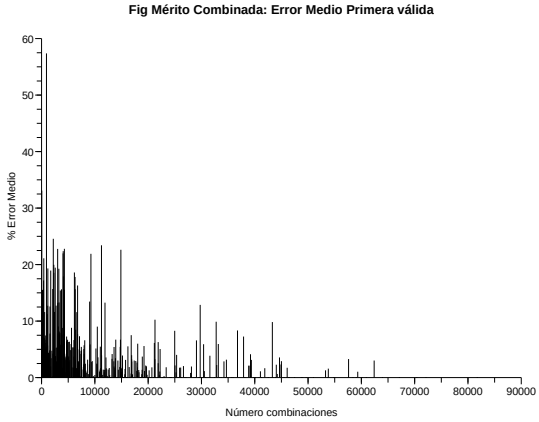
Intervalo	Tamaño variable		Primera Válida	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	10,758	81,104	4,369	52,007
$(50, 100]$	22,241	70,321	2,668	22,468
$(100, 250]$	27,129	67,885	1,485	7,044
$(250, 500]$	31,364	47,083	1,105	1,127
$(500, 1000]$	41,422	43,703	1	0,623
$(1000, 1500]$	53,486	33,720	1	0,969
$(1500, 2000]$	77,562	30,993	1	0,266
$(2000, 2500]$	99,250	32,017	1	0,151
$(2500, 3500]$	129,403	31,853	1	0,234
$(3500, 5000]$	173,250	32,908	1	0,128

Cuadro 5.4: Figura de Mérito Combinada: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo

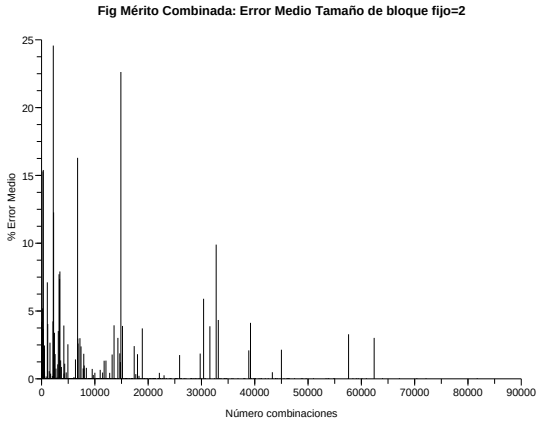
Intervalo Combinaciones	Tamaño de bloque fijo								
	2			3			4		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0,272	23,279	10,712	0	0	0	0	0	0
$(50, 100]$	0,428	8,874	5,419	0,051	2,624	1,931	0,035	2,624	2,624
$(100, 250]$	0,469	15,582	6,428	0,271	15,582	8,666	0,271	15,582	8,666
$(250, 500]$	1,127	30,604	14,230	0,001	0,140	0,140	0,001	0,140	0,140
$(500, 1000]$	0,100	7,387	2,701	0,004	0,541	0,541	0,004	0,541	0,541
$(1000, 1500]$	0,685	35,558	9,132	0,034	2,477	1,357	0	0	0
$(1500, 2000]$	0,065	2,667	1,040	0	0	0	0	0	0
$(2000, 2500]$	2,191	49,145	15,340	0,606	12,273	10,609	0,256	8,946	8,946
$(2500, 3500]$	0,764	22,109	6,419	0,209	7,737	5,858	0,026	2,143	2,143
$(3500, 5000]$	0,236	10,305	2,895	0,015	1,487	1,487	0,015	1,487	1,487
$(5000, 10000]$	0,292	16,296	4,103	0,036	4,944	2,508	0	0	0

Intervalo Combinaciones	Tamaño de bloque fijo 5			Tamaño Variable			Primera Válida		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	1,543	34,921	12,414
$(50, 100]$	0,035	2,624	2,624	0,051	2,624	1,931	3,193	29,826	11,029
$(100, 250]$	0	0	0	0,271	15,582	8,666	3,322	35,995	9,665
$(250, 500]$	0	0	0	0,001	0,140	0,140	4,590	56,755	13,636
$(500, 1000]$	0	0	0	0,004	0,541	0,541	2,973	57,380	10,291
$(1000, 1500]$	0	0	0	0	0	0	2,840	43,496	10,820
$(1500, 2000]$	0	0	0	0	0	0	2,496	34,551	7,988
$(2000, 2500]$	0,256	8,946	8,946	0	0	0	4,918	49,145	12,296
$(2500, 3500]$	0	0	0	0	0	0	3,230	31,213	9,355
$(3500, 5000]$	0	0	0	0	0	0	3,361	22,793	7,008
$(5000, 10000]$	0	0	0	0	0	0	2,185	21,913	5,623

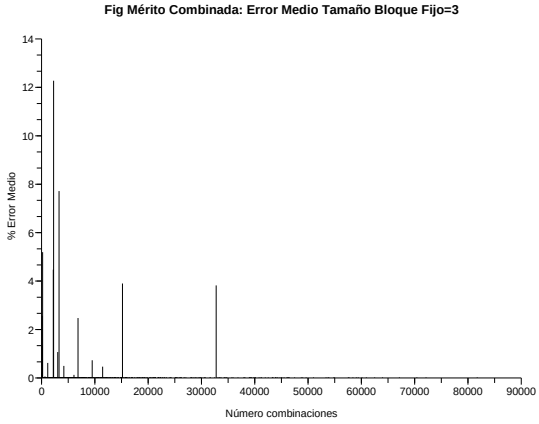
Cuadro 5.5: Figura de Mérito Combinada: Error medio, máximo y condicionado a error en porcentaje para el mejor caso.



5.6.1: Con el heurístico: Primera combinación válida



5.6.2: Con el heurístico: Tamaño de bloque fijo de tamaño=2



5.6.3: Con el heurístico: Tamaño de bloque fijo de tamaño=3

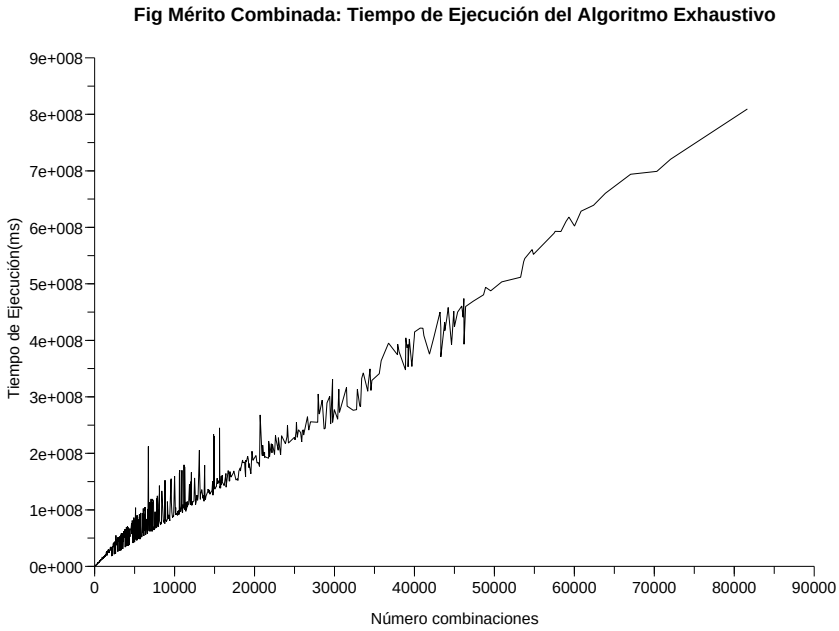
Figura 5.6: Figura de Mérito Combinada: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo.

Intervalo Combinaciones	Tamaño de bloque fijo 2			Tamaño de bloque fijo 3			Tamaño de bloque fijo 4		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0,437	63,447	14,785	0	0	0	0	0	0
$(50, 100]$	0,087	5,293	5,293	0,197	11,989	11,989	0	0	0
$(100, 250]$	1,718	72,180	21,868	0,739	72,180	17,238	0,169	14,338	5,932
$(250, 500]$	1,860	68,682	15,535	0,887	41,621	21,002	0	0	0
$(500, 1000]$	1,086	63,863	15,202	0,723	63,863	30,349	0	0	0
$(1000, 1500]$	1,443	22,496	10,929	0,386	16,464	10,228	0	0	0
$(1500, 2000]$	0,085	2,647	1,393	0,084	2,647	2,064	0,030	1,482	1,482
$(2000, 2500]$	2,881	25,754	16,875	0,688	25,754	14,104	0	0	0
$(2500, 3500]$	0,927	23,619	20,854	0,525	23,619	23,619	0,161	7,224	7,224
$(3500, 5000]$	0,547	8,199	8,199	0,547	8,199	8,199	0	0	0
$(5000, 10000]$	3,678	7,857	5,517	3,678	7,857	5,517	0	0	0
Intervalo Combinaciones	Tamaño de bloque fijo 5			Tamaño Variable			Primera Válida		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	1,529	63,447	21,533
$(50, 100]$	0	0	0	0,197	11,989	11,989	4,247	64,675	23,549
$(100, 250]$	0,147	17,459	10,319	0,707	72,180	19,805	2,810	35,790	14,048
$(250, 500]$	0	0	0	0,895	41,621	18,156	3,642	59,142	12,314
$(500, 1000]$	0	0	0	0,723	63,863	30,349	4,201	63,863	13,928
$(1000, 1500]$	0	0	0	0,141	7,457	7,457	6,470	47,298	16,330
$(1500, 2000]$	0	0	0	0,030	1,482	1,482	2,967	34,026	9,691
$(2000, 2500]$	0,014	0,556	0,556	2,041	25,754	20,922	10,880	55,940	23,477
$(2500, 3500]$	0	0	0	0,562	18,088	12,656	4,173	45,440	18,778
$(3500, 5000]$	0	0	0	0,547	8,199	8,199	5,091	26,442	15,273
$(5000, 10000]$	0	0	0	0	0	0	6,497	8,457	6,497

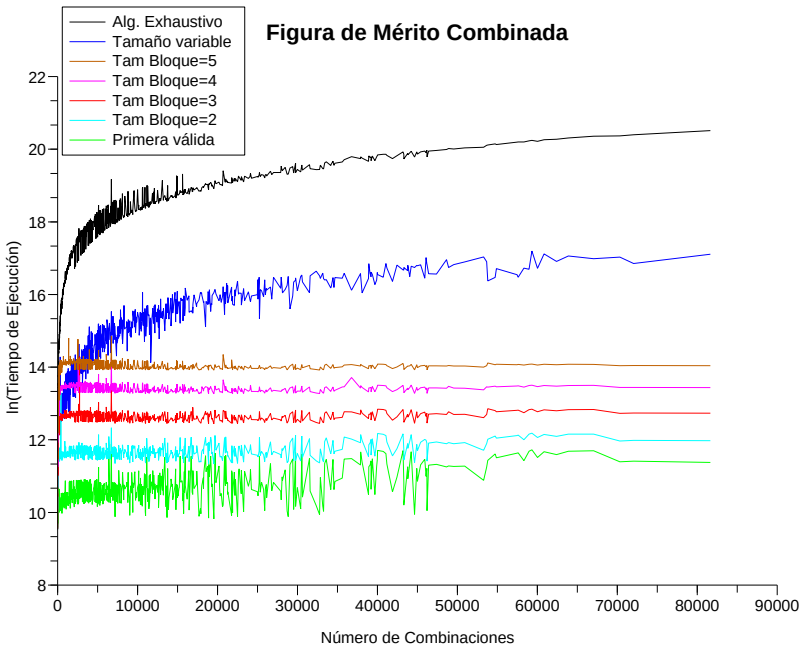
Cuadro 5.6: Figura de Mérito Combinada: Error Medio, Máximo y Error Condicionado a Error en Porcentaje con utilización media en el sistema de $\bar{U} = 0{,}2$

Intervalo	Alg Exhaustivo	Tamaño Variable	Primera Válida	Tamaño de bloque fijo 2	3	4	5
$(0, 50]$	21,192	21,192	19,409	21,165	21,192	21,192	21,192
$(50, 100]$	76,938	75,438	56,383	73,302	76,938	75,438	76,938
$(100, 250]$	173,452	171,710	93,882	155,194	173,452	171,710	173,452
$(250, 500]$	370,724	366,224	130,605	305,772	370,362	370,362	370,543
$(500, 1000]$	747,385	700,821	200,376	455,846	662,077	718,923	747,385
$(1000, 1500]$	1223,118	1112,765	288,627	587,137	920,706	1108,294	1223,118
$(1500, 2000]$	1780,067	1592,000	396,533	716,000	1106,200	1611,533	1780,067
$(2000, 2500]$	2198,750	1873,750	461,563	840	1413,750	1631,250	2198,750
$(2500, 3500]$	2950,444	2457,667	596,333	955,556	1693,000	1997,444	2950,444
$(3500, 5000]$	4253,700	3827,100	843,100	1108,800	1695,600	2060,400	4121,100

Cuadro 5.7: Figura de Mérito Combinada: Comparativa número de combinaciones exploradas para el peor caso



5.7.1: Tiempo de ejecución del algoritmo exhaustivo



5.7.2: Tiempos de ejecución para los distintos heurísticos y para el algoritmo exhaustivo

Figura 5.7: Figura de Mérito Combinada: Tiempos de ejecución en función del número de combinaciones

Intervalo	Tamaño de bloque fijo							
	2		3		4		5	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	7,067	69,182	10,763	80,280	15,461	90,245	18,676	95,037
$(50, 100]$	9,249	47,536	21,281	69,713	40,278	86,595	54,200	91,778
$(100, 250]$	8,798	34,900	26,317	68,461	58,382	86,620	97,590	94,456
$(250, 500]$	8,191	16,982	27	46,270	63,236	72,748	117,733	87,452
$(500, 1000]$	8	11,019	27	36,895	64	60,737	123,173	79,958
$(1000, 1500]$	8	7,245	27	21,739	64	43,630	124,769	69,541
$(1500, 2000]$	8	3,094	27	17,321	64	36,548	125	60,121
$(2000, 2500]$	8	1,100	27	15,261	64	25,857	125	52,418
$(2500, 3500]$	8	2,814	27	10,961	64	27,055	125	49,473
$(3500, 5000]$	8	1,350	27	4,531	64	14,465	125	33,916

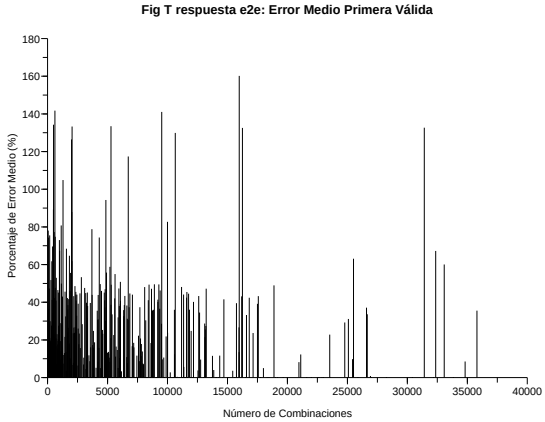
Intervalo	Tamaño variable		Primera Válida	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	10,763	80,280	4,371	51,977
$(50, 100]$	22,214	70,652	2,662	23,046
$(100, 250]$	27,217	69,199	1,475	6,978
$(250, 500]$	31,816	49,391	1,096	1,173
$(500, 1000]$	41,539	45,980	1	0,745
$(1000, 1500]$	54,245	36,814	1	1,294
$(1500, 2000]$	77,294	34,885	1	0,332
$(2000, 2500]$	98,821	35,650	1	0,214
$(2500, 3500]$	121,935	34,584	1	0,281
$(3500, 5000]$	179,571	40,824	1	0,236

Cuadro 5.8: Figura de Mérito R_{e2e}: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo

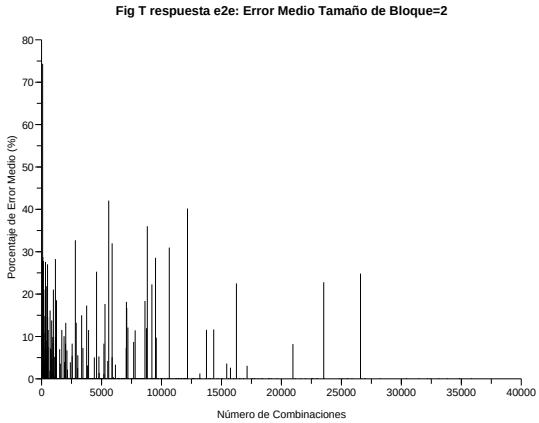
Intervalo Combinaciones	Tamaño de bloque fijo								
	2			3			4		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	0	0	0
$(50, 100]$	8,279	74,333	30,356	2,720	31,818	22,443	0	0	0
$(100, 250]$	4,459	74,121	24,081	2,056	28,740	15,860	0	0	0
$(250, 500]$	2,681	43,304	16,852	1,027	43,304	15,059	0	0	0
$(500, 1000]$	2,669	42,129	16,514	0,364	16,956	7,203	0	0	0
$(1000, 1500]$	2,484	46,675	26,491	0,181	8,783	5,777	0	0	0
$(1500, 2000]$	2,725	40,233	25,067	0,744	34,237	34,237	0	0	0
$(2000, 2500]$	1,619	33,174	9,946	0	0,012	0,012	0	0	0
$(2500, 3500]$	3,125	35,483	19,999	0,170	10,850	10,850	0	0	0
$(3500, 5000]$	1,071	34,567	15,262	0,025	2,807	2,807	0,025	2,807	2,807
$(5000, 10000]$	2,205	44,581	19,940	0,236	22,905	9,818	0,047	9,703	9,703

Intervalo Combinaciones	Tamaño de bloque fijo			Tamaño Variable			Primera Válida		
	5								
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	5,398	78,038	35,989
$(50, 100]$	0	0	0	2,720	31,818	22,443	22,301	101,961	35,044
$(100, 250]$	0	0	0	1,666	28,740	14,995	21,664	187,193	44,994
$(250, 500]$	0	0	0	0,938	43,304	16,503	20,064	134,262	38,383
$(500, 1000]$	0	0	0	0,021	2,055	2,055	16,761	141,734	37,711
$(1000, 1500]$	0	0	0	0	0	0	15,254	169,040	37,547
$(1500, 2000]$	0	0	0	0	0	0	29,404	168,133	48,307
$(2000, 2500]$	0	0	0	0	0	0	37,448	169,848	55,526
$(2500, 3500]$	0	0	0	0	0	0	13,668	76,741	30,165
$(3500, 5000]$	0	0	0	0	0	0	16,263	142,304	38,626
$(5000, 10000]$	0	0	0	0	0	0	17,450	141,081	36,297

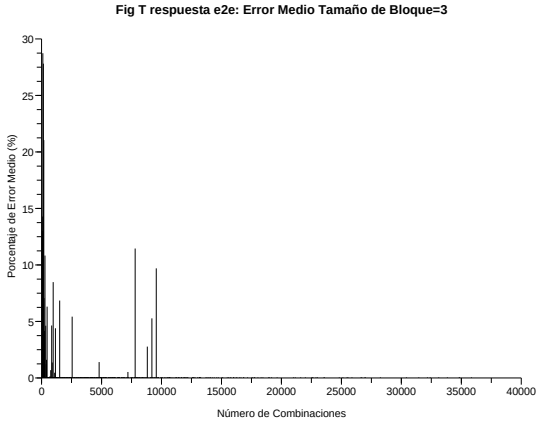
Cuadro 5.9: Figura de Mérito R_{e2e}: Error medio, máximo y condicionado a error en porcentaje para el mejor caso.



5.8.1: Con el heurístico: Primera combinación válida



5.8.2: Con el heurístico: Tamaño de bloque fijo de tamaño=2



5.8.3: Con el heurístico: Tamaño de bloque fijo de tamaño=3

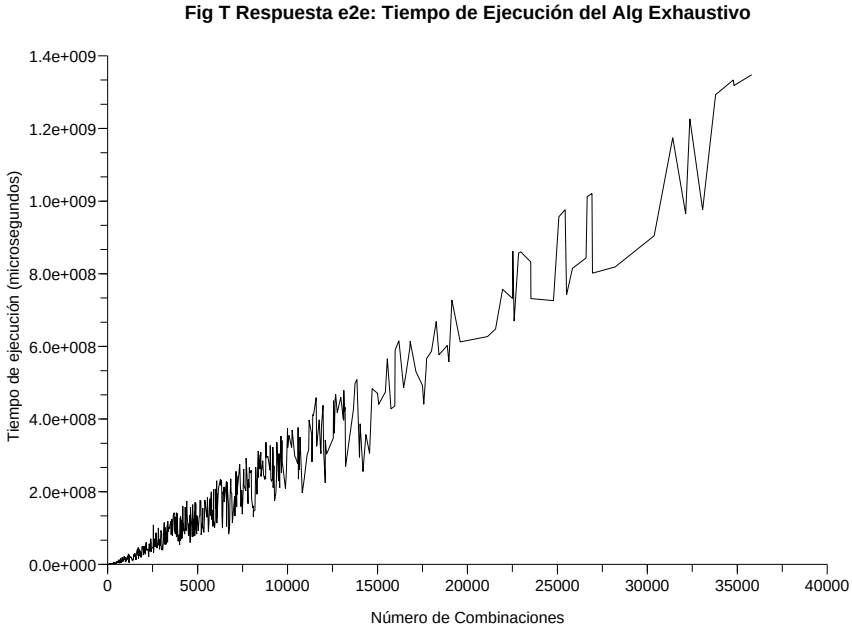
Figura 5.8: Figura de Mérito R_{e2e}: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo.

Intervalo	Tamaño de bloque fijo								
	2			3			4		
Combinaciones	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	2,503	106,077	28,197	0,299	29,475	25,287	0,125	21,099	21,099
$(50, 100]$	3,111	71,566	18,976	1,664	40,351	20,303	0,374	12,315	7,614
$(100, 250]$	3,416	38,470	15,426	3,019	77,644	16,255	3,613	77,644	21,075
$(250, 500]$	4,270	60,779	14,437	3,532	60,779	17,295	2,526	41,864	16,303
$(500, 1000]$	2,620	35,584	10,648	3,550	47,057	15,423	2,479	56,609	16,443
$(1000, 1500]$	4,550	40,790	14,989	4,069	50,137	14,241	3,072	32,919	13,231
$(1500, 2000]$	2,321	28,600	8,290	3,881	37,683	13,859	2,796	35,734	17,477
$(2000, 2500]$	9,889	62,094	22,026	6,542	62,094	17,808	7,074	62,094	23,107
$(2500, 3500]$	3,538	32,686	12,972	4,600	48,734	15,813	2,294	26,389	11,470
$(3500, 5000]$	8,515	64,299	18,341	1,352	15,334	9,461	2,968	38,285	20,773
$(5000, 10000]$	7,967	41,506	16,931	8,069	30,456	10,551	3,028	23,288	10,296
Intervalo	Tamaño de bloque fijo			Tamaño Variable			Primera Válida		
	5								
Combinaciones	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0,000	0,000	0,000	0,299	29,475	25,287	2,344	113,759	30,467
$(50, 100]$	0,202	12,315	12,315	1,657	40,351	25,276	4,664	71,566	21,886
$(100, 250]$	3,090	59,593	20,601	2,823	77,644	15,203	2,660	35,021	14,325
$(250, 500]$	2,584	41,864	18,347	2,750	60,779	19,522	3,529	63,878	13,188
$(500, 1000]$	2,354	59,593	18,535	2,462	35,584	17,231	3,276	44,103	11,467
$(1000, 1500]$	2,945	32,919	13,745	3,552	50,137	16,575	6,371	50,137	16,989
$(1500, 2000]$	1,410	26,981	14,101	5,695	63,560	21,906	3,276	24,531	8,621
$(2000, 2500]$	4,967	62,094	18,720	7,612	62,094	19,632	7,157	39,019	18,457
$(2500, 3500]$	1,548	26,400	9,462	3,970	48,734	15,597	5,905	48,734	18,042
$(3500, 5000]$	1,765	38,285	24,704	5,023	41,205	28,130	5,863	30,799	13,681
$(5000, 10000]$	2,356	20,090	13,349	3,452	30,456	19,563	9,022	40,856	12,782

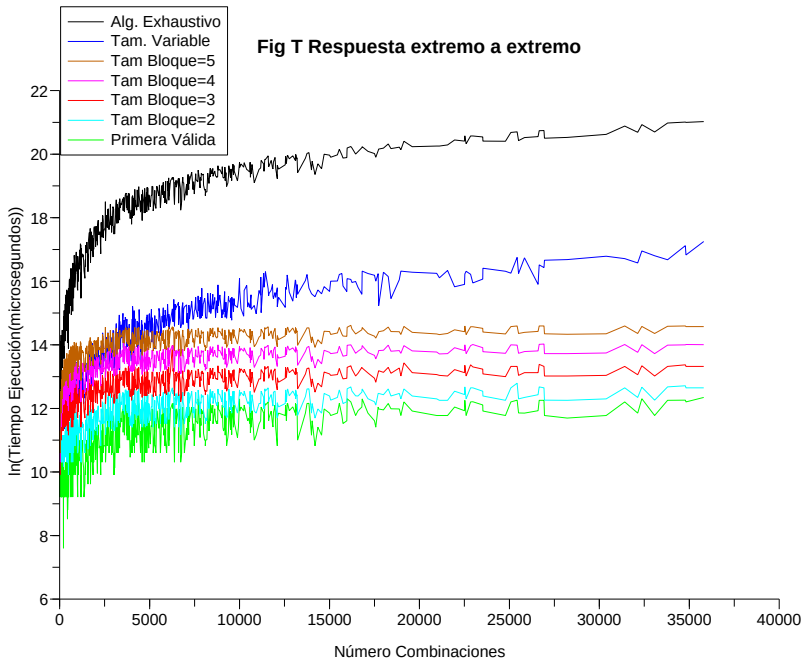
Cuadro 5.10: Figura de Mérito R_{e2e}: Error medio, máximo y condicionado a error en porcentaje para $\bar{U} = 0{,}20$.

Intervalo	Alg Exhaustivo	Tamaño Variable	Primera Válida	Tamaño de bloque fijo			
				2	3	4	5
$(0, 50]$	20,483	20,408	18,187	20,270	20,425	20,420	20,483
$(50, 100]$	74,867	73,448	53,243	70,358	73,267	74,648	73,667
$(100, 250]$	178,000	178,000	94,770	157,125	175,600	178,000	178,000
$(250, 500]$	362,343	360,629	132,083	279,808	349,543	362,343	360,629
$(500, 1000]$	747,571	713,120	205,412	447,607	664,513	736,465	747,571
$(1000, 1500]$	1255,313	1120,071	305,219	544,732	1006,107	1168,295	1255,313
$(1500, 2000]$	1697,722	1265,019	379,056	672,000	1090,167	1410,611	1697,722
$(2000, 2500]$	2225,750	1754,250	462,000	745,000	1464,000	2063,250	2225,750
$(2500, 3500]$	2881,727	2397,545	577,545	856,727	1456,364	2394,091	2797,727
$(3500, 5000]$	4037,200	3582,800	767,800	1096,000	1792,800	2769,600	3768,400

Cuadro 5.11: Figura de Mérito R_{e2e}: Comparativa número medio de combinaciones exploradas para el peor caso



5.9.1: Tiempo de ejecución del algoritmo exhaustivo



5.9.2: Tiempos de ejecución para los distintos heurísticos y para el algoritmo exhaustivo

Figura 5.9: Figura de Mérito R_{e2e}: Tiempos de ejecución en función del número de combinaciones

Intervalo	Tamaño de bloque fijo							
	2		3		4		5	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	7,896	75,663	11,627	84,969	16,106	91,189	18,973	96,028
$(50, 100]$	10,657	64,956	22,815	84,160	40,894	92,003	55,703	95,211
$(100, 250]$	9,352	44,347	26,477	74,053	56,779	90,527	92,562	94,892
$(250, 500]$	8,448	24,340	27	57,841	62,206	81,894	115,391	93,210
$(500, 1000]$	8	16,019	27	44,212	64	69,561	125	86,100
$(1000, 1500]$	8	9,397	27	36,282	64	60,798	125	78,933
$(1500, 2000]$	8	5,025	27	19,456	64	47,148	125	70,125
$(2000, 2500]$	8	3,811	27	33,830	64	51,908	125	77,950
$(2500, 3500]$	8	2,907	27	29,410	64	47,737	125	81,689
$(3500, 5000]$	8	1,933	27	7,759	64	49,860	125	57,143

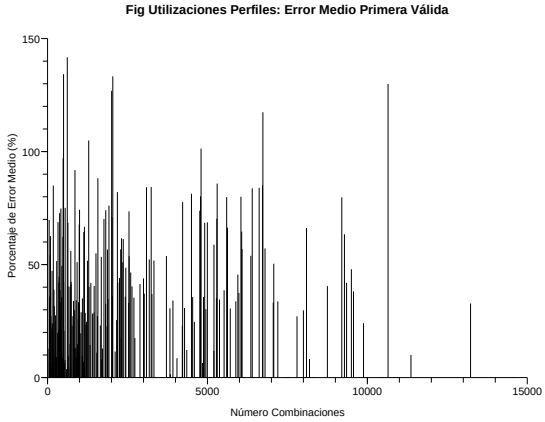
Intervalo	Tamaño variable		Primera Válida	
	$\bar{n}$	$Tr(\%)$	$\bar{n}$	$Tr(\%)$
$(0, 50]$	11,755	85,195	4,912	62,145
$(50, 100]$	24,164	84,302	3,393	36,483
$(100, 250]$	29,530	75,024	1,745	14,247
$(250, 500]$	32,107	59,533	1,224	8,040
$(500, 1000]$	43,758	49,708	1	1,463
$(1000, 1500]$	61,221	47,469	1	0,972
$(1500, 2000]$	74,347	38,414	1	0,787
$(2000, 2500]$	79,286	57,431	1	1,606
$(2500, 3500]$	135,000	51,747	1	0,098
$(3500, 5000]$	150	62,717	1	0,056

Cuadro 5.12: Figura de Mérito Utilización Perfiles: Relación entre las figuras de mérito encontradas con distintos heurísticos y las del exhaustivo

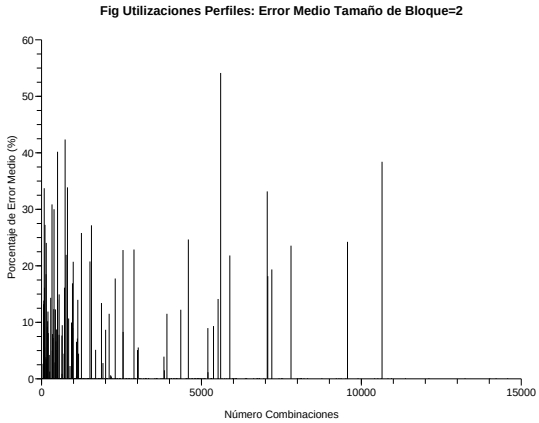
Intervalo Combinaciones	Tamaño de bloque fijo								
	2			3			4		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0,796	15,923	15,923	0	0	0	0	0	0
$(50, 100]$	5,941	67,460	21,785	2,377	30,568	19,614	0	0	0
$(100, 250]$	4,777	40,456	19,842	2,903	27,251	15,678	0	0	0
$(250, 500]$	4,073	49,707	21,085	1,889	49,707	23,749	0,087	7,615	7,615
$(500, 1000]$	5,089	76,185	25,189	1,248	49,291	24,713	0	0	0
$(1000, 1500]$	2,877	70,478	26,303	0,485	14,262	10,338	0,218	13,983	13,983
$(1500, 2000]$	4,535	40,233	27,858	0,563	24,224	24,224	0	0	0
$(2000, 2500]$	1,999	34,733	11,328	0	0,012	0,012	0	0	0
$(2500, 3500]$	4,002	52,962	26,016	0,753	29,371	29,371	0	0	0
$(3500, 5000]$	1,289	24,653	13,109	0,050	3,031	3,031	0	0	0

Intervalo Combinaciones	Tamaño de bloque fijo 5			Tamaño Variable			Primera Válida		
	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	7,355	69,699	29,421
$(50, 100]$	0	0	0	2,377	30,568	19,614	19,080	101,961	29,983
$(100, 250]$	0	0	0	2,348	27,251	18,115	24,298	187,193	45,244
$(250, 500]$	0,087	7,615	7,615	1,691	49,707	24,796	24,598	134,262	47,056
$(500, 1000]$	0	0	0	0,700	49,291	23,111	21,249	141,734	43,826
$(1000, 1500]$	0	0	0	0,218	13,983	13,983	21,717	169,040	43,434
$(1500, 2000]$	0	0	0	0	0	0	36,617	168,133	56,234
$(2000, 2500]$	0	0	0	0	0	0	52,740	169,848	66,413
$(2500, 3500]$	0	0	0	0	0	0	25,584	99,453	49,890
$(3500, 5000]$	0	0	0	0	0	0	23,316	142,304	52,677

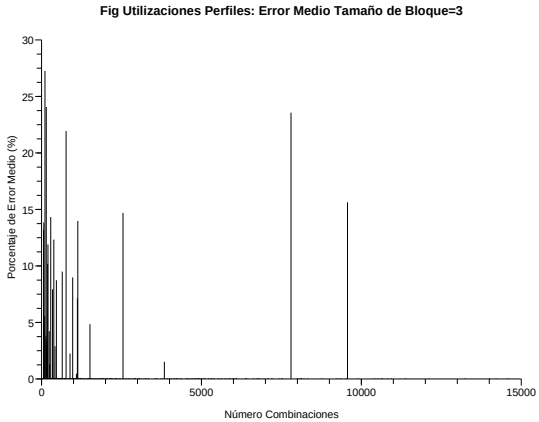
Cuadro 5.13: Figura de Mérito Utilización Perfiles: Error medio, máximo y condicionado a error en porcentaje para el mejor caso.



5.10.1: Con el heurístico: Primera combinación válida



5.10.2: Con el heurístico: Tamaño de bloque fijo de tamaño=2



5.10.3: Con el heurístico: Tamaño de bloque fijo de tamaño=3

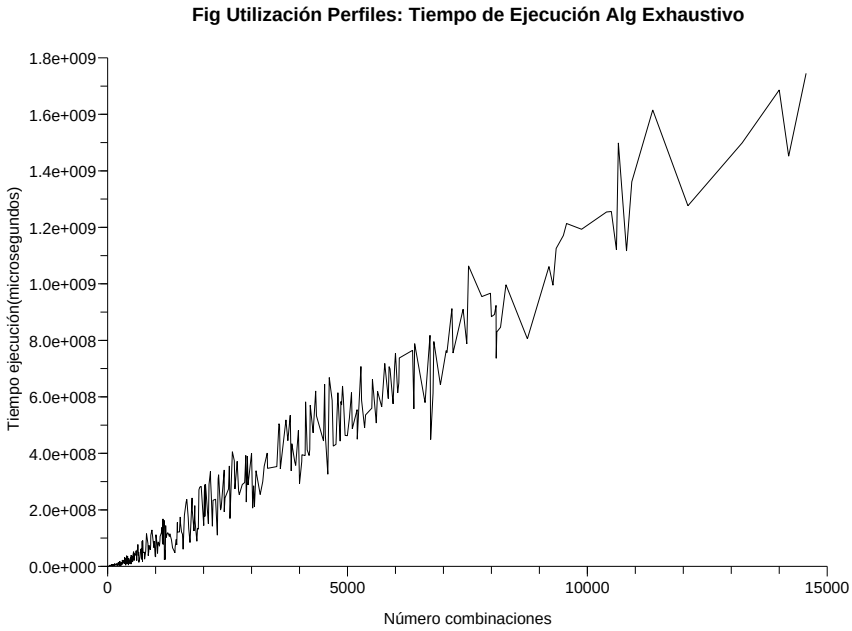
Figura 5.10: Figura de Mérito Utilización Perfiles: Errores Medios con respecto a la Figura de Mérito hallada por el exhaustivo.

Intervalo	Tamaño de bloque fijo								
	2			3			4		
Combinaciones	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0,002	0,343	0,343	0	0	0	0	0	0
$(50, 100]$	0,205	12,079	12,079	0,205	12,079	12,079	0,205	12,079	12,079
$(100, 250]$	1,977	57,376	41,179	1,071	40,658	33,477	0	0	0
$(250, 500]$	3,503	125,762	39,230	1,922	79,828	43,044	0	0	0
$(500, 1000]$	4,691	72,279	33,309	1,096	26,371	15,564	0,725	24,575	12,863
$(1000, 1500]$	1,270	29,218	29,218	0	0	0	0	0	0
$(1500, 2000]$	0,306	4,898	4,898	0	0	0	0	0	0
$(2000, 2500]$	1,606	17,669	17,669	1,606	17,669	17,669	0	0	0
$(2500, 3500]$	1,378	15,159	15,159	0	0	0	0	0	0
Intervalo	Tamaño de bloque fijo			Tamaño Variable			Primera Válida		
	5								
Combinaciones	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$	$\bar{e}$	e_{max}	$\bar{e}_e$
$(0, 50]$	0	0	0	0	0	0	0,723	71,457	39,511
$(50, 100]$	0	0	0	0,205	12,079	12,079	6,506	73,444	42,653
$(100, 250]$	0	0	0	1,071	40,658	33,477	11,195	103,120	45,141
$(250, 500]$	0	0	0	1,877	79,828	42,038	15,155	101,426	47,148
$(500, 1000]$	0	0	0	1,096	26,371	15,564	15,226	82,839	45,044
$(1000, 1500]$	0	0	0	0	0	0	12,397	88,791	47,522
$(1500, 2000]$	0	0	0	0	0	0	15,361	63,110	35,111
$(2000, 2500]$	0	0	0	0	0	0	25,418	124,862	46,600
$(2500, 3500]$	0	0	0	0	0	0	25,578	67,463	46,893

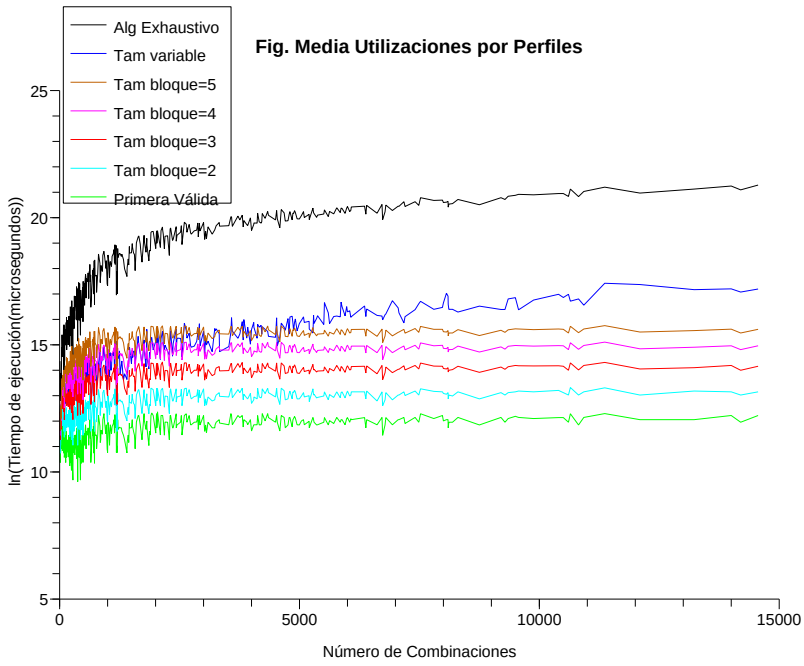
Cuadro 5.14: Figura de Mérito Utilización Perfiles: Error medio, máximo y condicionado a error en porcentaje para $\bar{U} = 0{,}2$.

Intervalo	Alg Exhaustivo	Tamaño Variable	Primera Válida	Tamaño de bloque fijo			
				2	3	4	5
$(0, 50]$	20,483	20,425	18,288	20,287	20,425	20,437	20,483
$(50, 100]$	74,643	74,643	54,993	72,500	74,643	74,643	74,643
$(100, 250]$	171,556	171,556	98,195	164,906	171,556	171,556	171,556
$(250, 500]$	359,840	359,840	129,605	288,220	345,120	359,840	359,840
$(500, 1000]$	750,533	723,733	216,467	426,400	723,733	723,733	750,533
$(1000, 1500]$	1216,714	1188,429	292,000	475,429	1140,429	1188,429	1216,714
$(2000, 2500]$	2200,500	1800,500	463,000	780	1450	2090,250	2200,500
$(2500, 3500]$	2901,400	2300,545	580,500	905,988	1501,671	2290,080	2901,400
$(3500, 5000]$	4150,500	3682,700	805,500	1104,000	1705,700	2509,400	4105,300

Cuadro 5.15: Figura de Mérito Utilización Perfiles: Comparativa número de combinaciones exploradas para el peor caso



5.11.1: Tiempo de ejecución del algoritmo exhaustivo



5.11.2: Tiempos de ejecución para los distintos heurísticos y para el algoritmo exhaustivo

Figura 5.11: Figura de Utilización Media Perfiles: Tiempos de ejecución en función del número de combinaciones

5.7. Conclusiones

En este capítulo, se han propuesto dos algoritmos para la composición de aplicaciones distribuidas de tiempo real. A la complejidad inherente de la composición *software* como elección del camino óptimo en un grafo multivaluado, se une la necesidad de que dicho camino sea planificable, es decir, que todas las acciones involucradas en dicho grafo sean planificables en sus recursos respectivos, que la aplicación compuesta sea planificable y que su instanciación no merme las prestaciones del sistema completo.

La solución ofrecida se basó en la definición de figuras de mérito para la elección de un camino posible, que tuviesen en cuenta las posibles múltiples restricciones impuestas por el usuario o por el sistema y las características propias de cada acción involucrada. La planificación se realizó de manera holística, aproximación que fuerza a una búsqueda en profundidad, de tal forma que había que seleccionar primero un camino, y después decidir si era adecuado realizando la planificación en cada nodo.

A la hora de implementar un algoritmo de búsqueda de un camino inicial para una aplicación basada en servicios, el único medio para asegurarse de que dicho camino es óptimo es realizarla de forma exhaustiva, como es el caso del trabajo de Gu y Nahrstedt [88], o el presentado en el apartado 5.3 del presente capítulo. Sin embargo, esta aproximación es costosa en términos de tiempo de ejecución, pues el número de caminos posibles crece exponencialmente con la profundidad del árbol o grafo, lo que implica, si se tarda el mismo tiempo en evaluar cada camino, un crecimiento exponencial en el tiempo de ejecución.

Como la pretensión de realizar la composición en tiempo de ejecución del sistema, implica la necesidad de que ésta se realice en un tiempo razonable y acotado, un algoritmo exhaustivo no es apropiado. Para permitir dicha característica se propuso un algoritmo mejorado, basado en heurísticos y figuras de mérito relativas para la búsqueda de caminos posibles. Las soluciones ofrecidas por el algoritmo mejorado pueden ser subóptimas. Sin embargo, su utilización asegura tiempos de ejecución sensiblemente más bajos que los del algoritmo exhaustivo cuando crece el número de estados.

Además de las propuestas aquí realizadas, la autora de la presente tesis sólo tiene conocimiento de un algoritmo de composición para aplicaciones de tiempo real basadas en servicios [228]. Dicho algoritmo, para aplicaciones con servicios exclusivamente en secuencia (no permite ramas ni condicionales) parte de la premisa de la existencia de 1 servicio para cada nivel de una aplicación con k niveles de calidad de servicio y no tienen en cuenta ni la planificabilidad de cada servicio en cada recurso, ni la planificabilidad del sistema completo ni exponen su modelo de sistema, además de obviar los mensajes transmitidos por la red, sobre la que no hacen ninguna consideración.

Capítulo 6

Composición Dinámica de Servicios

En este capítulo se presenta la validación de las ideas presentadas en forma de prototipo de la arquitectura para la composición dinámica sobre un paradigma concreto de comunicaciones tiempo real. El paradigma elegido es FTT, en concreto su implementación sobre Switched Ethernet*, FTT-SE. Se exponen las razones de la elección realizada así como las características de FTT-SE. Posteriormente, se propone un arquitectura sobre dicho protocolo y se expone la implementación realizada del prototipo, así como los ajustes realizados en el protocolo de comunicaciones para que le ofrezca soporte.*

6.1. Introducción

La implementación de un prototipo sobre un protocolo concreto de comunicaciones de tiempo real de una de las arquitecturas propuestas fue uno de los objetivos marcados al inicio de esta tesis como medio de validar la premisa de la que ésta partía: la aplicación de conceptos del paradigma de orientación a servicios en sistemas distribuidos de tiempo real como un medio de proporcionarles una mayor flexibilidad. En capítulos anteriores se fijó un modelo de sistema así como de aplicación y de servicios y se estudiaron las distintas entidades que deberían estar presentes en una arquitectura de estas características. A partir de la propuesta presentada en el capítulo 4 para la composición dinámica de aplicaciones, se presenta una arquitectura concreta sobre un protocolo de comunicaciones. La elección del protocolo de comunicaciones vendrá condicionada al modelo de sistema fijado y a los requisitos de la arquitectura. Una vez realizada dicha elección, se presentarán las características del mismo y se definirá una arquitectura. Asimismo, se presentará como prueba de concepto un prototipo de la misma, que confirmará su viabilidad y buen comportamiento temporal.

6.2. Elección del protocolo de comunicaciones

Para implementar el prototipo de un arquitectura que permita la composición dinámica de aplicaciones, en esta tesis nos hemos basado, de todos los protocolos estudiados (véase apartado 2.3), en el paradigma FTT [177]. Este paradigma cubre todas las necesidades estudiadas en los apartados 3.2 y 4.3:

- *Existencia de una base de datos de requisitos del sistema centralizada en un nodo*: como el paradigma FTT sigue el modelo maestro-esclavo, toda la información del sistema está centralizada en el maestro, el cual planificará los mensajes que se transmitirán a través de la red.

- *Soporte para las comunicaciones gobernada por tiempo*: en este paradigma, la comunicación está estructurada en ranuras de duración fija llamadas ciclos elementales (*Elementary Cycles*, EC). Al principio de cada ciclo, el maestro envía un mensaje de activación (*Trigger Message*, TM) que contiene la planificación periódica para dicho ciclo elemental, y señala el momento de la transmisión de los mensajes síncronos.

- *Soporte para las comunicaciones asíncronas*: una vez se transmiten todos los mensajes síncronos de un ciclo, el tiempo restante en cada EC se dedica a la transmisión de los mensajes asíncronos.

- *Flexibilidad en la planificación de las tareas y mensajes*: la planificación de la red se realiza en el maestro en tiempo de ejecución. El hecho de que dicha planificación sea local a un nodo, facilita la utilización de cualquier política de planificación así como la realización de cambios en el conjunto de mensajes que se transmitirán. Esta característica permite tener garantías temporales al realizar en tiempo de ejecución la eliminación o la admisión de nuevos flujos de datos, así como la gestión del ancho de banda en tiempo de ejecución.

El uso de *hardware* de propósito general, hacen que la implementación del paradigma FTT sobre redes *Ethernet* y *Switched Ethernet* sea especialmente atractiva, dado el abaratamiento de costes, su disponibilidad y la facilidad de su despliegue. La implementación del prototipo se realizó sobre FTT-SE [149] (*FTT-Switched Ethernet*) desarrollado como una mejora a *FTT-Ethernet* [179].

6.3. Funcionamiento de FTT-SE

El protocolo *FTT-Ethernet* podría ser desplegado, ofreciendo las mismas garantías temporales que ofrece sobre una red *shared Ethernet*, sobre una red *switched Ethernet*. Sin embargo, las características propias de estas redes, como la ausencia de colisiones y la existencia de caminos de transmisión paralelos, pueden ser aprovechadas para mejorar la eficiencia de este protocolo, surgiendo así *FTT-SE* [149, 150]:

- *Ausencia de colisiones*: permite una simplificación de la implementación del protocolo en los clientes, que una vez recibido el TM, pueden transmitir inmediatamente los mensajes que correspondan en ese período, siendo labor del conmutador la correcta serialización de los mensajes. Así, la propia estructura del TM puede ser simplificada, pues ya no existe la necesidad de indicar el momento de transmisión de cada mensaje dentro del EC.

- *Existencia de caminos paralelos de transmisión*: esta característica puede ser aprovechada si se abandona la arquitectura puramente *broadcast* de FTT-Ethernet, informando al maestro de la naturaleza de los mensajes intercambiados (*unicast*, *broadcast*, *multicast*) y de qué nodos están involucrados. Así, el maestro puede identificar caminos disjuntos y construir planificaciones que exploten este paralelismo. Esta última característica, implica un cambio en el modelo de cooperación empleado. Del modelo productor-consumidor basado en *broadcast* del protocolo original *FTT-Ethernet*, se evoluciona a un esquema publicación-suscripción, donde el maestro podrá almacenar en una estructura de datos los grupos activos de publicadores y suscriptores, incluyendo los identificadores de los flujo respectivos y las direcciones físicas de estos nodos.

6.3.1. Tranmisión de mensajes síncronos

La planificación llevada a cabo por el maestro puede tener en cuenta prioridades individuales de cada mensaje, que no están restringidas ni correladas con los ocho niveles de 802.1D. Como en este paradigma, la resolución temporal es definida por el tamaño del EC, es decir, los mensajes que se transmiten en el mismo ciclo elemental, son, a efectos del protocolo, como si se transmitieran en el mismo instante, estas prioridades definidas son seguidas estrictamente a esa escala, pudiendo producirse el fenómeno de inversión de prioridades dentro de un mismo ciclo elemental.

Como se puede apreciar en la figura 6.1, que muestra la configuración interna del maestro en el protocolo *FTT-Ethernet* original, la base de datos del sistema mantiene la información acerca de los mensajes que se transmiten a través del sistema, tanto síncronos como asíncronos.

Las propiedades de los mensajes síncronos se almacenan en una estructura llamada Tabla de Requisitos Síncronos (*Synchronous Requirements Table*, SRT). Si existen en el sistema N flujos de datos periódicos, SM_i, esta tabla, en FTT-SE, se define como:

$$STR = \left\{ SM_i = \left(C_i, D_i, T_i, \phi_i, S_i, \left\{ R_i^j \right\}_{j=1\ldots m} \right), i = 1 \ldots N \right\} \tag{6.1}$$

donde,

- C_i es el tiempo de transmisión de cada mensaje. El protocolo fragmenta automáticamente los mensajes largos en un secuencia de paquetes que son planifi-

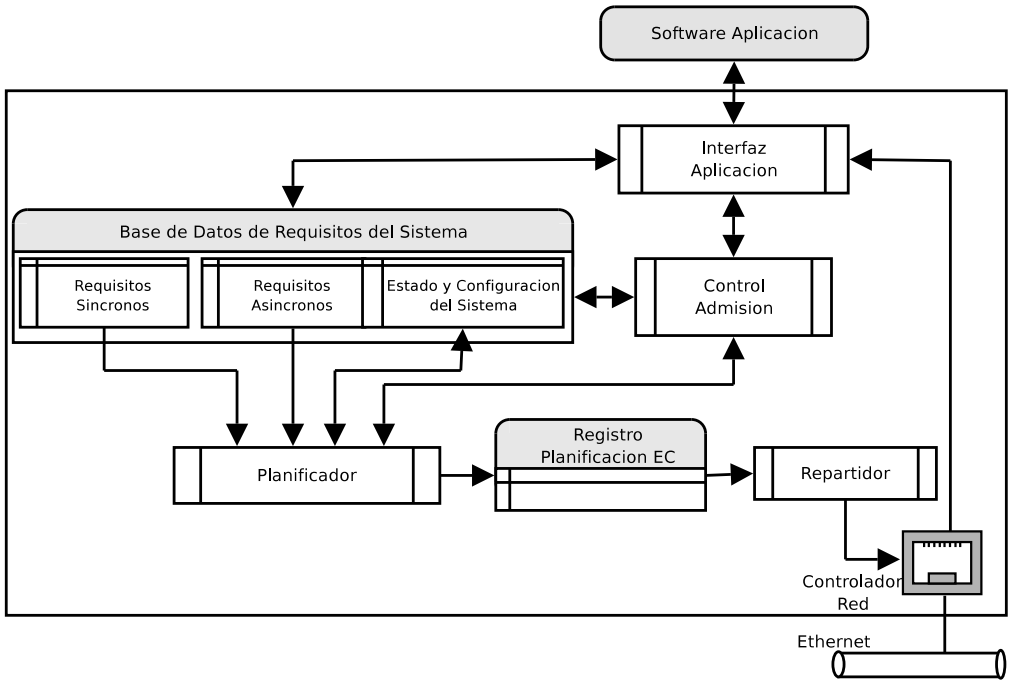


Figura 6.1: Arquitectura interna del maestro en el protocolo FTT-Ethernet original [176]

cados secuencialmente por el maestro. El umbral a partir del cual se realiza la fragmentación se define por flujo de datos.

- D_i es el plazo del flujo de datos.
- T_i es el período del flujo de datos.
- ϕ_i es el desplazamiento temporal respecto a $t = 0$ del momento transmisión del mensaje.
- S_i identifica al nodo emisor del mensaje.
- $\left\{R_i^j\right\}_{j=1...m}$ son los m nodos receptores del mensaje.

C_i,D_i,T_i y ϕ_i se expresarán como múltiplos de la duración del ciclo elemental, T_{EC}.

Dado un conjunto de mensajes almacenados en la SRT, ésta es planificada por el planificador presente en el maestro, que generará una cola simple de mensajes síncronos preparados para ser transmitidos. Esta cola será usada para construir la planificación del siguiente ciclo elemental, que será codificada en el TM y transmitida en *broadcast* a todos los nodos de la red. El TM provocará que los nodos emisores de mensaje en ese EC, envíen inmediatamente los mensajes planificados al conmutador a través de un conjunto de M enlaces de subida l_j^u. Previamente a su transmisión, estos mensajes estaban almacenados en colas locales en cada nodo

emisor. La transmisión es sin apropiación, a menos que se trate de un mensaje fragmentado, en cuyo caso, puede existir apropiación entre paquetes. Una vez llegan al conmutador, los mensajes son dirigidos a los puertos de salida adecuados con una latencia ϵ y encolados para su transmisión en los enlaces de bajada, l_j^d, correspondientes.

6.3.1.1. Construcción de la planificación para cada ciclo elemental

Para la construcción por parte del maestro de la planificación durante cada ciclo elemental se han de tener en cuenta los siguientes aspectos [150]:

- Los mensajes enviados en los enlaces de subida se corresponden con los de los enlaces de bajada, desplazados un intervalo temporal correspondiente con la latencia del conmutador, ϵ.

- La duración de la ventana de transmisión de mensajes asíncronos está limitada a una porción del ciclo elemental, cuyo tamaño máximo se denota como LSW. Al existir la latencia del conmutador, el máximo de ventana de transmisión de cada nodo está limitado a $LSW - \epsilon$.

- Cada nodo necesitará un tiempo para la decodificación del TM, tr, después del cual comenzará sus transmisiones síncronas.

- Las transmisiones en los enlaces de bajada son causales con respecto a las de los enlaces de subida, y siempre se producen al menos un tiempo ϵ después de éstas. Esto quiere decir que si un conjunto de paquetes llega al conmutador en un momento cercano a la finalización de la ventana de transmisión, puede ocurrir que su transmisión en los enlaces de bajada deba extenderse más allá de la finalización de dicha ventana. Por lo tanto, no sólo es necesario comprobar para cada enlace de bajada la cantidad y duración del tráfico de datos, sino también sus instantes de transmisión.

Para construir la planificación para cada ciclo elemental se considerarán en orden los flujos que están preparados para la transmisión:

- *En cada enlace de subida*: se considerará que el inicio de la transmisión de los mensajes ocurren un tr después de la recepción del TM, y si existen varios mensajes en el mismo enlace de subida se transmitirán en secuencia. Se comprobará para cada enlace de subida, la carga hasta el límite marcado por $LSW - \epsilon$:

$$\max_j \left\{ \sum_{SM_i \in l_j^u} C_i \right\} \leq LSW - \epsilon \tag{6.2}$$

- *En cada enlace de bajada*: se tendrá en cuenta el tiempo de finalización de la última transmisión, f_i, y se comprobará si es menor que el límite LSW. Para determinar dicho instante se tendrán que tener en cuenta los instantes de transmisión de los paquetes anteriores en esa cola, y si hay solape, añadir el correspondiente tiempo de transmisión:

$$\max_j \left\{ \max_{SM_i \in l_j^d} f_i \right\} \leq LSW \tag{6.3}$$

En el ejemplo mostrado en la figura 6.2, los flujos $SM3$ y $SM6$ se solapan en el enlace de bajada del Nodo C, lo que implica que el tiempo de transmisión de $SM6$ se ve aumentado, con lo que ya no cumple la condición y debe ser planificado en el siguiente ciclo elemental.

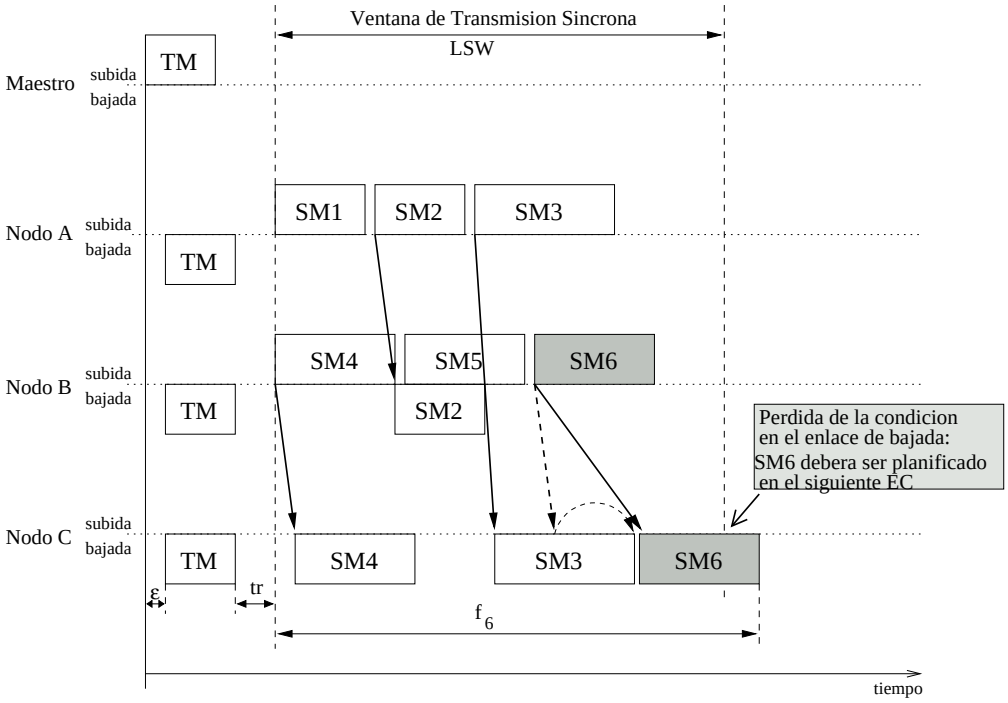


Figura 6.2: Causalidad temporal en los enlaces de bajada [150]

La planificación del ciclo elemental se cerrará en el momento que cualquiera de las dos condiciones anteriores no se cumpla. El flujo causante del incumplimiento se almacenará en la cola de preparados y será planificado en el siguiente ciclo elemental.

6.3.2. Transmisión de mensajes asíncronos

En FTT-SE están implementados dos mecanismos diferentes de gestión de la transmisión de mensajes asíncronos.

El primer mecanismo [149], sigue el esquema definido en *FTT-Ethernet* donde el maestro dentro del propio TM realiza una petición periódica (*polling*) de mensajes asíncronos a los nodos. El período utilizado para cada mensaje asíncrono es igual a su tiempo mínimo entre activaciones (transmisiones), T_{mit}. De esta manera, el tratamiento recibido por los mensajes asíncronos es idéntico al de los mensajes síncronos. Como el *polling* se realiza sin un conocimiento previo de los mensajes asíncronos pendientes de transmitir, se reserva siempre ancho de banda en la ventana de transmisión asíncrona independientemente de que se vaya utilizar o no. Este mecanismo, por lo tanto, implica un uso ineficiente del ancho de banda que va asociado a un alto grado de pesimismo en las garantías temporales ofrecidas a los mensajes asíncronos. La latencia de la señalización, es decir, el tiempo desde que un mensaje asíncrono está preparado para su transmisión en un nodo hasta que llega al maestro, en el peor caso puede llegar a ser $2\,T_{mit}$.

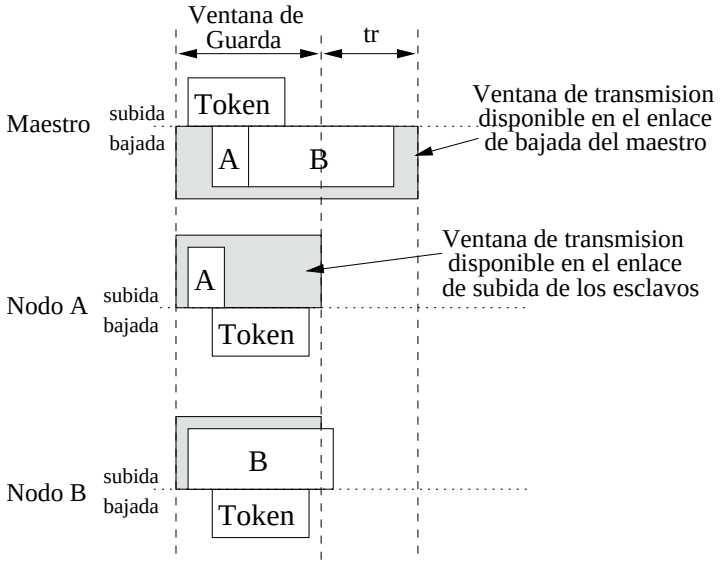


Figura 6.3: Esquema de señalización fuera de banda [150]

Recientemente, se ha propuesto un esquema de señalización fuera de banda para protocolos basados en el modelo maestro-esclavo [150]. En este esquema, los esclavos informan periódicamente al maestro de los mensajes asíncronos que tienen pendiente su transmisión. Este mecanismo se basa en la existencia de enlaces de subida y de bajada en un conmutador *full-duplex*. Así, mientras el maestro transmite el *token* por su enlace de bajada, los nodos esclavos pueden enviarle el estado actual de sus colas de mensajes asíncronos a través de su enlace de subida. Si la transmisión de los mensajes de informe de estado se restringe al periodo temporal que incluye

la ventana de transmisión del *token* y la ventana temporal correpondiente a la decodificación por parte de los esclavos del mismo,tr, este mecanismo no influirá en la operación normal del protocolo (véase figura 6.3).

Este mecanismo se ha implementado sobre FTT-SE, donde el *token* enviado por el maestro es el TM, y los mensajes de sincronización enviados, que incluyen la información asociada con cada nodo, deben tener como máximo el tamaño mínimo asociado con una trama Ethernet que porte carga de datos. Este mecanismo requiere una sincronización de los esclavos con el maestro antes de poder informar de su estado (véase figura 6.4).

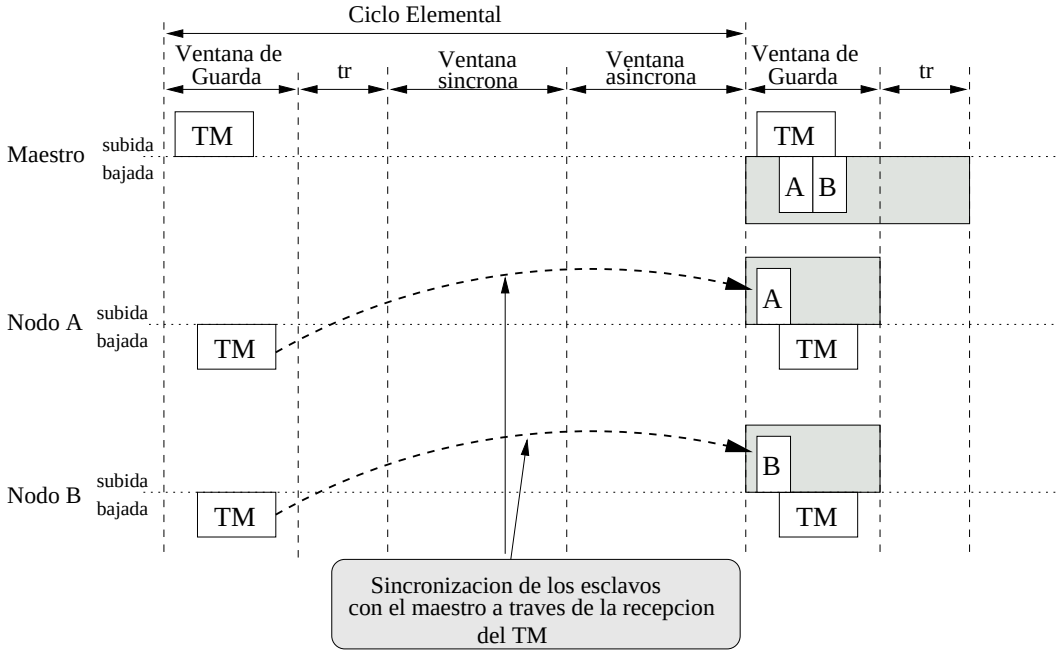


Figura 6.4: Esquema de señalización fuera de banda para FTT-SE [150]

Como la idea es transmitir un mensaje de señalización por nodo cada EC, y el tamaño de la ventana de transmisión de dichos mensajes está acotado, esta opción limita el número máximo de nodos que pueden estar presentes en el sistema:

$$MaxN(s) = \frac{tr(s) + TM_{size}(s)}{SIG_{size}(s)} \tag{6.4}$$

donde,

- $tr(s)$ es el tiempo necesario para la decodificación del mensaje de activación, TM. Este parámetro es ajustable y depende de las prestaciones ofrecidas por los nodos involucrados.

- $TM_{size}(s)$ es el tiempo de transmisión del mensaje de activación.

- $SIG_{size}(s)$ es el tiempo de transmisión de cada mensaje de señalización.

Por ejemplo, en una red *Fast Ethernet* a $100Mbps$, con $TM_{size}(s) = 24\mu s$ y $SIG_{size}(s) = 6{,}72\mu s$ (el tiempo mínimo de transmisión de una trama *Ethernet*). Asumiendo que los nodos del sistema son PCs que usan un núcleo y una pila de protocolos de tiempo real, puede establecerse el tiempo necesario para la decodificación en $tr(s) = 200\mu s$. Bajo estas condiciones, el número máximo de nodos en el sistema es 33.

En el mismo trabajo, se plantean dos aproximaciones para aumentar, si es necesario, el número máximo de nodos presentes en la red:

- *Reducir la tasa a la cual los nodos envían mensajes de señalización*: esta opción influye negativamente en la latencia de señalización, pero elimina la restricción en el número de nodos presentes en el sistema.

- *Extender la ventana en la que el maestro puede recibir mensajes de señalización*: esto puede requerir el ajuste apropiado de la planificación de mensajes síncronos para incluir el tráfico en el enlace de bajada del maestro, pero tiene la ventaja de no afectar a las aplicaciones de los esclavos, pues éstas, habitualmente, no dirigen mensajes al maestro.

La utilización de este segundo mecanismo tiene como ventajas un uso más eficiente del ancho de banda, ofreciendo la posibilidad de que mensajes asíncronos sin requisitos temporales de baja prioridad reclamen el ancho de banda disponible en ausencia de mensajes asíncronos a los que se les deba ofrecer garantías temporales.

Con respecto al análisis del comportamiento del tiempo de respuesta de los mensajes asíncronos, éste puede ser calculado como:

$$AM\_R_t = N_{act} + J_{sch} + C \tag{6.5}$$

donde,

- N_{act}, es el retraso en la activación de los mensajes asíncronos, es decir, el intervalo entre la activación en la cola del esclavo hasta su planificación en el maestro.

- J_{sch} es el término de retraso introducido por el planificador

- C es el tiempo de transmisión del mensaje asíncrono.

El único término de la fórmula que varía en función del mecanismo empleado es N_{act}. Si T_{EC} es la duración del ciclo elemental, T_{min} el período de activación de transmisiones de un mensaje asíncrono y Δ un infinitesimal de tiempo:

- Con *polling* periódico:

- Mejor caso: cuando el mensaje está preparado justo antes de la recepción del *polling* correspondiente a ese nodo:

$$N_{act} = \Delta \tag{6.6}$$

- Peor caso: cuando el mensaje está preparado justo después de la recepción del *polling* periódico.

$$N_{act} = T_{mit} - \Delta \tag{6.7}$$

- Con señalización fuera de banda:

  - Mejor caso: cuando el mensaje está preparado justo antes de la transmisión del mensaje de señalización.

$$N_{act} = T_{EC} + \Delta \tag{6.8}$$

  - Peor caso: cuando el mensaje está preparado justo después de la transmisión del mensaje de señalización. Es decir, al mensaje le lleva un ciclo llegar hasta el maestro y es planificado en el siguiente ciclo elemental.

$$N_{act} = 2\,T_{Ec} - \Delta \tag{6.9}$$

Puede verse que para valores de $T_{min} > 2\,T_{EC}$, el segundo mecanismo tiene mejores tiempos de respuesta en el peor caso.

6.4. Extensiones propuestas al paradigma FTT

La arquitectura propuesta [69] es una extensión al paradigma FTT, que simplifica substancialmente el diseño general de tareas del sistema a través de la reutilización de código y el uso de las características de flexibilidad operacional ofrecidas por el paradigma FTT. Además, es fácilmente portable a las diferentes plataformas *hardware* en las cuales están disponibles implementaciones del protocolo FTT. La figura 6.5 muestra una implementación sobre FTT-SE. El uso sobre otras plataformas de comunicaciones, por ejemplo, CAN, sólo requiere el reemplazo del bloque de FTT-SE por un bloque FTT-CAN, cuya interfaz ofrecida a las aplicaciones es similar.

Básicamente, la arquitectura interna del maestro FTT es complementada con una capa adicional que gestionará aplicaciones y servicios. A nivel de aplicación, la nueva capa mantiene un registro sobre las aplicaciones que están siendo ejecutadas y los servicios que las componen, mientras que a nivel de servicio, gestiona las diferentes versiones de los servicios disponibles, su ubicación, su tiempo de ejecución y respuesta en el peor caso, otros parámetros de calidad de servicio y los mensajes

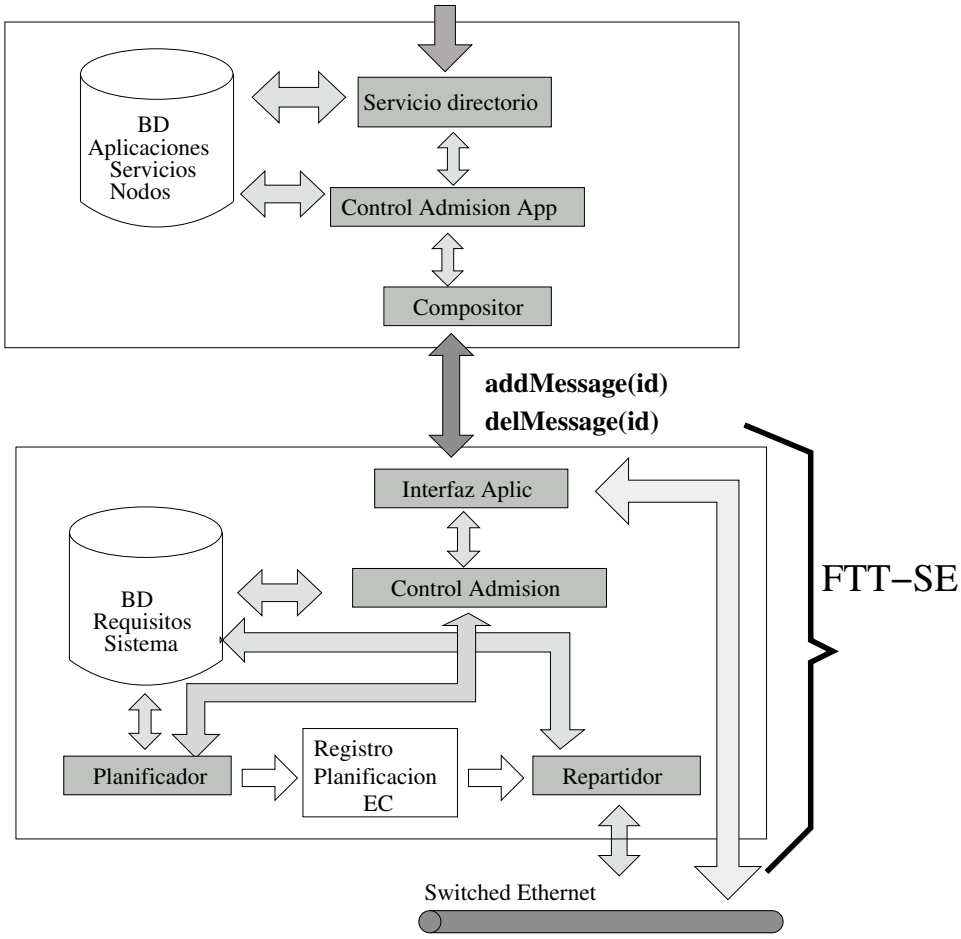


Figura 6.5: Arquitectura propuesta

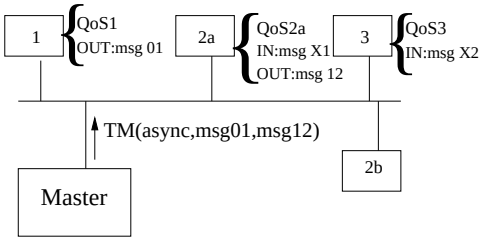
que están involucrados. Toda esta información se almacena en la base de datos de Aplicaciones-Servicios-Nodos.

Cuando un nuevo perfil de un servicio llega al sistema, se lo notifica al maestro a través de un mensaje asíncrono (gestionado por el Servicio de Directorio). El maestro tiene la posibilidad de comprobar qué aplicaciones existentes en el sistema hacen uso del servicio en cuestión y, después de analizar los tiempos de respuesta en el peor caso tanto del perfil de servicio como de las aplicaciones que harían uso de él, si algún parámetro de calidad de servicio es mejorado, por ejemplo el tiempo de respuesta extremo a extremo de una aplicación, y la planificabilidad y prestaciones del sistema completo no se ponen en peligro, el maestro (Compositor) podrá hacer los cambios necesarios a nivel de aplicación para reemplazar la versión vieja por la nueva en la correspondiente transacción. Estos cambios se trasladan a la capa FTT como mensajes o tareas a ser añadidas o reemplazadas en la base de datos de requisitos del sistema.

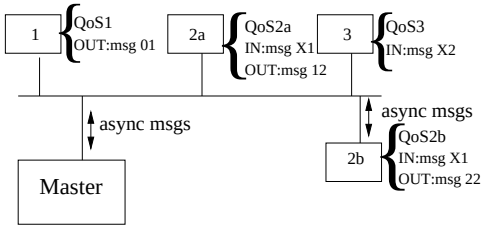
Para conseguir que este mecanismo fuese transparente para el resto de las enti-

Type		TM Flags		Num msgs.	ID	Tiempo	...
tipo TM	Master ID	Reserv.	Num Sec.	Sínc.		Tx	
2 Bytes		2 Bytes		2 Bytes	2 Bytes	1 Byte	...
$[b_{15}, b_{12}]$	$[b_{11}, b_0]$	No def	$[b_7, b_0]$	$[b_{15}, b_0]$	$[b_{15}, b_0]$	$[b_7, b_0]$	...
TM_MESG_ID	$[0, 4096)$	No def	$[0, 256)$	$[0, 65536)$	$[0, 65536)$	$[0, 256)$	...

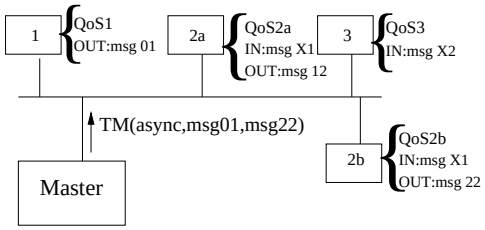
Cuadro 6.1: Estructura original del Mensaje de Activación



6.6.1: Configuración de los nodos



6.6.2: Attach de un nuevo nodo



6.6.3: Reconfiguración

Figura 6.6: Ejemplo de reconfiguración

dades (acciones) involucradas en una aplicación el identificador único de cada mensaje (véase tabla 6.1) se divide en dos partes: un identificador a nivel de servicio y un identificador a nivel de aplicación. El primero identifica diferentes perfiles del mismo servicio, mientras que el segundo identifica los diferentes servicios presentes en el sistema. Los consumidores de un mensaje sólo comprobarán el identificador a nivel de aplicación, es decir, escucharán indistintamente cualquier versión del servicio. Por otra parte, el maestro y los productores deberán comprobar el identificador completo, que permite planificar un perfil concreto entre varios disponibles para el mismo servicio.

La figura 6.6.1 muestra un escenario simplificado que servirá para ilustrar el proceso de reconfiguración. En este escenario, todos los nodos implementan un único servicio y la aplicación se expande sobre tres nodos. Los identificadores de mensaje tienen dos dígitos, el primero de ellos correspondiente al identificador a nivel de servicio y el segundo al identificador a nivel de aplicación. En tiempo de ejecución,

el mensaje de activación, TM, activa la transmisión de los mensajes síncronos 01 (perfil 0, servicio 1) y 12 (perfil 1, servicio 2). Así, los nodos 1 y $2a$ en los que están ubicados los perfiles planificados transmiten sus mensajes sobre la red en *broadcast*. Los mensajes son recibidos por los nodos $2a$ y 3 respectivamente. La X en el identificador a nivel de servicio significa *indiferente*. Cuando un nuevo nodo llega a la red, este le envía al maestro un mensaje de señalización indicando que tiene mensajes asíncronos que transmitir y, cuando el maestro planifica dicho mensaje, envía toda su información (servicios y perfiles de servicio que implementa, así como sus características) al maestro. El maestro asigna un identificador a cada nuevo perfil de servicio ($2b$ en este caso producirá el mensaje 22) y decide reemplazar en la aplicación el perfil que está ubicado en $2a$ por esta nueva versión. Así, empieza a planificar el mensaje 22 (perfil 2, servicio 2) en vez del 12, figura 6.6.3. El siguiente servicio en la aplicación, es decir, el 3, continúa recibiendo los mensajes que tengan el identificador a nivel de aplicación 2, ya sea el 12 como el 22, y no será consciente del cambio realizado.

6.5. Prototipo del Sistema

Para realizar una validación práctica de la arquitectura propuesta, se implementó un prototipo sobre una implementación de FTT-SE, desarrollada por el *Laboratório de Sistemas Electrónicos da Universidade de Aveiro*.

Dicho prototipo se comporta como un compositor que, en tiempo de ejecución del sistema, permite realizar cambios en la configuración de los perfiles utilizados por una aplicación.

La plataforma *software* sobre la que se ejecuta la implementación es RT-Linux y el lenguaje empleado C, con bibliotecas de código propias de la plataforma. En el momento de la implementación, la comunicación asíncrona usando señalización fuera de banda posteriormente desarrollada por dicho grupo, véase apartado 6.3.2, no estaba completamente desarrollada, y las interfaces ofrecidas a la aplicación se restringían a añadir o borrar un nuevo mensaje síncrono o asíncrono (estos últimos seguían el mecanismo para tráfico asíncrono desarrollado para *FTT-Ethernet* [179]).

6.5.1. Cambios introducidos en los nodos esclavos

En primer lugar, para permitir que un nodo consumidor de un mensaje de un servicio recibiese todos los mensajes con el mismo identificador a nivel de aplicación independientemente del perfil concreto que lo generase, hubo que modificar los módulos de recepción y transmisión de los nodos esclavos.

La figura 6.7 muestra el proceso seguido por un nodo esclavo a la hora de recibir una trama, ya sea un mensaje de activación (TM), ya sea un mensaje de datos. Aunque en esta figura, por simplicidad, se presenta como un proceso secuencial, cada una de las tareas se lleva a cabo por una entidad distinta:

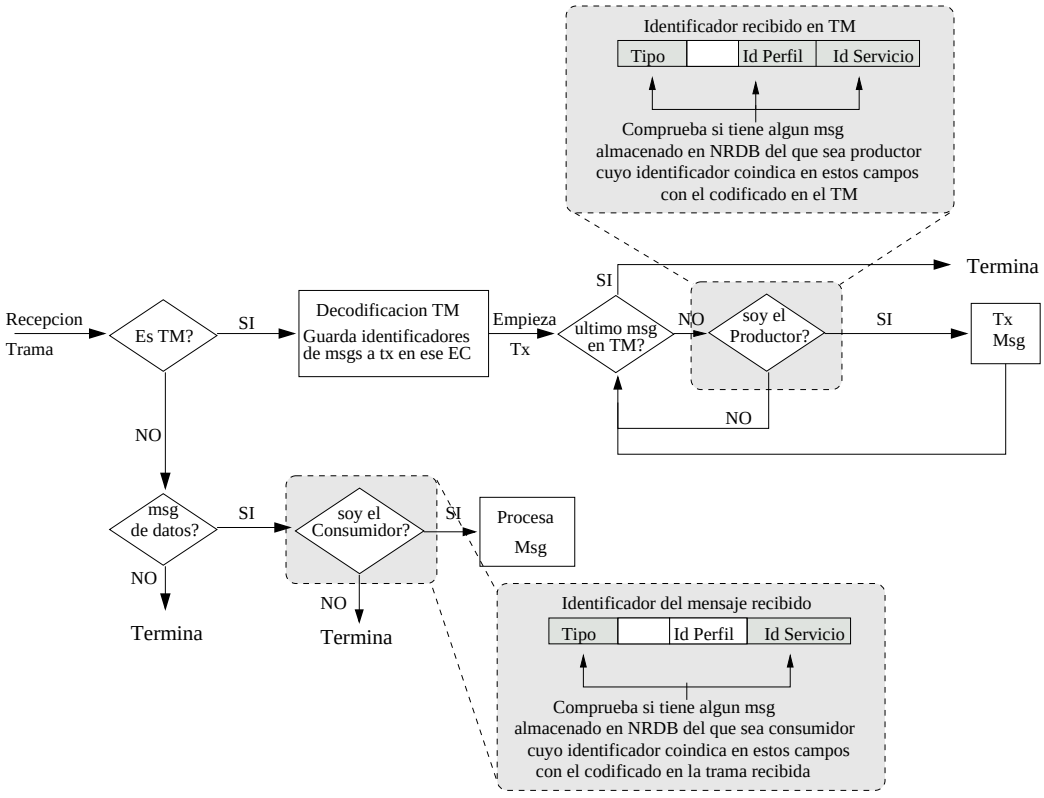


Figura 6.7: Cambios realizados en el módulo de recepción y transmisión de los nodos esclavos

- La Interfaz de Aplicación del nodo esclavo será la encargada de realizar el preprocesado, es decir, determinar el tipo de mensaje que se ha recibido:
  - Si es un mensaje de activación, procesarlo y despertar a la entidad encargada de enviar las tramas (*Dispatcher* o Repartidor).
  - Si es un mensaje de datos, determinar si el esclavo es consumidor de dichos datos, preprocesarlos y despertar a la función *callback*, si la hubiere, asociada a la recepción de ese mensaje.
- La entidad *Dispatcher* será quien enviará cada ciclo elemental las tramas producidas por este nodo, en función de lo indicado en el TM.
- Las distintas funciones *callback*, asociadas cada una de ellas con un mensaje consumido, serán quienes procesen los mensajes.

En la figura 6.7 se aprecia dónde se tuvieron que introducir las modificaciones, resaltadas en gris. Básicamente, consistieron en la aplicación de una máscara a los mensajes entrantes, de tal forma que, en el caso de ser consumidor de un servicio, se consumiesen todos los mensajes procedentes de dicho servicio, independientemente

del perfil que los generase. La máscara a aplicar es un parámetro que recibe el nodo esclavo en el momento de su inicialización.

Esta solución, permite que, únicamente modificando la máscara a aplicar, los nodos esclavos recuperen el funcionamiento anteriormente definido en el protocolo, es decir, que tengan en cuenta todo el identificador a la hora de consumir un mensaje. Por otra parte, puede modificarse de manera sencilla el número máximo de perfiles por servicio soportados por el sistema, únicamente modificando dicha máscara.

6.5.2. Extensiones realizadas en el maestro

En el nodo maestro, se tomó la decisión de implementar el prototipo como una extensión al protocolo FTT-SE, haciendo uso de las primitivas de manipulación de la tabla de mensajes que éste ofrece: `addmesg` y `delmesg`.

Para dar soporte a aplicaciones compuestas, se definió e implementó una base de datos en C. Esta base de datos guarda información sobre los nodos físicos, los servicios presentes en la red y las implementaciones de éstos, así como sobre las aplicaciones del sistema, estructurada de la siguiente forma:

- **Tabla de nodos físicos**: de tamaño `MAX_NODES`. Para cada nodo presente en el sistema, guarda:
  - Un identificador único.
  - Su dirección MAC.
  - El número de perfiles totales presentes en el nodo.
  - Un puntero a su fila correspondiente en la matriz de relaciones entre nodos físicos y servicios
- **Tabla de servicios presentes en el sistema**: de tamaño `MAX_SERVICES`. Para cada servicio implementado en el sistema, guarda:
  - Un identificador de servicio.
  - El nombre de la funcionalidad que implementa.
  - El tamaño del mensaje que generarán sus perfiles.
  - El número de implementaciones presentes en el sistema.
  - Un puntero a su columna correspondiente en la matriz de relaciones entre nodos físicos y servicios.
  - El conjunto de identificadores de aplicaciones que usan dicho servicio, así como su número.
- **Matriz de relaciones entre nodos físicos y servicios**: de tamaño `MAX_NODES x MAX_SERVICES`. Cada celda guarda la siguiente información:

- Número de perfiles del servicio presentes en el nodo.
- Número de aplicaciones que usan ese servicio en dicho nodo.
- Puntero a la lista enlazada de perfiles que implementan dicho servicio en ese nodo.

Nótese que el número de perfiles de un servicio presentes en un nodo, no está acotada.

- **Tabla de perfiles**: de tamaño `MAX_PROFILES`. En esta tabla se guardan todos los perfiles presentes en el sistema, de todos los nodos. Para cada perfil se guarda la siguiente información:
  - Características de calidad de servicio. En esta implementación, sólo se tuvo en cuenta el tiempo en el peor caso de respuesta del perfil.
  - Aplicaciones que lo usan.
  - Puntero al siguiente perfil en la lista enlazada de perfiles.
- **Tabla de aplicaciones compuestas del sistema**: de tamaño `MAX_APP_SYSTEM`. Para cada aplicación en el sistema guarda:
  - Identificador único de aplicación.
  - Nombre de la aplicación.
  - Plazo deseado de la aplicación.
  - Calidad de servicio actual ofrecida a dicha aplicación.
  - Puntero al primer vértice del grafo de la aplicación en ejecución.

  El grafo de la aplicación se estructura de forma secuencial por vértices unidos a través de arcos. Cada arco se corresponde con un mensaje que se intercambia y los vértices a los servicios de dicha aplicación. servicios. Las estructuras de datos correspondientes a los arcos y a los vértices se almacenan en sendas tablas de arcos y vértices del sistema. Para cada vértice se guarda la siguiente información:
  - Tipo: simple o paralelo.
  - Puntero a un perfil determinado de la matriz de servicios y nodos del sistema.
  - Puntero al arco correspondiente al mensaje generado.
  - Puntero al siguiente vértice del grafo.

  Para cada arco del grafo:
  - Identificador de la aplicación.

- Identificador del mensaje.
- Tamaño del mensaje
- Tamaño máximo del mensaje.
- Período del mensaje.
- Plazo del mensaje.
- Desplazamiento temporal del mensaje, expresado como múltiplo de la duración del ciclo elemental.
- Identificadores de los vértices de la aplicación productores y consumidores.

Como el tiempo de creación y destrucción de memoria no está acotado, deben evitarse operaciones de gestión de memoria durante la ejecución de la aplicación, así que toda la memoria que precisará la aplicación debe ser reservada antes de la ejecución del sistema, por lo que todas las tablas involucradas tienen un tamaño máximo, que puede ser modificado antes de la ejecución del sistema, no existiendo limitación sobre éste.

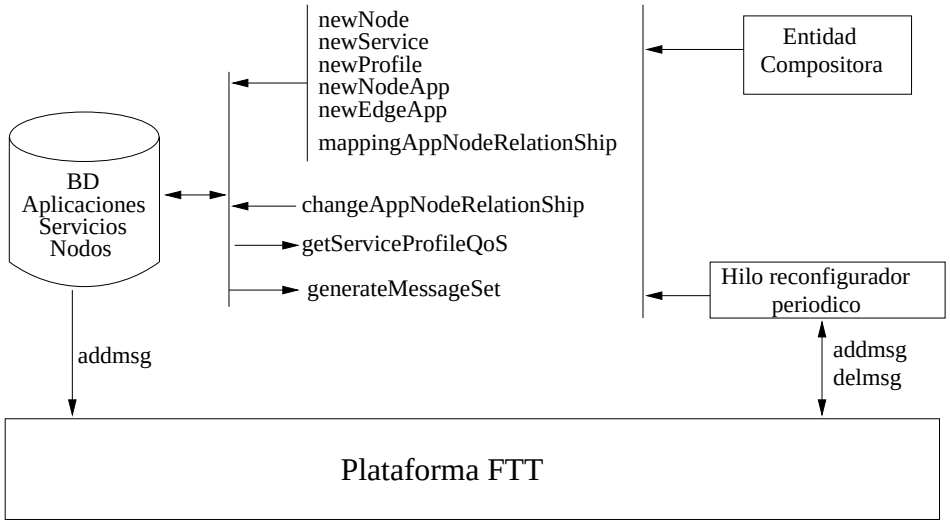


Figura 6.8: Esquema del prototipo implementado en el maestro

Las tablas se manejan a través de interfaces (véase figura 6.8, donde se han detallan las primitivas más relevantes) que permiten almacenar información sobre los nodos, servicios y perfiles del sistema. Estas interfaces también permiten almacenar información sobre la estructura de las aplicaciones en ejecución del sistema, ofreciendo la posibilidad de generar y añadir el conjunto de mensajes definidos por las aplicaciones a través de la primitiva `generateNewMessageSet`, la cual hace uso de `addmesg` para modificar la base de datos de mensajes a planificar en el

sistema. A través de la primitiva `mappingAppNodeRelationShip` se ofrece, además, la posibilidad de, dado un vértice de la aplicación, cambiar el perfil que lo implementa.

6.5.3. Entidades implementadas en el maestro

En el momento de la implementación del prototipo, como todavía no se disponía de una comunicación asíncrona fuera de banda con los nodos que permitiese la implementación de un mecanismo *plug&play*, o que éstos comunicasen al maestro nuevos perfiles disponibles, se permite que el sistema se inicialice a partir de la especificación completa del mismo: nodos, servicios, perfiles y aplicaciones, realizada por la entidad compositora, que asume así, en el prototipo, las funciones de la entidad de publicación.

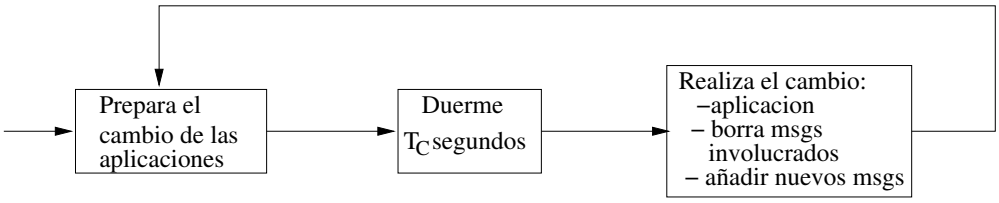


Figura 6.9: Esquema del hilo periódico que realiza el cambio en las aplicaciones compuestas

Para llevar a cabo la prueba de concepto, se implementó un hilo periódico según el esquema mostrado en la figura 6.9. Este hilo, de período T_C (especificado en su instanciación), realizaba los cambios necesarios en las aplicaciones existentes en el sistema, centrándose en los perfiles utilizados en la aplicación compuesta. Los perfiles podían diferenciarse en ubicación o calidad de servicio ofrecida, limitanto el concepto de calidad de servicio al tiempo de respuesta en el peor caso del perfil. Si se produce un cambio en el tiempo de respuesta, éste afecta a los desplazamientos temporales de todos los mensajes subsiguientes. Así, el nodo compositor debía modificar los desplazamientos temporales de todos los mensajes siguientes al modificado.

6.5.4. Herramientas de medida implementadas

Para realizar las medidas, se optó por la utilización de dos herramientas: una no intrusiva, consistente en la utilización de una solución de *software* libre de análisis de tráfico en una red; y una herramienta *software* implementada tanto en los nodos esclavos como en el maestro para realizar trazas de la ejecución del sistema. En la figura 6.10 se muestran las interfaces de la herramienta implementada. Para evitar excesivas interferencias, la herramienta almacenaba en una estructura de datos la traza del sistema y, una vez terminada su ejecución, volcaba dicha traza a un fichero,

mediante el uso de FIFOs de tiempo real para comunicar el espacio del núcleo de tiempo real con el espacio del usuario. Como la gestión de memoria es no predecible, las cadenas asociadas a los tiempos medidos tenían que ser especificadas *a priori*. La evaluación de la interferencia de la herramienta *software* implementada se presentará en el apartado 6.6, dedicado a las medidas experimentales realizadas.

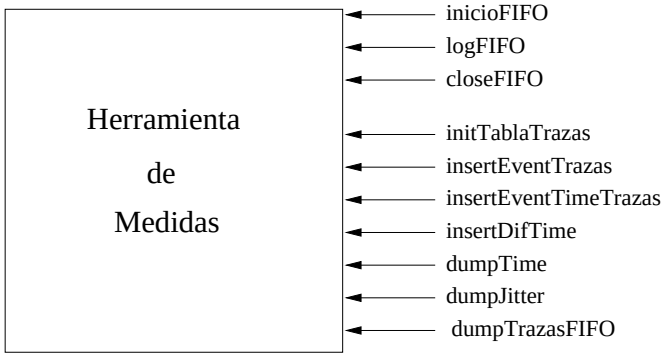


Figura 6.10: Interfaces ofrecidas por la herramienta de medida

Dada la imposibilidad de reservar memoria dinámicamente, por su falta de predictibilidad temporal, la herramienta ofrece, además, la posibilidad de ir guardando en tiempo de ejecución trazas que tengan parámetros variables en forma de cadenas en una FIFO perteneciente al espacio del núcleo de tiempo real, que serán leídas por un *script bash* que se ejecuta en el espacio de usuario. Las operaciones de escritura en la FIFO no representan apenas sobrecarga, pues se empleó una función del núcleo de tiempo real, `rtl_printf`, para implementarlas. Sin embargo, como el hilo encargado de leer la FIFO en el espacio de usuario, no tiene restricciones temporales, si el núcleo está muy cargado, pueden perderse trazas debido a que la FIFO, que está implementada como un búfer, se llene. En el prototipo, como había memoria suficiente en todos los nodos, la FIFO se sobredimensionó para evitar este problema. En otros entornos, donde la memoria sea un recurso escaso y la carga estimada en el nodo a trazar sea elevada, esta funcionalidad de la herramienta de medida podría no ser adecuada.

6.5.5. Consideraciones prácticas

En este apartado se realizan algunas consideraciones de tipo práctico sobre el prototipo implementado.

6.5.5.1. Direccionamiento

En el prototipo implementado se empleó direccionamiento *broadcast* para todos los mensajes intercambiados por los perfiles de servicio de una aplicación.

Esta decisión se tomó teniendo en cuenta ciertas limitaciones de la implementación en ese momento de FTT-SE, donde existía una carencia de mecanismos asíncronos de comunicación fuera de banda entre el maestro y los nodos, que permitiese indicarles a estos últimos el conjunto de consumidores de un mensaje determinado, necesario para poner implementar la comunicación entre productores y consumidores a través de un mecanismo *multicast*.

Esta decisión de diseño implica no utilizar toda la potencialidad del protocolo FTT-SE. Sin embargo, se aisla a los productores de los consumidores de un mensaje, consiguiendo transparencia de ubicación con respecto a las localizaciones de ambos grupos. Los productores sólo deben saber el momento en el que deben enviar un mensaje y los consumidores sólo deben escuchar la red a la espera de un mensaje que proceda de un servicio, sea cual fuere la ubicación del perfil que lo genere.

6.5.5.2. Gasto de recursos por parte de perfiles instanciados pero que no forman parte de una aplicación compuesta

En la implementación de FTT-SE sobre la que se trabajó, las tareas encargadas de procesar un mensaje son despertadas por el módulo de recepción (véase figura 6.7), mientras que la transmisión de los mensajes es gobernada por tiempo. Cada mensaje enviado por un nodo esclavo tiene asociado una zona de datos distinta, donde se almacenará el mensaje a transmitir, que será el que lea la tarea encargada de transmitir dicho mensaje.

Así, en el prototipo se implementaron:

- Las tareas productoras como tareas periódicas que, una vez ejecutadas, almacenan los datos en su zona de memoria de salida.

- Las tareas consumidoras como funciones *callback*, que procesan los datos que reciben en su zona de memoria de entrada, y

- Las tareas productoras-consumidoras como funciones *callback*, que procesan los datos que reciben en su zona de memoria de entrada y, una vez realizada su tarea, almacenan el resultado obtenido en su zona de memoria de salida.

Dichas zonas de memoria no se comportan como búferes, sino que son sobreescritas en cada operación de escritura.

Nótese que, dada la implementación del sistema, si una tarea productora-consumidora recibe datos escribirá siempre en su zona de memoria de salida el mensaje resultado de su ejecución, siendo labor del maestro, decidir si dicho mensaje se transmitirá o no.

Es decir, en el prototipo, los perfiles consumidores de un determinado mensaje, consumirán siempre recursos en el nodo en el que están instanciados, aunque no se transmita el mensaje que generan. Esto podría modificarse realizando un cambio en

la implementación de los nodos consumidores, de tal forma que la recepción del TM no sólo indicase la transmisión de mensajes, sino también activaciones de tareas.

Tal y como está implementado el prototipo, si existen varios perfiles de un servicio consumidores del mismo mensaje y sólo uno que deba procesarlo, todos los demás gastan recursos en los nodos en los que están instanciados. Si se fuerza a que el TM indique, además, activaciones de tareas, podría elegirse el perfil exacto que se está utilizando y los demás no gastarían recursos innecesariamente.

Nótese, sin embargo, que esta característica puede ser usada provechosamente en la implementación de tolerancia a fallos mediante la definición de grupos de perfiles que realizan una misma funcionalidad y si uno falla, en sólo dos ciclos elementales, podría el maestro recomponer la aplicación.

6.6. Validación experimental

Para realizar una validación práctica de la arquitectura propuesta. ésta se implementó sobre FTT-SE. Definimos una configuración experimental, véase figura 6.11, usando ordenadores cuyo sistema operativo era RT-Linux. La duración del ciclo elemental en estos experimentos fue inicializado a 1 milisegundo. Para observar el tráfico en la red, se forzó a que todos los mensajes se transmitiesen en *broadcast* y se dispuso en otro nodo un programa que escuchaba y capturaba tráfico de la red. El sistema ejecuta una aplicación compuesta por tres servicios en secuencia, tal y como se muestra en la figura 6.12.

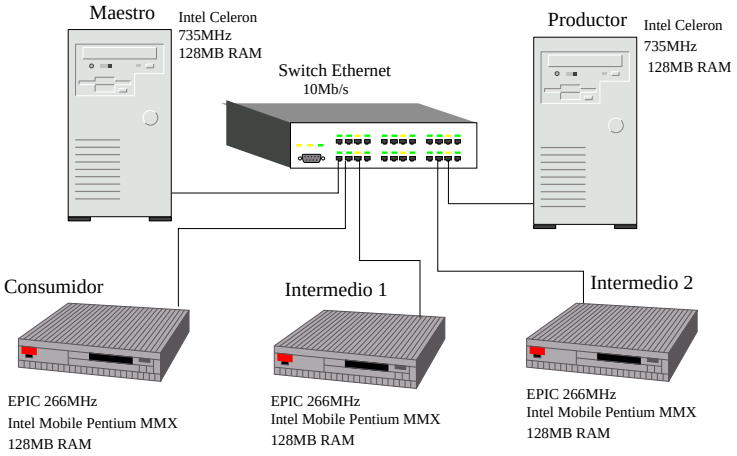


Figura 6.11: Configuración de los nodos en los experimentos realizados

6.6.1. Interferencia introducida por las herramientas de medida

Para realizar las medidas del modelo implementado, se han utilizado dos medios: por un lado una herramienta software y por otro la observación de los paquetes pre-

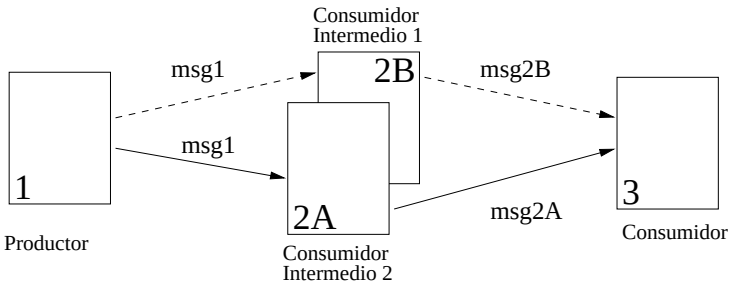


Figura 6.12: Configuración de la aplicación en los experimentos realizados

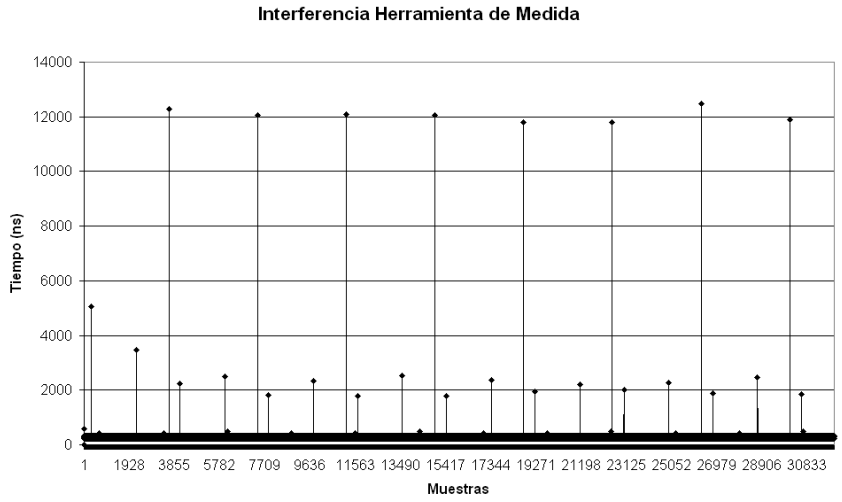


Figura 6.13: Interferencia introducida por la herramienta de medida

sentes en la red mediante la utilización de una solución *software* libre de análisis de protocolos de red. Al transmitirse los paquetes en *broadcast*, esta solución *software* es no intrusiva. Sin embargo, la herramienta *software* implementada podría interferir en los datos medidos.

En la figura 6.13 se muestra la interferencia de la herramienta de medida. Para medirla se realizaron 32000 invocaciones sucesivas al método `insertEventTrazas`, que accede al reloj del sistema, y almacena su valor actual y una cadena relativa al evento indicado en la estructura de datos interna de la herramienta de medidas. La figura representa la diferencia entre valores temporales consecutivos. Las medidas se realizaron sin ningún otro hilo del sistema que interfiriese en la ejecución y con la red inactiva.

Se puede ver que, en media, la interferencia introducida es muy baja, en concreto para la ejecución mostrada en la figura de $\mu_{inter} = 264{,}19ns$, con una desviación típica de $\sigma_{inter} = 197{,}5ns$. Sin embargo, se observan picos en el valor de la interferencia, siendo su máximo en la ejecución que se muestra de $12471ns$. En otras

ejecuciones realizadas, se observan aproximadamente los mismos valores tanto para la media como para la desviación típica, presentándose los mismos picos de tipo periódico. Los picos máximos, con valores entornos a los $12000ns$ se presentan aproximadamente cada $1ms$. También se observa la presencia de picos menores de valores de interferencia entorno a los $2000ns$, que se presentan cada $0{,}5ms$.

Este comportamiento se repite en todos los experimentos realizados, no sólo en la evaluación de la implementación de este prototipo, sino también en medidas realizadas directamente a la implementación de FTT-SE sobre RT-Linux con otras herramientas de medida, y en medidas realizadas en otros trabajos del grupo de investigación haciendo uso de Java de tiempo real sobre RT-Linux [25].

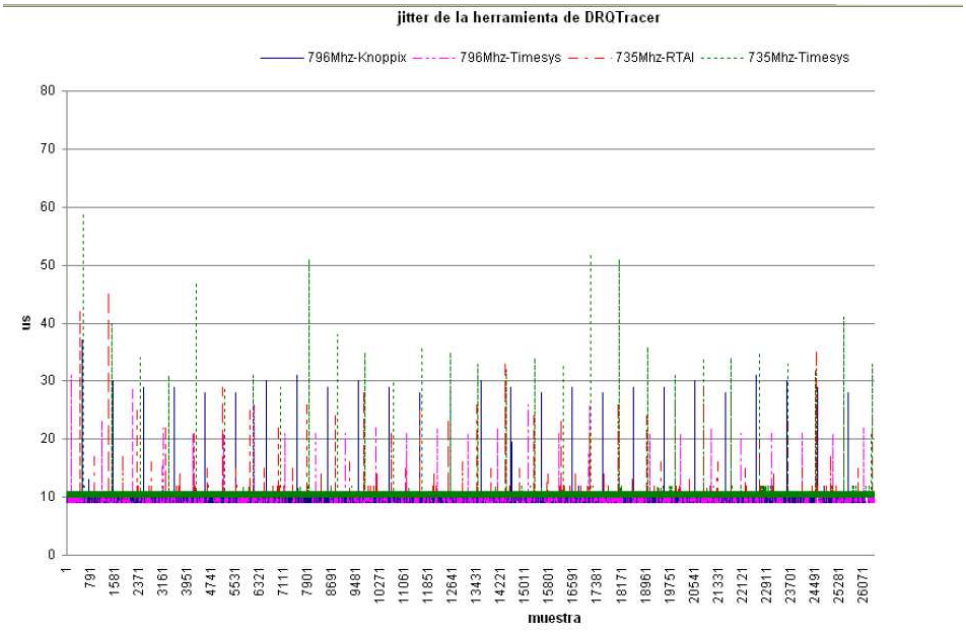


Figura 6.14: Interferencia introducida por la herramienta de medida DRQTracer (figura tomada de [25])

En la figura 6.14 se muestra una gráfica de las medidas de la interferencia de una herramienta de medida implementada en RT-Java, DRQTracer [25], en distintas plataformas. Si se comparan las medidas obtenidas para *Timesys* de esa herramienta, la media de la interferencia está entorno a los $10\mu s$ y el peor caso está entorno a $57\mu s$. El peor caso observado en los experimentos realizados a la herramienta de la que trata esta sección está entorno a $25\mu s$. Teniendo en cuenta que el anterior trabajo se ejecutaba en espacio de usuario, mientras que estas pruebas en concreto, se ejecutaron como módulos que se cargan en el espacio del núcleo de tiempo real, con la máquina desnuda, es decir, en modo consola, eliminando todos los servidores innecesarios, antes de la ejecución de la plataforma FTT-SE, se puede llegar a la conclusión de que alguna tarea interna de alta prioridad del sistema operativo de tiempo real está interfiriendo en la ejecución de los programas, sin que se haya conseguido

determinar qué tarea es exactamente. No se descarta que pueda ser en concreto una interrupción *hardware*, pues éstas tienen prioridad sobre los hilos de tiempo real, y sobre este tipo de interrupciones, RT-Linux asegura que una interrupción *hardware* sólo se apropiará del sistema una vez cada ciclo de actividad de los hilos de tiempo real [76], lo que se ajusta al comportamiento periódico observado y a que, para diferentes entornos, esta interferencia parezca ajustarse a sus ciclos.

6.6.2. Coste cambio de la configuración de las aplicaciones

Por otra parte se midió el coste del cambio en las aplicaciones compuestas en el hilo periódico encargado de realizarlo, véase figura 6.15.

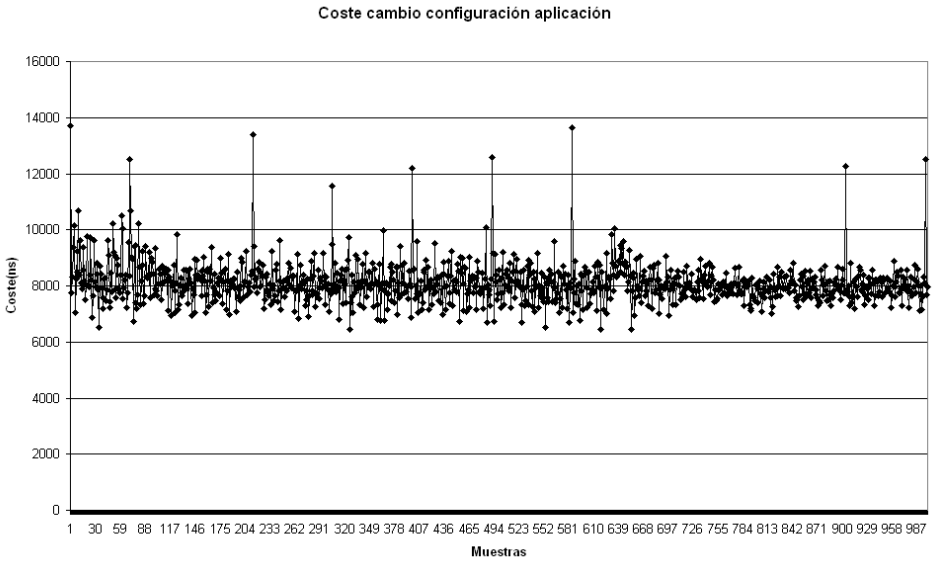


Figura 6.15: Coste cambio de la configuración de las aplicaciones

En concreto se midió el coste de la eliminación periódica, con período $T_C = 0{,}5s$, de un mensaje en la base de datos de requisitos del sistema, BDRS, a través de la interfaz ofrecida por la infraestructura básica de FTT y la introducción de uno nuevo en la misma modificando, previamente, la configuración de la aplicación en la base de datos definida en el prototipo.

En media, al maestro le cuesta realizar un cambio en la aplicación $\mu_{coste} = 8\mu s$ con una desviación típica de $\sigma_{coste} = 750ns$. Se vuelven a observar picos de aproximadamente $12\mu s$ en la ejecución, producto de la interferencia de tareas de alta prioridad del sistema operativo subyacente en la ejecución del maestro.

6.6.3. Dos implementaciones similares de una tarea productora-consumidora

Este experimento verifica que el cambio entre dos perfiles del mismo servicio correspondiente a una tarea productor-consumidora se produce de manera transparente tanto a la tarea productora precedente en la aplicación como a la tarea consumidora inmediatamente posterior.

Los perfiles que se intercambiarán residen en los nodos *Consumidor Intermedio* 1 y *Consumidor Intermedio* 2 y exhiben parámetros temporales similares, en este caso: $\phi = 200\,T_{EC}$, $T = 1000\,T_{EC}$, $D = T$. El maestro realiza el cambio entre perfiles equivalentes cada 10 segundos.

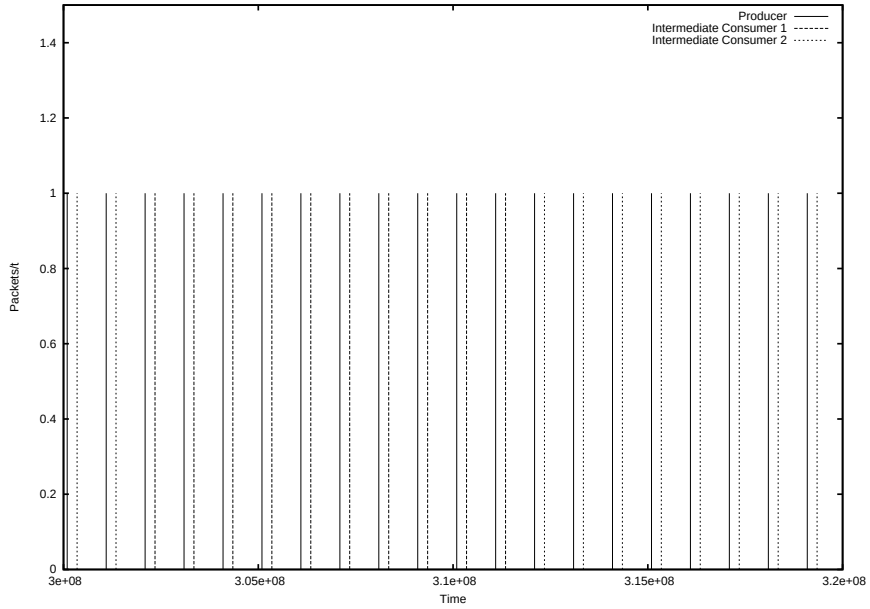


Figura 6.16: Experimento 1: Transmisión de paquetes en la red observada por el programa capturador de tráfico

Por su parte, el nodo donde se ubica la tarea consumidora recibe el mensaje producido por los consumidores intermedios y registra su instante de recepción, mientras el programa capturador de tráfico registra los mensajes intercambiados entre los diferentes nodos (véase figura 6.16). Puede observarse que el intercambiar ambos perfiles no causa o significa ningún retraso adicional en la recepción del mensaje.

Por otra parte, si se computa el tiempo de respuesta de los productores intermedios, es decir, el tiempo desde que se transmite el paquete del productor hasta que los productores intermedios envían el suyo, véase figura 6.17, se mantiene, como era de esperar alrededor de 253 ms con un retraso o *jitter* de ± 30 ns y picos aislados de 100 ns.

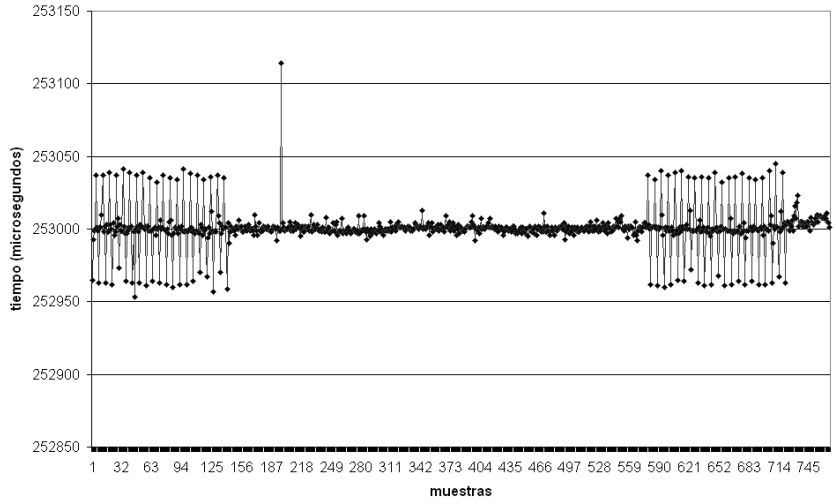


Figura 6.17: Experimento 1: Tiempo de respuesta de los nodos productores medido por el programa capturador de tráfico

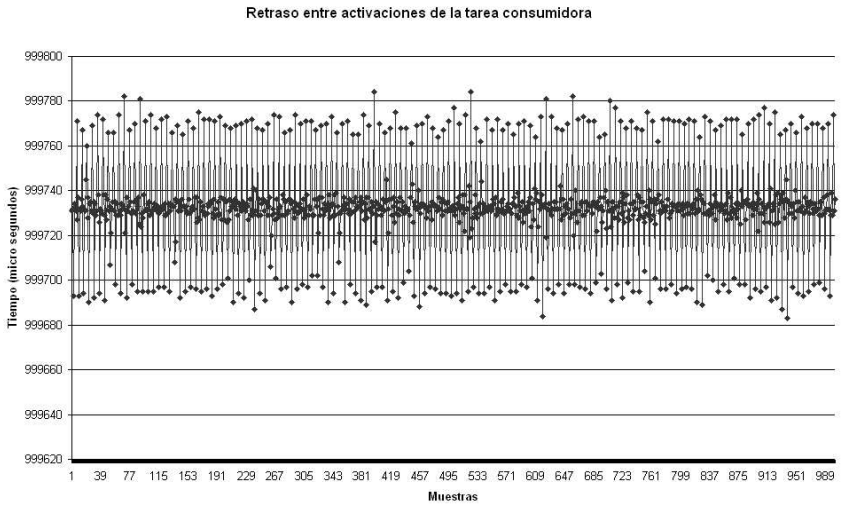


Figura 6.18: Experimento 1: Diferencia temporal entre activaciones de la tarea consumidora, medido en el nodo consumidor

Asimismo la diferencia temporal entre activaciones de la tarea consumidora se mantiene alrededor de 1 segundo(véase 6.18), como era de esperar, con una desviación típica de alrededor de $21ns$ y un *jitter* máximo de $\pm 65ns$, provocado tal vez por la herramienta de medida empleada.

Este experimento muestra que la arquitectura propuesta es capaz de intercambiar perfiles equivalentes de un mismo servicio presentes en nodos diferentes, en tiempo de ejecución, manteniendo la misma calidad de servicio, y de forma transparente al resto de las tareas que constituyen la aplicación.

6.6.4. Dos implementaciones de una tarea productora-consumidora que ofrecen diferente calidad de servicio

En este experimento, se cambia con respecto al anterior un parámetro temporal de uno de los consumidores intermedios, en este caso su tiempo de respuesta, que se refleja en un cambio en el desplazamiento temporal del mensaje que envía dicha tarea. En un caso $\phi_{2A}^m = 100\,T_{EC}$, mientras que en el otro $\phi_{2B}^m = 200\,T_{EC}$. Nótese que los perfiles están ubicados en localizaciones diferentes, es decir, en nodos físicos distintos.

Este desplazamiento temporal u *offset* tiene un impacto directo sobre la calidad de servicio ofrecida por la aplicación, ofreciendo diferentes tiempos de respuesta extremo a extremo en función del perfil empleado.

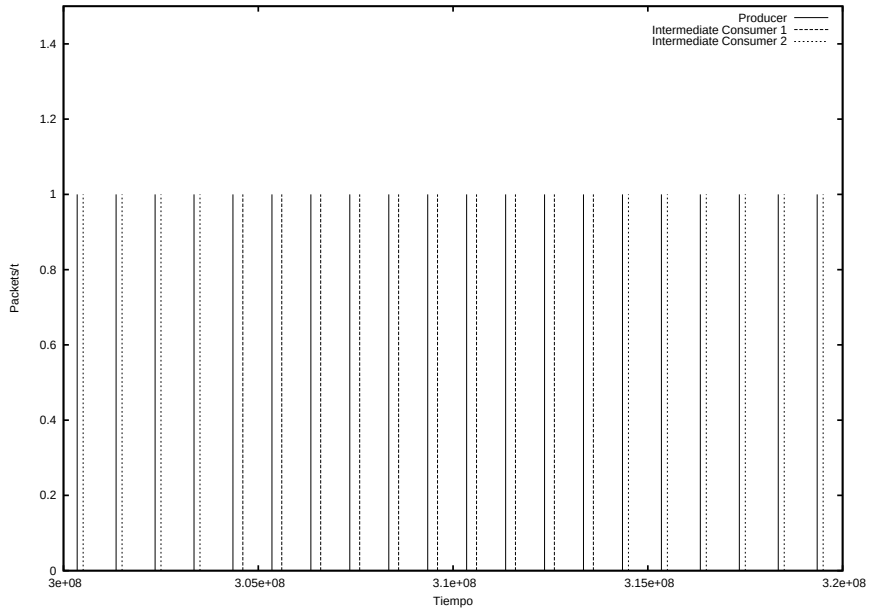


Figura 6.19: Experimento 2: Transmisión de paquetes en la red observada por el programa capturador de tráfico

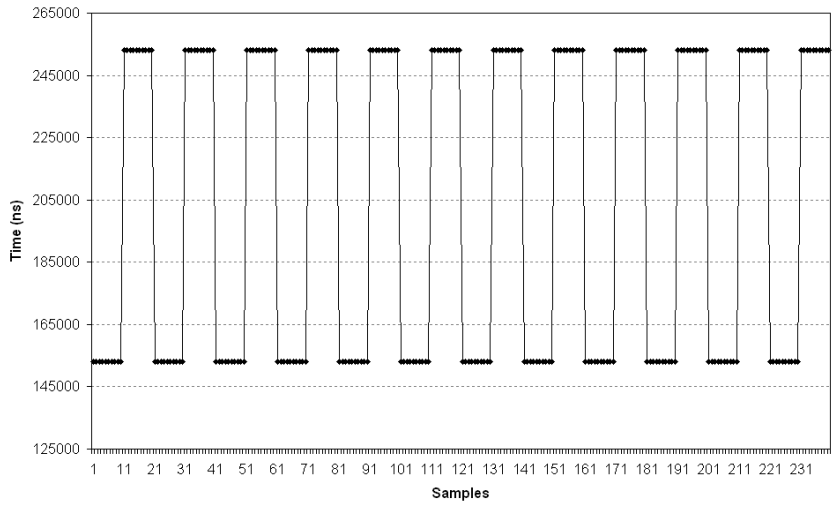


Figura 6.20: Experimento 2: Tiempo de respuesta de los nodos productores medido por el programa capturador de tráfico

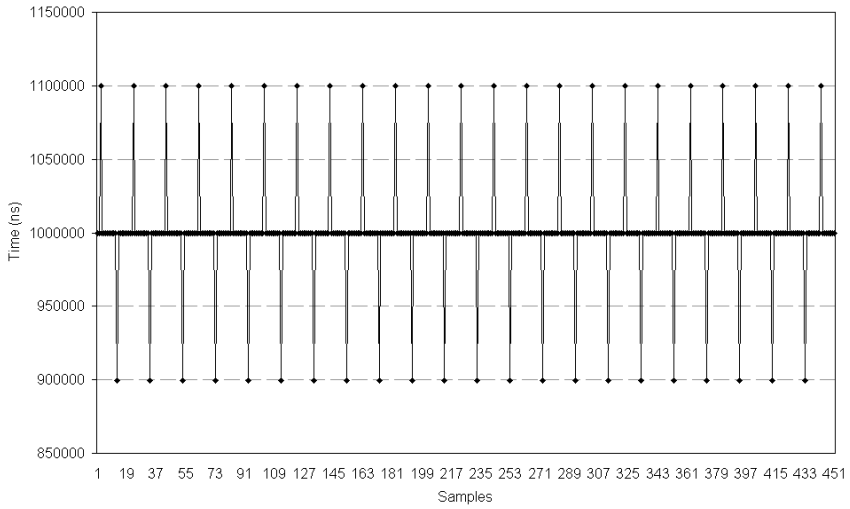


Figura 6.21: Experimento 2: Diferencia temporal entre activaciones de la tarea consumidora, medido en el nodo consumidor

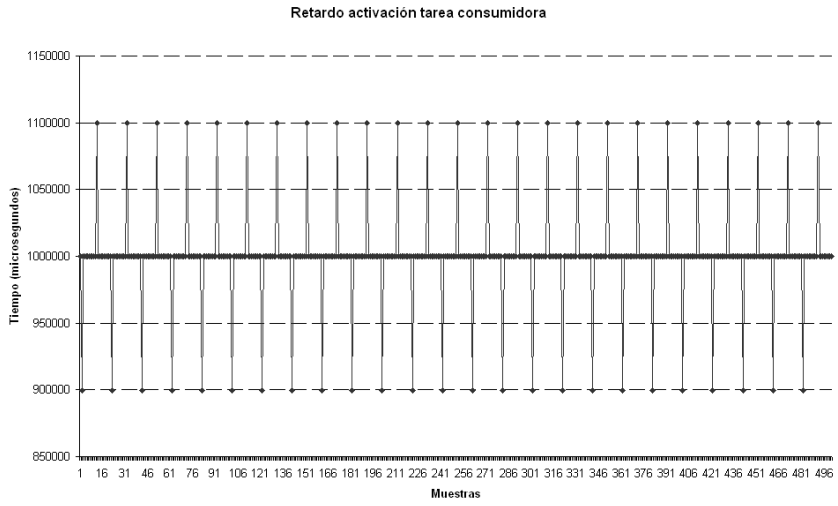


Figura 6.22: Perfiles $2A$ y $2B$ ubicados en el mismo nodo físico ($\phi^m_{2A} = 100\,T_{EC}$, $\phi^m_{2B} = 200\,T_{EC}$)

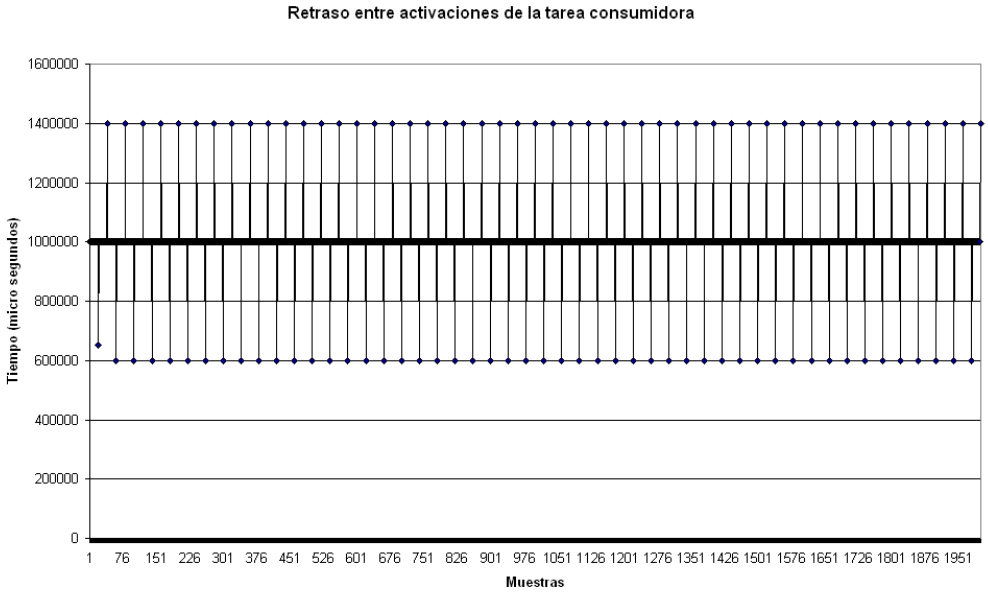
En las figuras 6.19 y 6.20 se puede observar cómo se modifica el tiempo de respuesta de los productores de 150 a 250 milisegundos cada vez que se cambia el perfil empleado, como era de esperar, mientras que el cambio en el tiempo de respuesta extremo a extremo se evidencia en la tarea consumidora: cada vez que se cambia el perfil del consumidor intermedio, hay un retraso o un adelanto en la diferencia temporal entre activaciones de la tarea consumidora, que debe ser periódica, con período $T = 1s$.

El retraso entre activaciones en el nodo consumidor muestra picos de $\pm 100ms$ cuando se produce el cambio debido a la diferencia entre los desplazamientos temporales de los perfiles intercambiados (véase figura 6.21).

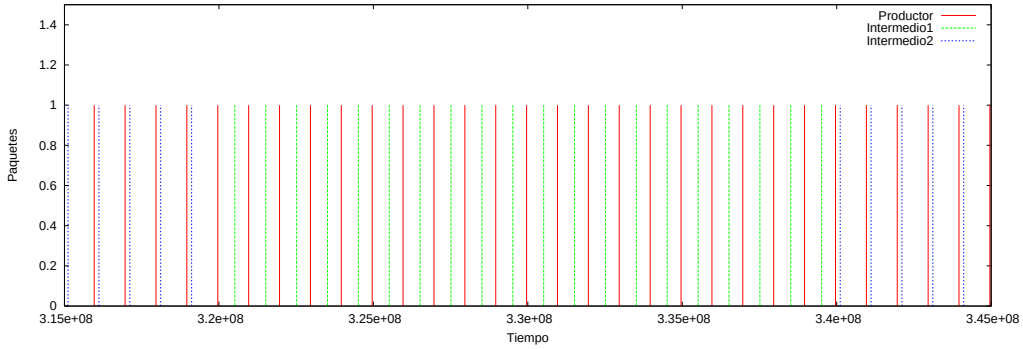
En la figura 6.22 se muestra el mismo experimento, pero localizando los perfiles $2A$ y $2B$ en el mismo nodo físico. Puede observarse que el comportamiento es idéntico, independientemente de su ubicación.

Este experimento muestra que cuando se realiza un intercambio entre dos perfiles diferentes del mismo servicio, éste será transparente al consumidor del mensaje producido por dicho servicio, excepto por la anticipación o retraso en la activación de la tarea consumidora, causada por los diferentes parámetros temporales de los distintos perfiles, lo que lleva a la correspondiente modificación en el tiempo de respuesta extremo a extremo. Este resultado muestra que la arquitectura propuesta puede soportar la actualización dinámica de código, incluso cuando sus características temporales cambian.

Las figuras 6.23 y 6.24 muestran el mismo experimento, modificando el período del hilo que realiza el cambio, $T_C = 20$, y con distintos valores de ϕ_{2A} y ϕ_{2B}.

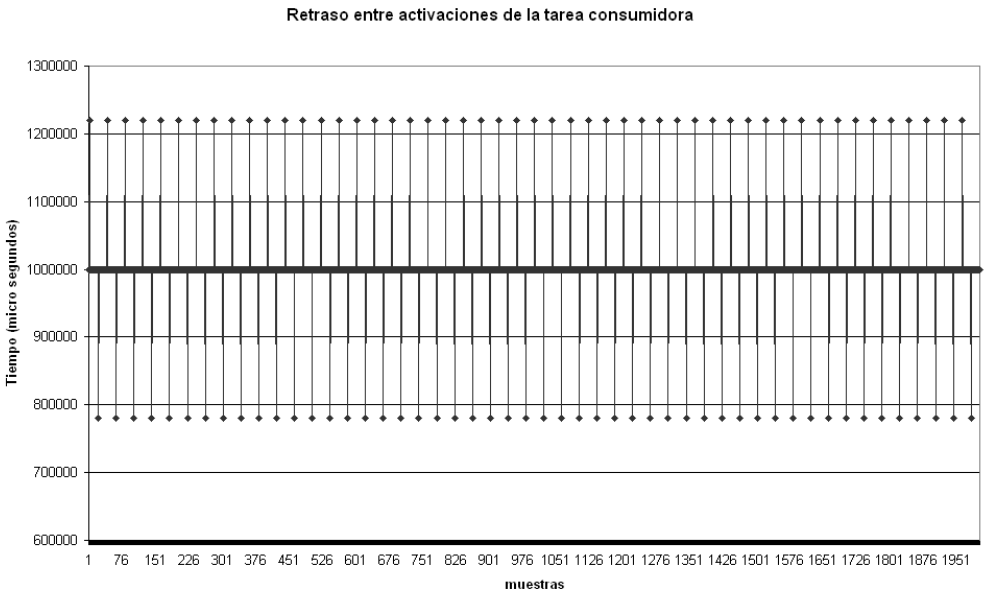


6.23.1: Diferencia temporal entre activaciones de la tarea consumidora

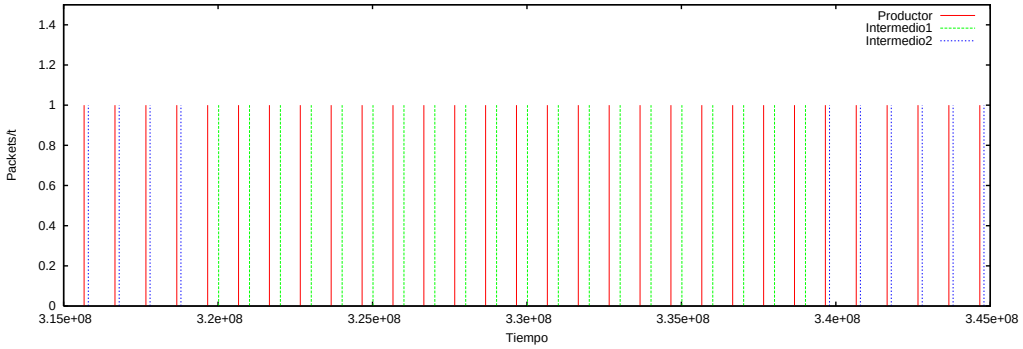


6.23.2: Transmisión de paquetes en la red

Figura 6.23: Perfiles ubicados en nodos físicos distintos ($\phi^m_{2A} = 500\,T_{EC}$, $\phi^m_{2B} = 200\,T_{EC}$, $T_C = 20$)



6.24.1: Diferencia temporal entre activaciones de la tarea consumidora



6.24.2: Transmisión de paquetes en la red

Figura 6.24: Perfiles ubicados en nodos físicos distintos ($\phi^m_{2A} = 300\,T_{EC}$, $\phi^m_{2B} = 80\,T_{EC}$, $T_C = 20$)

6.7. Conclusiones

En este capítulo se ha propuesto una arquitectura sobre un protocolo concreto de comunicaciones de tiempo real, FTT-SE, que soporta composición dinámica de aplicaciones basadas en servicios en un entorno embarcado distribuido de tiempo real. Esta arquitectura permite que una aplicación sea compuesta por distintos perfiles dispersos en la red y el cambio dinámico de los perfiles involucrados en un aplicación de forma transparente a la localización de los perfiles, sus características temporales o al conjunto de consumidores de los mensajes que estos perfiles producen.

La arquitectura se construyó como una extensión al paradigma FTT. Esta extensión requirió la introducción de la noción de servicio y de aplicación basada en servicios en el contexto de FTT. Así como la introducción de pequeñas modificaciones en el protocolo para permitir que éste soportase el cambio entre productores de un mismo mensaje de forma transparente a los consumidores del mismo.

Además, un prototipo de la arquitectura propuesta fue implementado como prueba de concepto, y los resultados experimentales muestran que la arquitectura es capaz de soportar la realización de cambios en el conjunto de perfiles que componen una aplicación, tanto completamente equivalentes, es decir, réplicas con las mismas características temporales, como con distintas características temporales. Lo que implica que esta arquitectura es capaz de soportar la actualización dinámica de código, y, con la implementación de las entidades adecuadas, como un gestor de réplicas o un gestor de QoS, de ofrecer tolerancia a fallos a nivel de aplicación y gestión de la calidad de servicio.

Capítulo 7

Conclusiones y Líneas Futuras

Este capítulo concluye esta tesis doctoral. En primer lugar, se resumen a grandes rasgos las ideas planteadas en la misma. Posteriormente se exponen las principales contribuciones realizadas. Finalmente, se proponen cuestiones de posible estudio futuro que pueden ser desarrolladas a partir del presente trabajo

7.1. Conclusiones generales

Esta tesis se enmarca dentro de la creciente tendencia actual a dotar de una mayor flexibilidad a los sistemas distribuidos de tiempo real debido a la aparición de nuevos entornos de aplicación. Durante el estudio dedicado a los paradigmas que aportan flexibilidad a diversos campos dentro de los sistemas *software*, se detectó que, si bien algunos están siendo aplicados con éxito en sistemas de tiempo real, como las tecnologías basadas en componentes, las soluciones que aportaban estaban lejos de la flexibilidad que ya estaba presente en otros campos, como los servicios Web, basados en el paradigma de orientación a servicios, o en la Computación Ubicua, en la cual la flexibilidad y el dinamismo son características inherentes.

El objetivo inicial que se marcó al comienzo de la presente tesis, era la búsqueda de mecanismos que permitieran implementar arquitecturas de tiempo real distribuido más abiertas, dinámicas y flexibles, destinadas a hacer frente a los nuevos desafíos que el imparable desarrollo tecnológico ofrece. Entre estos destacan: la proliferación de dispositivos cada vez más pequeños y limitados en los que es necesario desarrollar sistemas que ofrezcan garantías temporales y donde no siempre pueden estar presentes todos los servicios necesarios; el avance de la Computación Ubicua que, aunque no está basada en un paradigma de orientación a servicios, tiene puntos de conexión con éste, y podría ser usada, en un futuro, de forma provechosa en el campo de los sistemas de tiempo real; la implantación de sistemas de tiempo real en entornos dinámicos, donde se debe reaccionar a estímulos externos y evolucionar con el entorno. Además de la creciente necesidad de actualizar de manera dinámica y transparente las aplicaciones de tiempo real asegurando garantías temporales, y

la exploración de nuevas vías para el desarrollo de aplicaciones flexibles, tolerantes a fallos y que ofrezcan facilidades para la gestión de calidad de servicio a nivel de aplicación

Así, surgió la idea de desarrollar un sistema completo con garantías temporales que siguiese el paradigma de orientación a servicios, en principio muy alejado del mundo clásico del tiempo real, con el objetivo de ofrecer flexibilidad mediante la composición de aplicaciones. Dentro de este amplio dominio, se contribuyó en un espacio más acotado: la demostración de que se podían adaptar con éxito y de forma beneficiosa características y conceptos propios del paradigma de orientación a servicios a sistemas de tiempo real distribuidos.

Para conseguirlo, se debieron definir modelos, tanto de sistema como de aplicación, servicio y perfil de servicio. Se propusieron distintas arquitecturas genéricas, y se definieron algoritmos de composición de aplicaciones que, haciendo uso de análisis de planificabilidad, permitieran no sólo seleccionar perfiles de servicios en función de sus características funcionales o de calidad de servicio, sino que también proporcionasen la planificación de las aplicaciones compuestas y que asegurasen que las prestaciones del sistema completo no se degradan al incluir una nueva aplicación en el mismo. Además, se implementó un prototipo de una de las arquitecturas propuestas, como forma de validar en un sistema real las ideas que se propusieron.

7.2. Resumen de la tesis y resultados destacables

En esta tesis se han realizado varias contribuciones en el ámbito de la aplicación de conceptos propios del paradigma de orientación a servicios al campo de los sistemas de tiempo real. A continuación se enumeran y explican las principales:

1. *Estudio del estado del arte y de los trabajos relacionados*.

   La demostración de la aplicabilidad de un paradigma a un entorno como los sistemas de tiempo real distribuidos, implica necesariamente que el estudio del estado del arte, realizado en el capítulo 2, no se puede reducir a una tecnología o a un campo específicos, sino que debe abordarse desde una perspectiva global que permita posteriormente realizar decisiones razonadas de diseño. Así, se estudiaron las distintas aproximaciones a la planificación centralizada y distribuida, que se aplicarían posteriormente, en la composición de aplicaciones basadas en servicios. Se estudiaron los distintos modelos de sistema distribuido y se analizaron las aproximaciones de diseño a éstos, para poder decidir el tipo de sistema en el que se basaría la demostración. Se realizó un estudio sobre un conjunto específico de diversas redes y protocolos de comunicaciones de tiempo real, que permitiría elegir el más adecuado para el desarrollo de un prototipo que validase los modelos e ideas propuestas. Se estudió el paradigma de orientación a servicios, centrándonos en las características que podrían ser aprovechadas de manera beneficiosa en sistemas de tiempo real.

Finalmente, se estudiaron diferentes trabajos relacionados que aportaban, realizando diferentes aproximaciones, flexibilidad a sistemas de tiempo real. La mayoría de estos trabajos estaban basados, de una u otra manera, en el concepto de cambio de modo, ya sea empleando arquitecturas de calidad de servicio, tecnologías de componentes o conjungando éstas últimas con programación orientada a aspectos. La flexibilidad que aporta esta aproximación es bastante restringida, pues los modos, la mayoría de las veces, son definidos previamente a la ejecución del sistema y no contemplan, en la mayoría de los casos, la posibilidad de una reconfiguración con nuevas entidades que aparecen y desaparecen dinámicamente en el sistema. Otras soluciones estudiadas sí permiten dicha flexibilidad, pero se abstraen de las propiedades y características de las redes de tiempo real subyacentes, así como del propio sistema. Así, se observó una carencia de soluciones que combinasen flexibilidad, comportamiento temporal predecible y comunicaciones de tiempo real en la aplicación del concepto de servicio a sistemas de tiempo real distribuido, lo que justifica la realización de esta tesis.

2. *Propuesta de un modelo general de sistema y modelos de servicio, de perfil y de aplicación basados en éste*.

   La demostración de la viabilidad de la aplicación del paradigma de orientación a servicios debía partir de un modelo tanto del sistema de tiempo real como de las aplicaciones basadas en servicios que se ejecutarían en éste. Así, en el capítulo 3 se presentan los modelos que en los que se basa esta tesis, enfocados a aplicaciones de tiempo real no crítico.

   Los modelos propuestos se basan en las siguientes características:

   - El modelo se define sobre la abstracción ofrecida por un protocolo de comunicaciones de tiempo real, en este caso se construye directamente sobre el nivel 2 de OSI.
   - Las aplicaciones deseadas por el usuario serán definidas en términos de sus servicios y de las interacciones entre éstos.
   - Los servicios son definidos por su funcionalidad, y son implementados por entidades concretas, denominadas perfiles, con calidades de servicio y características temporales distintas, residentes en nodos físicos y dispersas en la red.
   - Cada una de las invocaciones a un perfil se materializará en una única tarea en el nodo físico. Si un perfil forma parte de varias aplicaciones o es usado varias veces en la misma aplicación, al ser invocaciones diferentes, serán implementadas por tareas distintas asociadas a ese perfil.
   - Una aplicación en ejecución se definirá por el grafo ordenado de tareas y mensajes intercambiados entre éstas.

- Las aplicaciones seguirán la aproximación gobernada por tiempo, es decir, las acciones involucradas en éstas (tareas y mensajes) tendrán instantes de activación predefinidos y siempre mayores que los tiempos de respuesta en el peor caso de las acciones precedentes.
- El modelo de diseño del sistema seguirá una aproximación holística, pues es necesario considerar la aplicación en conjunto, teniendo en cuenta todos los servicios y perfiles utilizados y las tareas que instanciaremos en cada nodo, para poder establecer sus parámetros.
- Para facilitar la dotación de flexibilidad, se decidió que el análisis de planificabilidad sería realizado de manera independiente en cada recurso, aproximación pesimista pero que permite que los tiempos de respuesta en el peor caso de las tareas y mensajes no se vean influenciados por los cambios en los tiempos de activación de ellas mismas, de las tareas precedentes en la aplicación o de tareas con las que comparten recurso.
- La comunicación entre entidades del sistema se realizará de forma asíncrona, para dotar de flexibilidad al sistema completo y que éste permita la inclusión de nuevas aplicaciones, servicios y nodos físicos.

Así, se caracterizó formalmente el concepto de perfil de servicio y se propuso un análisis de compatibilidad entre perfiles y peticiones a éstos. Se presentó la caracterización temporal de un perfil de servicio y se presentó el modelo funcional de aplicaciones basadas en servicios, diferenciando entre la caracterización de la aplicación deseada por un cliente y la de una aplicación en ejecución. Finalmente, se eligió como modelo temporal de aplicación el desarrollado por Calha [47] para aplicaciones de tiempo real gobernadas por tiempo y se propuso una pequeña extensión a éste.

3. *Identificación de funcionalidades, procesos y entidades que deberían estar presentes en una arquitectura de tiempo real no crítico que dé soporte a composición de aplicaciones basadas en servicios*.

   La aplicación de conceptos propios de la orientación a servicios a sistemas y arquitecturas de tiempo real, implica necesariamente una adaptación de las entidades presentes en la orientación a servicios a estos entornos, siendo necesaria una redefinición y establecimiento de los requisitos que deberían cumplir en una arquitectura de tiempo real que los aplique. Éste ha sido uno de los objetivos abordados en el capítulo 4.

   En una arquitectura que soporte orientación a servicios existen tres elementos principales: un *proveedor* que publica sus servicios en un *directorio*, y un *consumidor* que los busca en éste y los utiliza. En este trabajo, cada perfil, implementación concreta de un servicio, será programado por un *programador de servicios* que será el encargado, a través de una *entidad de publicación*, de

publicarlos en un directorio. Los consumidores, *programadores de aplicaciones*, especificarán una aplicación en términos de los servicios que utilizará y será una *entidad compositora* la encargada de realizar la búsqueda a través de una *entidad de búsqueda*, y de elegir a los más adecuados. La elección no sólo dependerá de los requisitos establecidos por el cliente, sino también de los requisitos del sistema: se ha de asegurar que la instanciación de una aplicación no pone en peligro las prestaciones del sistema completo, ni la planificabilidad de ninguna aplicación ya instanciada. Así, la entidad compositora deberá interaccionar con el *gestor del sistema*, quien conoce el estado del mismo a través de *monitorización*.

4. *Identificación de tipos de composición, requisitos arquitectónicos que generan y propuesta de arquitecturas genéricas para cada tipo.*

   En el capítulo 4, se realiza la diferenciación entre composición dinámica y estática, pues estas dos aproximaciones generan diferentes requisitos y necesidades a cubrir en la arquitectura que los implemente:

   - *Composición estática*: una vez realizada no se modifica en tiempo de ejecución. Existe un conocimiento previo de todos los perfiles y servicios existentes en el sistema, con la seguridad de que, una vez compuesta la aplicación, éstos no desaparecerán. El sistema puede decidir permitir composiciones en tiempo de ejecución u obligar a que éstas sean previas a la ejecución del propio sistema.
   - *Composición dinámica*: se realiza en tiempo de ejecución del sistema. Permite cambios en las acciones que componen una aplicación mientras ésta se está ejecutando, es decir, reconfiguración de aplicaciones. Permite, además, la inclusión/desaparición de nuevos elementos (servicios, perfiles, nodos físicos) en el sistema en tiempo de ejecución.

   En el mismo capítulo, se realizan propuestas de arquitecturas para los diferentes tipos de composición, fuertemente influenciadas por la existencia de protocolos de comunicaciones de tiempo real subyacentes, cuestión soslayada o abstraída por propuestas existentes en otros trabajos relacionados.

5. *Propuesta de algoritmos para la composición inicial de aplicaciones basadas en servicios.*

   En el capítulo 5 se estudia la problemática de la composición inicial de aplicaciones de tiempo real basadas en servicios. A la complejidad inherente de la composición como búsqueda del camino óptimo en un grafo multivaluado cumpliendo unas determinadas restricciones, se une la necesidad de que dicho camino sea planificable, es decir, que todas las tareas y mensajes involucrados

sean planificables, que la aplicación completa cumpla sus requisitos temporales extremo a extremo y que el sistema completo no se ponga en peligro ni mermen sus prestaciones debido a la instanciación de la nueva aplicación.

La solución propuesta se planteó como un algoritmo búsqueda exhaustivo que realizaba la planificación de manera global. La elección del camino óptimo se basa en buscar aquél que minimice una determinada figura de mérito global. Esta figura de mérito tiene en cuenta las posibles múltiples restricciones impuestas por el usuario o por el propio sistema, y las características propias de cada acción involucrada. Se definieron distintas figuras de mérito globales, cada una con un objetivo diferente.

La excesiva complejidad de un algoritmo de búsqueda exhaustivo hace inviable su aplicación en contextos donde la composición se realiza en tiempo de ejecución del sistema. Esto se debe a que el número de caminos posibles crece exponencialmente con la profundidad del árbol, lo que implica un crecimiento exponencial en el tiempo de ejecución del algoritmo, pues el tiempo de ejecución de la evaluación de cada camino posible es constante.

Se definió un algoritmo mejorado que reduce el número de combinaciones a explorar. Este algoritmo se basó en el uso de heurísticos, que determinan el número de combinaciones a explorar, y de figuras de mérito relativas, que discriminan los caminos a seguir.

La validación experimental realizada de los diferentes heurísticos reflejó su buen comportamiento temporal, siendo capaces de ofrecer soluciones subóptimas, pero razonablemente cercanas a la solución ofrecida por el exhaustivo, en un tiempo considerablemente inferior, es decir, explorando un menor número de combinaciones.

En entornos de búsqueda como los que se pueden encontrar en sistemas de tiempo real embarcados, donde el número máximo de perfiles disponibles de un servicio puede ser conocido y acotado por un valor, es decir el número de combinaciones puede ser acotado, este algoritmo mejorado puede ser aplicado dando cotas temporales a su ejecución, pues el número de combinaciones que explorará en el peor caso depende directamente del número máximo de combinaciones posibles, pudiéndose determinar analíticamente, y, dado un número de niveles de la aplicación, se puede determinar el tiempo que tardará en evaluar cada combinación. Evidentemente, la aplicabilidad de dichos algoritmos dependerá de las restricciones temporales de cada entorno en concreto.

6. *Propuesta de extensiones para un protocolo de comunicaciones de tiempo real existente, para permitir la definición de una arquitectura para la composición dinámica.*

   La implementación de un prototipo sobre un protocolo concreto de comunicaciones de tiempo real fue uno de los objetivos marcados como medio de

validar las ideas presentadas a lo largo de esta tesis doctoral. En el capítulo 6 se define una arquitectura concreta y se presenta el prototipo implementado.

En base a las necesidades estudiadas en capítulos anteriores, el paradigma elegido fue FTT, en concreto su implementación sobre *Switched Ethernet*. Así, se propone una arquitectura para la composición dinámica de aplicaciones basadas en servicios como una extensión a dicho paradigma. Esta arquitectura simplifica substancialmente el diseño general de tareas del sistema a través de la reutilización de código y del uso de las características de flexibilidad operacional ofrecidas por el paradigma FTT. Además, es fácilmente portable a otras plataformas *hardware* en las que está disponible dicho paradigma.

Los resultados experimentales obtenidos del prototipo implementado muestran que la arquitectura propuesta es capaz de soportar la actualización dinámica de código y, con la implementación de las entidades adecuadas, de ofrecer soporte de tolerancia a fallos a nivel de aplicación y gestión de la calidad de servicio.

7.3. Futuras líneas de trabajo

En esta tesis se ha demostrado la viabilidad de aplicar conceptos propios del paradigma de orientación a servicios al ámbito de los sistemas de tiempo real. A partir de los resultados obtenidos se pueden abrir varias líneas de investigación. Estas posibles líneas incluyen temas como las mejoras al modelo de sistema y los algoritmos de composición, mejoras relacionadas con la implementación de la arquitectura sobre FTT-SE, la implementación de la arquitectura sobre tecnologías Java o el desarrollo de herramientas de ayuda al desarrollador de aplicaciones.

7.3.1. Mejoras al modelo de sistema y a los algoritmos de composición

Existen varias líneas de investigación abiertas sobre el modelo de sistema y los algoritmos de composición planteados que deberán ser abordadas en el futuro. En este apartado se plantean algunas.

7.3.1.1. Desarrollo y evaluación de nuevas figuras de mérito para la composición

Ampliación del espectro de figuras de mérito globales disponibles así como sus respectivas figuras de mérito relativas para que tengan en cuenta factores como la utilidad global del sistema, tal y como se define en [185], u otros relativos a la calidad de servicio de la aplicación global. Elaboración de otras figuras de mérito con mayor semántica de campos de aplicación concretos.

7.3.1.2. Evaluación exhaustiva de los algoritmos de composición

Como línea de trabajo futura, es necesaria una evaluación exhaustiva de todas las figuras de mérito tanto globales como relativas desarrolladas, que permita el análisis de los errores introducidos por el empleo del algoritmo mejorado con diferentes heurísticos y la reducción de éstos.

Por otra parte, los algoritmos de composición presentados (véase capítulo 5) han sido validados mediante simulación. Dentro de los trabajos futuros planteados entra su implementación en una arquitectura concreta y la evaluación de sus prestaciones en tiempo de ejecución del sistema. Así, se podrán realizar pruebas empíricas sobre su comportamiento temporal, ya que éste influye directamente sobre su adecuación para distintos entornos con diferentes restricciones temporales.

7.3.1.3. Extensión de los algoritmos de composición para la inclusión de algoritmos heurísticos de asignación de prioridades

En esta tesis no se ha abordado la cuestión de la asignación de prioridades en entornos distribuidos. Un posible trabajo futuro consiste en el desarrollo de algoritmos de composición que apliquen alguno de los heurísticos de asignación de prioridades existentes [14, 94] y en la evaluación del coste que supone soportar esta característica tanto en la composición dinámica como en la estática.

7.3.1.4. Extensión de los algoritmos de composición para el tratamiento de las características propias de los mensajes

En los algoritmos propuestos se parte de las bases sentadas por el modelo de sistema para el comportamiento de la red, es decir, que los mensajes se diferencian unos de otros únicamente en su longitud y que la red subyacente es la misma para todos los nodos involucrados.

Dentro de los trabajos futuros, debería investigarse sobre la posibilidad de la transmisión de mensajes entre distintos tipos de redes, con pasarelas entre ellas y cómo afecta esta situación a los algoritmos de composición.

La única característica contemplada de los mensajes ha sido su tiempo de transmisión en el peor caso, dado por un análisis de planificabilidad para la red. Podría ser interesante analizar para distintas redes de tiempo real la influencia del ancho de banda utilizado e incluirlo dentro de las figuras de mérito como un dato más a tener en cuenta en la composición de aplicaciones.

7.3.1.5. Extensión del modelo de sistema a sistemas orientados a eventos

El modelo de sistema empleado en esta tesis se basa en la premisa de que las aplicaciones son gobernadas por tiempo, mientras que los mensajes intercambiados entre los diversos nodos físicos y el gestor o compositor son asíncronos. Dentro de

los trabajo futuros se encuentra la extensión del modelo para demostrar que arquitecturas de tiempo real orientadas a eventos pueden también beneficiarse de las ventajas ofrecidas por este paradigma.

Dependiendo del tipo de análisis de planificabilidad empleado, la extensión del modelo puede implicar más o menos cambios en los algoritmos de composición.

Como los tiempos de respuesta de una tarea perteneciente a una transacción en un sistema gobernado por eventos dependen directamente de las tareas precedentes en la transacción y de las tareas en el propio procesador, se deberá aplicar un algoritmo iterativo como el propuesto por Tindell y Clark [221] en la composición. Como este tipo de análisis implica una mayor carga computacional, deberán volver a evaluarse los algoritmos desarrollados, tanto los exhaustivos como los mejorados, y si los tiempos de ejecución de los algoritmos demasiado altos, o su ejecución representa una carga computacional excesiva, desarrollar nuevos heurísticos para el algoritmo mejorado.

Se deberá evaluar la posibilidad de aplicar composición dinámica, es decir, si es posible permitir la reconfiguración de una aplicación compuesta en estos entornos. Para ello, se deberán implementar y evaluar algoritmos de recomposición que, dada una tarea que modifica sus parámetros temporales, recalcule la planificabilidad en ese nodo y en todas las aplicaciones involucradas.

7.3.1.6. Aplicación de técnicas de gestión de calidad de servicio

Los modelos desarrollados fueron pensados para facilitar la inclusión de gestión de la calidad de servicio dentro de la arquitectura. En la presente tesis, al centrarnos sobre todo en la funcionalidad y procesos involucrados en la composición, la gestión de la calidad de servicio se relegó a un segundo plano. Ésta se realizó a un nivel muy simple, a través del control de admisión de aplicaciones en el sistema y ofreciendo la posibilidad de realizar cambios en tiempo de ejecución de los perfiles que constituyen una aplicación determinada.

Una posible línea de investigación consiste en el estudio de diferentes sistemas de gestión de calidad de servicio más complejos y su aplicación a una arquitectura de tiempo real que soporte composición de aplicaciones basadas en servicio. Así como la integración de gestores de recursos, por ejemplo, basados en arquitecturas de capas, como HOLA-QoS [78].

7.3.2. Mejoras a la arquitectura propuesta sobre FTT-SE

Dentro de la arquitectura propuesta sobre FTT-SE, existen todavía cuestiones abiertas susceptibles de ser líneas de trabajo futuro. En este apartado se enumeran algunas de ellas.

7.3.2.1. Implementación completa de la arquitectura propuesta

El prototipo expuesto en el capítulo 6 ha servido como prueba de concepto y de validación de que las ideas propuestas eran aplicables en este entorno. Dentro de los trabajos en desarrollo, se encuentra la implementación completa de la arquitectura propuesta y la inclusión de todos los algoritmos de composición dentro del maestro, así como su evaluación.

7.3.2.2. Gestión de la tolerancia a fallos a nivel de aplicación en la arquitectura

La existencia de múltiples perfiles en una red que implementen la misma funcionalidad se presta a una implementación de tolerancia a fallos en la arquitectura. Una posibilidad es extrapolar al paradigma FTT el modelo definido en la arquitectura TTA [133] donde existen grupos tolerantes a fallos que reaccionan ante un fallo en una de sus unidades. Así, se definirían grupos de perfiles que implementan el mismo servicio con las mismas prestaciones y que, en vez de ser activos, como ocurre en el caso de TTA, son pasivos, será el maestro el que detecte el fallo ocurrido y pase a elegir otro perfil del grupo en la siguiente invocación de la aplicación. Es relativamente sencilla esta implementación de la tolerancia a fallos en *FTT-Ethernet*, ya que al ser *shared Ethernet*, el maestro tiene la posibilidad de recibir toda la información intercambiada por los nodos.

Sin embargo, en FTT-SE, donde los mensajes *unicast* son enrutados directamente por el conmutador y donde puede haber solape temporal entre los mismos si también los enviamos al maestro, no es tan sencillo. Una posibilidad es permitir que el maestro reciba todos los mensajes, suscribiéndolo a todos los flujos intercambiados. Esta opción degrada las prestaciones ofrecidas por la red, puesto que el enlace de bajada del maestro se convierte en el cuello de botella a la hora de planificar los mensajes. Otra posibilidad es que los nodos receptores de los flujos notifiquen al maestro la no recepción de un mensaje mediante el uso de mensajes asíncronos, así el maestro podría reaccionar en como mucho 2 ciclos elementales, substituyendo el perfil en todas las aplicaciones en las que éste ha fallado por un perfil equivalente.

7.3.2.3. Composición inter-dominio en FTT-SE

Una red FTT-SE podría interconectarse con otras a través de *gateways* o pasarelas para incrementar el número de nodos interconectados, si se utiliza el método de sincronización basado en señalización fuera de banda, véase apartado 6.3.2.

Este entorno plantea dos líneas de investigación:

- La gestión de la comunicación inter-dominio en el paradigma FTT y cómo los maestros gestionan y planifican los mensajes que se transmiten en ambas redes.

- La gestión de la composición y de la publicación de perfiles y servicios por parte de los maestros.

Ambas líneas de investigación están interrelacionadas y requieren un arduo trabajo de investigación para, en primer lugar, extender el protocolo FTT para el soporte de este entorno, con la posibilidad de comunicación entre maestros fuera de banda y a través de la pasarela, implementación de planificación jerarquizada, extensión del modelo publicación-suscripción existente en FTT-SE y cambios en los protocolos de enrutado. Posteriormente, se trabajaría en el desarrollo de protocolos entre maestros para la composición dinámica de aplicaciones inter-dominio, de tal forma que mediante *pull* periódico o *push* bajo demanda puedan compartir información de los servicios y perfiles existentes en las redes vecinas.

7.3.3. Implementación de la arquitectura sobre tecnologías Java

En esta tesis se han analizado las entidades que deberían estar presentes en una arquitectura de tiempo real y propuesto diversas opciones para su implementación.

En el grupo de trabajo del Laboratorio de Sistemas de Tiempo Real del departamento de Ingeniería de Telemática, DREQUIEM, en el seno del cual se desarrolló gran parte del trabajo presentado en esta tesis, se han realizado en paralelo, y con la colaboración de la autora de la presente tesis, trabajos sobre el desarrollo de un modelo de distribución de tiempo real basado en RMI, DREQUIEMI [25], así como un conjunto de extensiones tanto para RTRMI [28, 29], como para Java de tiempo real centralizado [26, 30]. Este apartado describe posibles líneas de trabajo futuro que ligan dos de las líneas principales de investigación dentro del grupo DREQUIEM.

7.3.3.1. Implementación de una arquitectura de composición estática sobre RT-RMI

Un posible trabajo futuro, que se está considerando realizar, es la extensión del modelo de distribución definido por el grupo DREQUIEM, para que dé soporte a abstracciones de orden superior como pueden ser las aplicaciones de tiempo real basadas en servicios. Una vez realizada dicha extensión, se podría implementar una arquitectura que dé soporte a la composición estática de aplicaciones basadas en servicios remotos.

En primer lugar, se deberán realizar extensiones sobre la actual tecnología JINI para que, además de las extensiones ya realizadas durante la realización de la presente tesis para la publicación de información de tiempo real y calidad de servicio asociados a los servicios [80], los mecanismos de descubrimiento tengan un comportamiento temporal predecible y, así, poder acotar los tiempos de composición de las aplicaciones.

Por otra parte, se deberán implementar en RT-Java los algoritmos de composición propuestos en el capítulo 5, y desarrollar un protocolo de composición entre la

entidad compositora y los nodos en los cuales se ubican dichos servicios. Además de estudiar los posibles problemas que se deriven de la implementación de un protocolo distribuido de composición.

7.3.3.2. Implementación de una arquitectura de composición dinámica sobre RT-RMI basada en el paradigma FTT

Uno de los trabajos realizados durante las estancias de algunos de los miembros de DREQUIEM en la Universidade de Aveiro (Portugal), fue la definición de un servicio de sincronización de tiempo real basado en el paradigma FTT sobre DREQUIEMI [27].

Pese a que éste es un protocolo de sincronización, el trabajo desarrollado ha introducido la posibilidad de enriquecer el modelo de DREQUIEMI con técnicas asíncronas de tipo *publisher-suscriber*. Uno de los resultados de este trabajo es la posibilidad de aplicar los principios rectores de FTT al nivel de *software* intermedio, lo que abre el camino al desarrollo de una arquitectura similar a la presentada en el capítulo 6 sobre RT-RMI.

7.3.4. Implementación de herramientas de ayuda al diseñador

Dentro de los trabajos futuros relacionados con los resultados obtenidos por esta tesis, se encuentra el desarrollo de herramientas que, de forma automática, permitan el diseño de aplicaciones basadas en servicios y su despliegue. En el momento de la escritura del presente documento, se está desarrollando un proyecto fin de carrera bajo la supervisión de la autora, que a partir de diagramas UML en una herramienta genérica, genera el código adecuado para desplegarlo en el prototipo desarrollado sobre el paradigma FTT.

Se está estudiando, además, el desarrollo de una herramienta similar a la desarrollada por Calha [47], pero que, en vez de partir de la especificación en XML, utilice UML para la definición de las aplicaciones, servicios y perfiles. Esta herramienta permitirá al desarrollador elegir entre los algoritmos de composición propuestos en el capítulo 5 y, así, realizar simulaciones del sistema antes de su ejecución.

Bibliografía

[1] L. Abeni and G. Buttazzo. Resource reservation in dynamic real–time systems. *Real–Time Systems*, 27(2):123–165, 2004.

[2] L. Abeni and Giorgio Buttazzo. Integrating Multimedia applications in hard real-time systems. In *Proc. of the 19th IEEE Real–Time Systems Symposium*, Madrid, Spain, Diciembre 1998.

[3] Luis Almeida. *Flexibility and Timeliness in Fieldbus–based Real–Time Systems*. PhD thesis, Departamento de Electrónica e Telecomunicações of the University of Aveiro, Aveiro, Portugal, 1999.

[4] Luis Almeida, Paulo Pedreiras, and Jose Fonseca. The FTT–CAN protocol: why and how. *IEEE Transactions on Industrial Electronics*, 49(6):1189–1201, December 2002.

[5] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications*. Springer, 2004.

[6] James H. Anderson, Rohit Jain, and Srikanth Ramamurthy. Wait-free object-sharing schemes for real–time uniprocessors and multiprocessors. In *Proc. of the 18th IEEE Real–Time Systems Symposium*, pages 111–122, 1997.

[7] James H. Anderson, Srikanth Ramamurthy, and Kevin Jeffay. Real–Time Computing with Lock–Free Shared Objects. *ACM Transactions on Computer Systems*, 15(2):134–165, 1997.

[8] T. Andrews, F. Curbera, H. Dholakia, et al. *Business Process Execution Language for Web Services. Version 1.1 Specification*, 2003. Available at http://www-106.ibm.com/developerworks/webservices/library/ws-bpel.

[9] Jesus Arias-Fisteus. *Definición de un Modelo Formal para la Verificación de Procesos de Negocio*. PhD thesis, Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid, 2005.

[10] ARINC/RTCA-SC-182/EUROCAE-WG-48, Minimal Operational Performance Standard for Avionics Computer Resources, 1999.

[11] ARINC, Aeronautical Radio, Incorporated. *664P7 Aircraft Data Network, Part 7, Avionics Full Duplex Switched Ethernet (AFDX) Network*, Junio 2005. Available at http://www.arinc.com.

[12] Artist FP5 Consortium. *Artist FP5 Consortium: Embedded Systems Design*, volume LNCS 3436, chapter Networks 24, pages 316–337. Springer-Verlag, Berlin Heidelberg, 2005.

[13] N. Audsley. Deadline Monotonic Scheduling. Technical Report YCS_146, Department of Computer Science, University of York, Septiembre 1991.

[14] N. Audsley. Optimal Priority Assigment and Feasibility of Static Priority Tasks with Arbitrary Start Times. Technical Report YCS_164, Department of Computer Science, University of York, Diciembre 1991.

[15] N. Audsley. *Flexible Scheduling of Hard Real–Time Systems*. PhD thesis, Department of Computer Science. University of York, 1993.

[16] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings. Applying new scheduling theory to static priority pre–emptive scheduling. *Software Engineering Journal*, 8(5):284–292, Septiembre 1993.

[17] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Hard real–time scheduling: The deadline–monotonic approach. In *Proc. 8th IEEE Workshop on Real–Time Operating Systems and Software*, Mayo 1991.

[18] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings. Incorporating unbounded algorithms into predictable real–time systems. *Computer Systems Science and Engineering*, 8(3):80–89, 1991.

[19] A. Avizienis. Toward systematic design of fault-tolerant systems. *Computer*, 30(4):51–58, April 1997.

[20] Algirdas A. Avizienis. The Methodology of N-Version Programming. In Chen Lyu, editor, *Software Fault Tolerance*, pages 25–46. John Wiley & Sons Ltd, 1995.

[21] T. P. Baker. Stack-based scheduling for realtime processes. *Real-Time Systems*, 3(1):67–99, 1991.

[22] Theodore P. Baker and Alan C. Shaw. The cyclic executive model and ada. *Real-Time Systems*, 1(1):7–25, 1989.

[23] S.K. Baruah, R.R. Howell, and L.E. Rosier. Algorithms and complexity concerning the preemptive scheduling of periodic real–time tasks on one processor. *Real–Time Systems*, 2:173–179, 1990.

[24] S.K. Baruah, A.K. Mok, and L.E. Rosier. Preemptively scheduling hard real–time sporadic tasks on one processor. In *Proc. 11th IEEE Real–Time Systems Symposium*, pages 182–190, 1990.

[25] Pablo Basanta-Val. *Técnicas y extensiones para Java de tiempo real distribuido*. PhD thesis, Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid., Febrero 2007.

[26] Pablo Basanta-Val, Iria Estevez-Ayres, and Marisol Garcia-Valls. AGCMemory: A new Real-Time Java Region Type for Automatic Floating Garbage Recycling. *ACM SIGBED Review*, 2(3), July 2005.

[27] Pablo Basanta-Val, Luís Almeida Marisol García-Valls, and Iria Estévez-Ayres. Towards a Synchronous Scheduling Service on Top of an Unicast Distributed Real-Time Java. In *Proc. of the RTAS 2007 13th IEEE Real-Time Application Symposium*, Santorini Island, Greece, 2007.

[28] Pablo Basanta-Val, Marisol García-Valls, and Iria Estévez-Ayres. No Heap Remote Objects: Leaving Out Garbage Collection at the Server Side. In *OTM Workshops*, pages 359–370, 2004.

[29] Pablo Basanta-Val, Marisol Garcia-Valls, and Iria Estevez-Ayres. Towards the Integration of Scoped Memory in Distributed Real-Time Java. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 382–389, Washington, DC, USA, 2005. IEEE Computer Society.

[30] Pablo Basanta-Val, Marisol García-Valls, and Iria Estévez-Ayres. Extended-portal: violating the Assignment Rule and enforcing the single parent one. In *4th International Workshop on Java Technologies for Real-Time and Embedded Systems*, Octubre 2006.

[31] Guillem Bernat. *Specification and Analysis of Weakly Hard Real–Time Systems*. PhD thesis, Department de Ciències Matemàtiques i Informàtica. Universitat de les Illes Balears, 1998.

[32] Bernecker + Rainer Industrie-Elektronic Ges.m.b.H. *ETHERNET Powerlink Data Transport Services White-Paper Ver. 0005*, September 2002. Available at http://www.ethernet-powerlink.org.

[33] R. Bettati and J. Liu. End-to-End Scheduling to Meet Deadlines in Distributed Systems. In *Proc. of the 12th International Conference on Distributed Systems*, pages 452–459, Japan, June 1992.

[34] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[35] F. Bogenberger, B. Muller, and T. Fuher. Protocol overview. In *Proceedings of the 1st FlexRay International Workshop: The Communication System for Advanced Automotive Control Applications*, Munich, Germany, April 2002.

[36] J.-Y. L. Boudec and P. Thiran. Network calculus. a theory of deterministic queuing systems for the internet. *LNCS*, 2050, 2000.

[37] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. *Simple Object Access Protocol (SOAP) 1.1. W3C Note*. World Wide Web Consortium, May 2000. Available at http://www.w3.org/TR/2000/NOTE-SOAP-20000508.

[38] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A Dynamic Quality of Service Middleware Agent for Mediating Application Resource Usage. In *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, page 307, Washington, DC, USA, 1998. IEEE Computer Society.

[39] Alan Burns, Neil Hayes, and M.F. Richardson. Generating feasible cyclic schedules. *Control Engineering and Practice*, 3(2):151–162, 1995.

[40] Alan Burns and Andy Wellings. *Real–Time Systems and their programming languages*. Addison Wesley, second edition, 1996.

[41] Giorgio Buttazzo. Research trends in real-time computing for embedded systems. *ACM SIGBED Rev. Special issue on major international initiatives on real-time and embedded systems*, 3(3):1–10, July 2006.

[42] Giorgio Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real Time–Systems*, 23(1–2):7–24, 2002.

[43] Giorgio Buttazzo and L. Abeni. Smooth rate adaptation through impedance control. In *Proc. of the 14th Euromicro Conference on Real–Time Systems*, pages 3–10, Vienna, Austria, 2002.

[44] Giorgio Buttazzo, G. Lipari, M. Caccamo, and L. Abeni. Elastic scheduling for flexible workload management. In *IEEE Transactions on Computers*, volume 51 of *3*, pages 289–302, March 2002.

[45] Mario Caccamo, Giorgio Buttazzo, and L. Sha. Capacity sharing for overrun control. In *Proc. of the 21th IEEE Real–Time Systems Symposium*, pages 295–304, Orlando, FL, USA, Diciembre 2000.

[46] M. Calha and J.A. Fonseca. Data streams – an analysis of the interactions between real–time tasks. In *ETFA 2005, 10th IEEE Conference on Emerging Technologies for Factory Automation*, pages 375– 380, Catania, Italy, September 2005.

[47] Mario Calha. *A Holistic Approach Towards Flexible Distributed Systems*. PhD thesis, DET / IEETA-LSE. Universidade de Aveiro, 2006.

[48] Celeste Campo. *Tecnologías Middleware para el Desarrollo de Servicios en Entornos de Computación Ubicua*. PhD thesis, Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid, 2004.

[49] G. M. Candea and M. B. Jones. Vassal: loadable scheduler support for multi-policy scheduling. In *Proc. of the 2nd USENIX Windows NT Symposium*, pages 157 – 66, 1998.

[50] J. Cano, J.C. Perez-Cortes, J.M. Valiente, R. Paredes, and J. Arlandis. Textile Inspection with a Parallel Computer Cluster. In *5th International Conference on Quality Control By Artificial Vision. QCAV-2001*, Le Creusot (France), 2001.

[51] Salvatore Cavalieri, Salvatore Monforte, Eduardo Tovar, and Francisco Vasques. Multi-Master Profibus DP Modelling and Worst Case Analysis-Based Evaluation. In *XV IFAC World Congress on Automatic Control*, Barcelona, Spain, July 2002. IFAC.

[52] CENELEC, European Committe for Electrotechnical Standadisation. *European Standard EN 50170: Fieldbus: Vol. 1:P-Net; Vol.2: PROFIBUS; Vol.3: WorldFIP*, 1996.

[53] Samarjit Chakraborty, Yanhong Liu, Nikolay Stoimenov, and Lothar Thiele an Ernesto Wandeler. Interface–Based Rate Analysis of Embedded Systems. In *Proceedings of the 27th IEEE Real-Time Systems Symposium*, pages 25–34, Diciembre 2006.

[54] M. Chen and K. Lin. Dynamic priority ceilings: A concurrency control protocol for real–time systems. *Real–Time Systems*, 2(4):325–346, 1990.

[55] Shigang Chen and Klara Nahrstedt. On finding multi-constrained paths. In *Proc. of IEEE International Conference on Communications, 1998. ICC 98. Conference Record.1998*, volume 2, pages 874–879, June 1998.

[56] Byung-Kyu Choi, Sangig Rho, and Riccardo Bettati. Dynamic Resource Discovery for Applications Survivability in Distributed Real-Time Systems. In *Proc. of the International Parallel and Distributed Processing Symposium, IPDPS*, page 122, 2003.

[57] Byung-Kyu Choi, Sangig Rho, and Riccardo Bettati. Fast Software Component Migration for Applications Survivability in Distributed Real-Time Systems. In *Proc. of the 7th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing ISORC 2004*, pages 269–276, 2004.

[58] Concurrent Systems Architecture Group. Agile objects project, August 2004. Disponible en http://www-csag.ucsd.edu/projects/agileO.html.

[59] FlexRay Consortium. *FlexRay Requirements Specification Version 1.9.7*, September 2001.

[60] G. Coulouris, J. Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. International Computer Science Series. Addison-Wesley Longman, Inc., 4nd edition edition, June 2005.

[61] Robert Davis and Andy J. Wellings. Dual Priority Scheduling. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

[62] M. A. de Miguel, J. Ruiz, and M. García-Valls. QoS–Aware Component Frameworks. In *Proc. of the 10th International Workshop on Quality of Service*, pages 161–169, May 2002.

[63] Miguel A. de Miguel. Integration of qos facilities into component container architectures. In *International Symposium on Object-Oriented Real-Time Distributed Computing, ISORC 2002*, pages 394–401, 2002.

[64] C. Y. Tatibana; R. Silva de Oliveira and C. Montez. Dynamic Guarantee in Component-Based Distributed Real-Time Systems. In *Proc. of 10th IEEE Conference on Emerging Technologies and Factory Automation, 2005. ETFA 2005*, volume 1, pages 7–14, Sep 2005.

[65] Geert Deconinck, Theodora A. Varvarigon Vincenzo De Florio, and Evangelos Verentziotis. The EFTOS Approach to Dependability in Embedded Supercomputing. *IEEE Transactions on reliability*, 51(1):76, March 2002.

[66] M. L. Dertouzos. Control robotics: the procedural control of physical processses. *Information Processing*, page 74, 1974.

[67] E. Dijkstra. *Cooperating Sequential Processes*. Academic Press, 1968.

[68] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik,* 1:269–271, 1959.

[69] Iria Estévez-Ayres, Luís Almeida Marisol García-Valls, and Pablo Basanta-Val. An Architecture to Support Dynamic Service Composition in Distributed Real-Time Systems. In *Proc. of the ISORC 2007 10th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, Santorini Island, Greece, 2007.

[70] Iria Estévez-Ayres, Marisol García-Valls, and Pablo Basanta-Val. Ad hoc composition of service based real-time applications. In *FALSE-II 2nd Workshop on High Performance, Fault Adaptative, Large Scale Embedded Real-Time Systems*, San Francisco (California), USA, Marzo 2005.

[71] Iria Estevez-Ayres, Marisol Garcia-Valls, and Pablo Basanta-Val. Enabling WCET–based Composition of Service–based Real–Time Applications. *ACM SIGBED Review*, 2(3):25–29, July 2005.

[72] Iria Estévez-Ayres, Marisol García-Valls, and Pablo Basanta-Val. Hacia la integración del cálculo del WCET en la composición de aplicaciones de tiempo real basadas en servicios. In Thomson Paraninfo, editor, *Simposio de Sistemas de Tiempo Real, I Congreso Español de Informática*, Granada, España, Septiembre 2005.

[73] Iria Estévez-Ayres, Marisol García-Valls, and Pablo Basanta-Val. Static Composition of Service-Based Real-Time Applications. In *Third IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2005), 16-17 May 2005, Seattle, WA, USA*, pages 11–15. IEEE Computer Society, 2005.

[74] Greg Bollela et al. *The Real-Time Specification for Java*. Adisson-Wesley, 2001.

[75] *ETHERNET Powerlink protocol*. Available at http://www.ethernet-powerlink.org.

[76] FSMLabs, Inc. Copyright Finite State Machine Labs Inc. 2001-2005. *Real–time Programming in RTCore*, November 2005.

[77] Xiaodong Fu, Weisong Shi, Anatoly Akkerman, and Vijay Karamcheti. CANS: Composable, Adaptive Network Services Infrastructure. In *3rd USENIX Symposium on Internet Technologies and Systems, USITS 2001*, pages 135–146, San Francisco, California, USA, 2001.

[78] Marisol García-Valls. *Calidad de servicio en sistemas multimedia empotrados mediante gestión dinámica de recursos*. PhD thesis, Universidad Politécnica de Madrid, Julio 2001.

[79] Marisol García-Valls, Alejandro Alonso-Muñoz, José Ruiz, and Angel Groba. An Architecture of a Quality of Service Resource Manager Middleware for Flexible Embedded Multimedia Systems. In Springer-Verlag, editor, *Proc. of the 3rd International Workshop on Software Engineering and Middleware (SEM 2002)*, number 2596 in LNCS, pages 36–55, Orlando (Florida), USA, November 2002.

[80] Marisol García-Valls, Iria Estévez-Ayres, Pablo Basanta-Val, and Carlos Delgado-Kloos. CoSeRT: A Framework for Composing Service-Based Real-Time Applications. In Christoph Bussler and Armin Haller, editors, *Business Process Management Workshops*, volume 3812 of *Lecture Notes in Computer Science*, pages 329–341. Springer, 2005.

[81] R. Garey and D. Johnson. Complexity Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM Journal of Computing*, 4(3):187–200, 1975.

[82] R. Garey and S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4(4):397–411, 1975.

[83] J. Goodenough and L. Sha. The priority ceiling protocol: a method for minimizing the blocking of high priority ada tasks. *Ada Letters*, 8(7), 1988.

[84] Steven D. Gribble, Matt Welsh, Rob von Behren, Eric A. Brewer, David Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, B. Zhao, and Robert C. Holte. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.

[85] Object Management Group. Uml profile for modeling and analysis of real-time and embedded systems (marte) rfp, February 2005. Available at http://www.omg.org/cgi- bin/doc?realtime/2005- 02- 06.

[86] XML Protocol Working Group. *SOAP Version 1.2 W3C Recommendation 24 June 2003*. World Wide Web Consortium, June 2003. Available at http://www.w3.org/TR/soap.

[87] X. Gu, A. Messer, I. Greenberg, D. Milojicic, and K. Nahrstedt. Adaptive offloading for pervasive computing. *IEEE Pervasive Computing Magazine*, 3(3), July 2004.

[88] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS–Assured Service Composition in Managed Service Overlay Networks. In *Proc. of IEEE 23nd International Conference on Distributed Computing Systems (ICDCS 2003)*, 2003.

[89] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML–based Quality of Service Enabling Language for the Web. *Journal of Visual Language and Computing, Special Issue on Multimedia Language for the Web*, 13(1):61–95, February 2002.

[90] Xiaohui Gu. *Spidernet: a Quality–Aware Service Composition Middleware*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champain, Urbana, Illinois, 2004.

[91] Xiaohui Gu and Klara Nahrstedt. Distributed multimedia service composition with statistical qos assurances. *IEEE Transactions on Multimedia*, 8(1):141–151, February 2006.

[92] Xiaohui Gu and Klara Nahrstedt. On composing stream applications in peer–to–peer environments. *IEEE Transactions on Parallel and Distributed Systems*, 17(8):824–837, August 2006.

[93] Xiaohui Gu, Zhen Wen, and Philip S. Yu. Bridgenet: An Adaptive Multi-Source Stream Dissemination Service Overlay. In *Proc. of the 26th Annual IEEE Conference on Computer Communications IEEE INFOCOM Mini-Symposium 2007*, Anchorage, Alaska, USA, May 2007.

[94] J.J. Gutierrez-García and Michael Gonzalez-Harbour. Optimized Priority Assignment for Tasks and Messages in Distributed Real–Time Systems. In *Proc. of the 3rd Workshop on Parallel and Distributed Real–Time Systems*, pages 124–132, April 1995.

[95] J.J Gutiérrez-García. *Planificación, análisis y optimización de sistemas distribuidos de tiempo real estricto*. PhD thesis, Grupo de Computadores y Tiempo Real. Universidad de Cantabria, 1995.

[96] Wei Hao, Tong Gao, I-Ling Yen, Yinong Chen, and Raymond Paul. An infrastructure for web services migration for real-time applications. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pages 41–48, Washington, DC, USA, October 2006. IEEE Computer Society.

[97] P. K. Harter. Response times in level structured systems. Technical Report CU–CS–269–94, Department of computer Science. University of Colorado, USA, 1984.

[98] P. K. Harter. Response times in level structured systems. *ACM Transactions on Computer Systems*, 5:232–248, 1997.

[99] Johannes Helander and Stefan Sigurdsson. Self–Tuning Planned Actions Time to Make Real–Time SOAP Real. In *ISORC '05: Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pages 80–89, Washington, DC, USA, May 2005. IEEE Computer Society.

[100] C. A. R. Hoare. Monitors: An operating system structuring concept. *Communications of the ACM*, 17(10):549–557, 1974.

[101] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall Int, 1985.

[102] Yu-Shing Hu. *A Portable Worst–Case Execution Time Analysis Framework for Real-Time Java Architectures.* PhD thesis, Real–Time Systems Research Group. Department of Computer Science. University of York, 2004.

[103] M. H. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, January/February 2005.

[104] IEC International Electrotechnical Comittee. *IEC International Standard 61158: Fieldbus standard for use in industrial control systems - Type 1: Existing IEC TS61158 parts3-6 (Foundation Fieldbus H1); Type 2: ControlNet; Type 3: PROFIBUS; Type 4:P-Net; Type 5: Fieldbus Foundation HSE; Type 6: SwiftNet; Type 7: WorldFIP; Type 8: Interbus-S*, 2000.

[105] IEEE. *IEEE 802.3 10BASE3 standard*.

[106] IEEE. *IEEE 802.3 10BASE5 standard*.

[107] IEEE. *IEEE 802.3ae-2002 - 10Gbps*.

[108] IEEE. *IEEE 802.3c 1BASE5 StarLan standard*.

[109] IEEE. *IEEE 802.3i 10BASE-T standard*.

[110] IEEE. *DIX Ethernet V2.0 specification*, 1982.

[111] IETF. An architecture for differentiated services. RFC 2475, 1980.

[112] IETF. Resource reservation protocol (rsvp) – version 1 applicability statement some guidelines on deployment. RFC 2208, 1997.

[113] IETF. Specification of guaranteed quality of service. RFC 2212, 1997.

[114] ISO. *Road Vehicles - Control Area Network (CAN) - Part 4: Time-Triggered Communication*, 2001.

[115] ISO-11519-2. *Road vehicles - Low-speed serial data communication - Part 2: Low-speed controller area network (CAN)*, 1994.

[116] ISO-11898. *Road vehicles - Interchange of digital information - Controller area network (CAN) for high-speed communication*, 1993.

[117] D. Isovic and C. Norström. Components in Real–time Systems. In *Proc. of the 8th Conf. on Real–Time Computing Systems and Applications, Tokyo*, 2002.

[118] J. M. Jaffe. Algorithms for finding paths with multiple constraints. *Networks*, 14:95–114, 1984.

[119] K. Jeffay. Scheduling sporadic tasks with shared resources in hard–real–time systems. In *Proc. of the 13th IEEE Real–Time Systems Symposium*, pages 89–99, Phoenix, USA, 1992.

[120] Jingwen Jin. *QoS-Aware Service Management for Internet-Scale Distributed Applications*. PhD thesis, Department of Computer Science, University of Illinois at Urbana-Champain, 2006.

[121] Jingwen Jin and Klara Nahrstedt. A Distributed Approach for QoS Service Multicast with Geometric Location Awareness. *IEEE Distributed Systems Online*, 4(6), June 2003.

[122] M. B. Jones, P. J. Leach, R. P. Draves, and J. S. Barrera. Modular real-time resource management in the rialto operating system. In *HOTOS '95: Proceedings of the Fifth Workshop on Hot Topics in Operating Systems (HotOS-V)*, page 12, Washington, DC, USA, 1995. IEEE Computer Society.

[123] Michael B. Jones, Daniel L. McCulley, Alessandro Forin, Paul J. Leach, Daniela Roşu, and Daniel L. Roberts. An overview of the rialto real-time architecture. In *EW 7: Proceedings of the 7th workshop on ACM SIGOPS European workshop*, pages 249–256, New York, NY, USA, 1996. ACM Press.

[124] Steve Jones. Toward an acceptable definition of service. *IEEE Software*, 22(3):87–93, 2005.

[125] M. Joseph and P. Pandya. Finding Response Times in a Real–Time System. *BCS Computer Journal*, 29(5):390–395, Octubre 1986.

[126] JSR-50. Distributed real-time specification, 2000. Available on-line at http://www.jcp.org/en/jsr/detail?id=50.

[127] Tero Kangas, Petri Kukkala, Heikki Orsila, Erno Salminen, Marko Hännikäinen, Timo D. Hämäläinen, Jouni Riihimäki, and Kimmo Kuusilinna. UML–based multiprocessor SoC design framework. *ACM Trans. on Embedded Computing Sys.*, 5(2):281–320, Mayo 2006.

[128] Ben Kao and Hector Garcia-Molina. Deadline Assignment in a Distributed Soft Real–Time System. *IEEE Transactions on Parallel and Distributed Systems*, 8(12):1268–1274, Diciembre 1997.

[129] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. M. Loingtier, and J. Irwin. Aspect-oriented programming. In Springer-Verlag, editor, *Proc. of the 11th European Conference on Object-Oriented Programming (ECOOP'97)*, volume 1241 of *Lecture Notes in Computer Science*, pages 220–242, 1997.

[130] Mark H. Klein and T. Ralya. An analysis of input/output paradigms for real–time systems. Technical report, Software engineering institute. CMU/SEI–90–TR–19, 1990.

[131] Hermann Kopetz. *Real–Time Systems: Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[132] Hermann Kopetz, Astrit Ademaj, Petr Grillinger, and Klaus Steinhammer. The Time–Triggered Ethernet (TTE) Design. *8th IEEE International Symposium on Object-oriented Real-time distributed Computing (ISORC), Seattle, Washington*, pages 22–33, May. 2005.

[133] Hermann Kopetz and Günther Bauer. The Time–Triggered Architecture. *Proceedings of the IEEE, Special Issue on Modeling and Design of Embedded Software*, 91(1):112–126, January 2003.

[134] Hermann Kopetz and Günter Grünsteidl. TTP: A Protocol for Fault-Tolerant Real-Time Systems. *Computer*, 27(1):14–23, 1994.

[135] T. Korkmaz and M. Krunz. A randomized algorithm for finding a path subject to multiple qos constraints. In *Proc. of the IEEE Global Telecommunications Conference*, pages 1694–1698, 1999.

[136] T. Korkmaz, M. Krunz, and S. Tragoudas. An efficient algorithm for finding a path subject to two additive constraints. *Computer Communications*, 25:225–238, 2002.

[137] Lakshman Krishnamurthy. *AQUA: an adaptive quality of service architecture for distributed multimedia applications*. PhD thesis, University of Kentucky, Lexington, KY, USA, 1997.

[138] C. Lee, R. Rajkumar, and C. Mercer. Experiences with processor reservation and dynamic qos in real-time mach. In *Proc. of Multimedia Japan 96*, April 1996.

[139] Chen Lee. *On Quality of Service Management*. PhD thesis, School of Computer Science. Carnegie Mellon University., 1999.

[140] J. Lehoczky. Fixed priority scheduling of periodic task sets with arbitrary deadlines. In *Proceedings of the 11th IEEE Real-Time Systems Symposium*, pages 201–209, 1990.

[141] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm:exact characterization and average case behaviour. In *Proceedings of the 10th IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[142] J. Leung and J. Withehead. On the complexity of fixed priority scheduling of periodic real–time tasks. *Performance Evaluation*, 2(4), 1982.

[143] Jin Liang, Xiaohui Gu, and Klara Nahrstedt. Self-Configuring Information Management for Large-Scale Service Overlays. In *Proc. of the 26th Annual IEEE Conference on Computer Communications IEEE INFOCOM 2007*, Anchorage, Alaska, USA, May 2007.

[144] G. Lipari and S.K. Baruah. Greedy reclamation of unused bandwidth in constant bandwidth servers. In *Proc. of the 12th Euromicro Conference on Real–Time Systems*, pages 192–200, Stokholm, Sweden, 2000.

[145] M. Litoiu, T. C. Ionescu, and J. Labarta. Dynamic Task Schduling in Distributed Real–Time Systems using Fuzzy Rules. *Microprocessors and Microsystems*, 21:299–311, 1998.

[146] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard–real–time environment. *J. ACM*, 20(1):46–61, 1973.

[147] C. Locke. Software architecture for hard real–time applications: Cyclic executives versus fixed priority executives. *Real–Time Systems*, 4(1):37–53, 1992.

[148] N. Malcom and W. Zhao. Hard real-time communication in multiple-access networks. In Kluwer Academic Publishers, editor, *Real Time Systems*, volume 9, pages 75–107, Boston, USA, 1995.

[149] Ricardo Marau, Luis Almeida, and Paulo Pedreiras. Enhancing Real–Time Communication over COTS Ethernet switches. In *WFCS 2006, IEEE 6th Workshop on Factory Communication Systems*, Turin, Italy, June 2006.

[150] Ricardo Marau, Paulo Pedreiras, and Luis Almeida. Asynchronous Traffic Signaling over Master–Slave Switched Ethernet protocols. In *RTN 2007*, 2007.

[151] Clifford W. Mercer, Stefan Savage, and Hideyuki Tokuda. Processor capacity reserves: Operating system support for multimedia applications. In *Proc. of the International Conference on Multimedia Computing and Systems*, pages 90–99, 1994.

[152] Microsoft. *COM: The Component Object Model Specification*, February 2001. http://www.microsoft.com/com/resources/comdocs.asp.

[153] A. Mok. *Fundamental Design Problems of Distributed Systems for the Hard Real-Time Environment.* PhD thesis, Massachusetss Institute of Technology, 1983.

[154] A. Mok and M. Dertouzos. Multiprocessor Scheduling in a Hard Real–Time Environment. In *Proceedings of the 7th Texas Conference on Computer Systems*, 1978.

[155] M. Molle and L. Kleinrock. Virtual time csma: Why two clocks are better than one. *IEEE Transactions on Communications*, 33(9):919–933, 1985.

[156] Sape Mullender, editor. *Distributed systems*. Addison Wesley, 2nd edition, 1993. Chapters 7 and 8.

[157] K. Nahrstedt, D. Xu, D. Wichadakul, and B. Li. QoS–Aware Middleware for Ubiquitous and Heterogeneous Environments. *IEEE Communications Magazine*, 39(2):140–148, Nov. 2001.

[158] Klara Nahrstedt and Jonathan M. Smith. Design, Implementation, and Experiences of the OMEGA End-Point Architecture. *IEEE Journal on Selected Areas in Communications*, 14(7):1263–1279, 1996.

[159] J. Nieh and M. S. Lam. Integrated processor scheduling for multimedia. In *Proc. of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, 1995.

[160] Jason Nieh and Monica S. Lam. The design, implementation and evaluation of smart: a scheduler for multimedia applications. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 184–197, New York, NY, USA, 1997. ACM Press.

[161] Bill Nitzberg and Virginia Mary Lo. Distributed Shared Memory: A Survey of Issues and Algorithms. *IEEE Computer*, 24(8):52–60, 1991.

[162] Object Management Group (OMG). *UML 2.0 Infrastructure Specification Specification*, Julio 2005.

[163] Object Management Group (OMG). *UML 2.0 Superstructure Specification Specification*, Julio 2005.

[164] Object Management Group (OMG). *UML Profile for Schedulability, Performance, and Time*, Enero 2005.

[165] Object Management Group (OMG). *UML Profile for CORBA and CORBA Components Final Adopted Specification*, Octubre 2006.

[166] Object Management Group (OMG). *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and mechanisms*, Mayo 2006.

[167] ODVA - Open DeviceNet Vendor Association, Inc. *DeviceNet Specification - release 2.0, Vol. I and II*. USA, 1997.

[168] OMG. Common Object Request Broker Architecture (CORBA/IIOP). CORBA v.3.1, 2004.

[169] OMG. Real Time Corba Specification Version 1.2. formal/05-01-04, 2005.

[170] OMG. Corba Component Model. CCM v.4.0, 2006.

[171] Maximilian Ott, Georg Michelitsch, Daniel Reininger, and Girish Welling. An architecture for adaptive QoS and its application to multimedia systems design. *Computer Communications*, 21(4):334–349, 1998.

[172] Jose Carlos Palencia. *Análisis de planificabilidad de sistemas distribuidos de tiempo real*. PhD thesis, Grupo de Computadores y Tiempo Real. Universidad de Cantabria, 1999.

[173] Jose Carlos Palencia-Gutierrez and Michael Gonzalez-Harbour. Schedulability Analysis for Tasks with Static and Dynamic Offsets. In *Real–Time Systems Symposium, 1998.*, pages 26–37, December 1998.

[174] Jose Carlos Palencia-Gutierrez and Michael Gonzalez-Harbour. Exploiting Precedence Relations in the Schedulability Analysis of Distributed Real–Time Systems. In *Proc. of the 20th IEEE Real-Time Systems Symposium*, pages 328–339, December 1999.

[175] Gerardo Pardo-Castellote and Stan Schneider. The Network Data Delivery Service: Real–Time Data Connectivity for distributed control applications. In *ICRA*, pages 2870–2876. IEEE, 1994.

[176] Paulo Pedreiras. *Supporting Flexible Real-Time Communication on Distributed Systems*. PhD thesis, Departamento de Electrónica e Telecomunicações of the University of Aveiro, Aveiro, Portugal, 2003.

[177] Paulo Pedreiras and Luis Almeida. The Flexible Time–Triggered (FTT) Paradigm: An Approach to QoS Management in Distributed Real–Time Systems. In *IPDPS '03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 123.1, Washington, DC, USA, 2003. IEEE Computer Society.

[178] Paulo Pedreiras and Luis Almeida. Approaches to Enforce Real–time Behavior in Ethernet. In Richard Zurawski, editor, *The Industrial Communication Technology Handbook*. CRC Press/Taylor Francis, 2005.

[179] Paulo Pedreiras, P. Gai, Luís Almeida, and G.C. Buttazzo. FTT–ethernet: a flexible real-time communication protocol that supports dynamic QoS management on Ethernet–based systems. *IEEE Transactions on Industrial Informatics*, 1(3):162–172, August 2005.

[180] M. Peller, J. Berwanger, and R. Griesbach. Byteflight specification, version 0.5. http://www.byteflight.com, October 1999.

[181] Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for web service composition. In *In 11th World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 2002.

[182] D. Prasad, A. Burns, and M. Atkin. The Measurement and Usage of Utility in Adaptive Real–Time Systems. *Journal of Real–Time Systems*, 25(2/3):277–296, 2003.

[183] P. Puschner and A. Schedl. Computing Maximum Task Execution Times – A Graph–Based Approach. *Real–Time Systems*, 13(1):67–91, 1997.

[184] R. Rajkumar, Chen Lee, J.P. Lehoczky, and Daniel P. Siewiorek. A QoS-based resource allocation model. In *Proc. of the IEEE Real-Time Systems Symposium. RTSS*, 1997.

[185] R. Rajkumar, Chen Lee, J.P. Lehoczky, and Daniel P. Siewiorek. Practical solutions for QoS-based resource allocation problems. In *Proc. of the IEEE Real-Time Systems Symposium. RTSS*, 1998.

[186] R. Rajkumar, L. Sha, and J. P. Lehoczky. Real–time synchronization protocols for multiprocessors. In *Proc. of the 9th IEEE Real–Time Systems Symposium*, pages 259–269, 1988.

[187] K. Ramamritham and John A. Stankovic. Scheduling Strategies Adopted in Spring: an overview. In A. van Tilborg and G. Koob, editors, *Foundations of Real–Time Computing Scheduling and Resource Management*. Kluwer Academic Publishers, Boston, 1991.

[188] Bhaskaran Raman, Sharad Agarwal, Yan Chen, Matthew Caesar, Weidong Cui, Per Johansson, Kevin Lai, Tal Lavian, Sridhar Machiraju, Zhuoqing Morley Mao, George Porter, Timothy Roscoe, Mukund Seshadri, Jimmy S. Shih, Keith Sklower, Lakshminarayanan Subramanian, Takashi Suzuki, Shelley Zhuang, Anthony D. Joseph, Randy H. Katz, and Ion Stoica. The SAHARA Model for Service Composition across Multiple Providers. In *Pervasive '02: Proceedings of the First International Conference on Pervasive Computing*, pages 1–14, London, UK, 2002. Springer-Verlag.

[189] Jorge Real and Alfons Crespo. Mode Change Protocols for Real–Time Systems: A Survey and a New Proposal. *Real Time Systems Journal*, 26(2):161–197, March 2004.

[190] Luciana Rech, Rômulo Silva de Oliveira, and Carlos Montez. Dynamic determination of the itinerary of mobile agents with timing constraints. In Andrzej

Skowron, Jean-Paul A. Barthès, Lakhmi C. Jain, Ron Sun, Pierre Morizet-Mahoudeaux, Jiming Liu, and Ning Zhong, editors, *IAT*, pages 45–50. IEEE Computer Society, 2005.

[191] Sangig Rho. *A Distributed Hard Real-Time Java for High Mobility Components*. PhD thesis, Texas, December 2004.

[192] I. Ripoll. *Planificación con Prioridades Dinámicas en Sistemas de Tiempo Real Crítico*. PhD thesis, Valencia, 1996.

[193] Robert Bosch GmbH, Stuttgart, Germany. *CAN Specification version 2.0*, 1991.

[194] Jose Francisco Ruiz. *A hierarchical and Decentralized QoS and Resource Management Architecture for Embedded Systems*. PhD thesis, Universidad Politécnica de Madrid, 2004.

[195] John Rushby. A Comparison of Bus Architectures for Safety-Critical Embedded Systems. Technical report, Langley Research Center, National Aeronautics and Space Administration, NASA, Hampton, Virginia 23681-2199, March 2003. NASA/CR-2003-212161.

[196] J.H. Saltzer, D.P. Reed, and D.D. Clark. End–to–end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, 1984.

[197] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[198] R. Schantz, J. Loyalll, M. Atighetchi, and P. Pal. Packaging quality of service control behaviors for reuse. In *ISORC '02: Proceedings of the Fifth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, page 375, Washington, DC, USA, 2002. IEEE Computer Society.

[199] L. Sha, R. Rajhumar, and J.P. Lehoczky. Priority Inheritance Protocols: An Approach to Real–Time Synchronization. *IEEE Transaction on Computers*, 39(9):1175–1185, Septiembre 1990.

[200] Lui Sha, Tarek Abdelzaher, Karl-Erik Arzen, Anton Cervin, Theodore Baker, Alan Burns, Giorgio Buttazzo, Marco Caccamo, John Lehoczky, and Aloysious K. Mok. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems Journal*, 28(2/3):101–155, 2004.

[201] Lui Sha and John B. Goodenough. Real–time scheduling theory and ada. *IEEE Computer*, 23(4):53–62, 1990.

[202] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Computers*, 39(9):1175–1185, 1990.

[203] Kang G. Shin and P. Ramanathan. Real–time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1), 1994.

[204] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic Task Scheduling for Hard–Real-Time Systems. *Real–Time Systems*, 1:27–60, 1989.

[205] Marco Spuri and Giorgio C. Buttazzo. Efficient aperiodic service under the earliest deadline scheduling. In *Proc. of the IEEE Real–Time Systems Symposium*, 1994.

[206] Marco Spuri and Giorgio C. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real–Time Systems*, 10(2):179–210, 1996.

[207] J.A. Stankovic, K. Ramamritham, M. Spuri, and G. Buttazzo. *Deadline scheduling for real–time systems*. Kluwer, Boston–Dordrecht–London, 1998.

[208] John A. Stankovic. Misconceptions about real–time computing: A serious problem for next generation systems. *IEEE Computer*, 21(10):10–19, 1988.

[209] John A. Stankovic et al. Implications of Classical Scheduling Results for Real–Time Systems. *IEEE Computer*, 28(6), 1995.

[210] Sun. Java remote method invocation. RMI v.1.5, 2004. Available on-line at http://java.sun.com/j2se/1.5/pdf/rmi-spec-1.5.0.pdf.

[211] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. In *Proc. of the 16th International Conference on Distributed Systems*, Mayo 1996.

[212] Sun Microsystems. *Jini Specification, version 2.0*. Available on http://www.sun.com.

[213] Sun Microsystems. Enterprise Java Beans. EJB v.2.1, 2005.

[214] C. Szyperski. *Component Based Software - Beyond Object-Oriented Programming*. Addison–Wesley, 1999.

[215] Ming-Chit Tam, Jonathan M. Smith, and David J. Farber. A taxonomy–based comparison of several distributed shared memory systems. *SIGOPS Oper. Syst. Rev.*, 24(3):40–67, July 1990.

[216] A. Tešanović. Towards Aspectual Component–Based Real–Time System Development. Master's thesis, Linköping University, Junio 2003.

[217] A. Tešanović. *Developing Reusable and Reconfigurable Real–Time Software Using Aspects and Components*. PhD thesis, Linköping University, 2006.

[218] Aleksandra Tešanović, Mehdi Amirijoo, Mikael Björk, and Jörgen Hansson. Empowering Configurable QoS Management in Real-Time Systems. In *Proceedings of the 4th ACM SIG International Conference on Aspect-Oriented Software Development (AOSD'05)*, pages 39–50. ACM Press, March 2005.

[219] Aleksandra Tešanović, Dag Nyström, Jörgen Hansson, and Christer Norström. Aspects and Components in Real–Time System Development: Towards Reconfigurable and Reusable Software. *Journal of Embedded Computing*, February 2004.

[220] J.-P. Thomesse. Time and industrial local area networks. In *Proceedings of COMPEURO'93*, Paris, France, 1993.

[221] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 50(2–3), 1994.

[222] Ken Tindell. An extendible approach for analysis fixed priority hard real-time tasks. *Real–Time Systems Journal*, 6(2), Marzo 1993.

[223] Ken Tindell. Adding Time–Offsets to Schedulability Analysis. Technical Report YCS221, Department of Computer Science, University of York, England, 1994.

[224] Ken Tindell, Alan Burns, and Andy Wellings. Allocating hard real–time tasks (an NP–hard problem made easy). *Real–Time Systems*, 4(2):145–165, 1992.

[225] Keneth William Tindell. *Fixed Priority Scheduling of Hard Real–Time Systems*. PhD thesis, University of York, 1993.

[226] Ruth Tolosa, José P. Mayo, Miguel A. de Miguel, M. Teresa Higuera-Toledano, and Alejandro Alonso. Container model based on rtsj services. In Robert Meersman and Zahir Tari, editors, *OTM Workshops*, volume 2889 of *Lecture Notes in Computer Science*, pages 385–396. Springer, 2003.

[227] E. Tovar and F. Vasques. Real-time fieldbus communications using profibus networks. *IEEE Transactions on Industrial Electronics*, 46(6):1241–1251, December 1999.

[228] W. T. Tsai, Yann-Hang Lee, Zhibin Cao, Yinong Chen, and Bingnan Xiao. RTSOA: Real-Time Service-Oriented Architecture. In *SOSE '06: Proceedings of the Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*, pages 49–56, Washington, DC, USA, October 2006. IEEE Computer Society.

[229] TTTech Computertechnik AG. *Specification of the TTP/C protocol*. Available at http://www.tttech.com.

[230] Paulo Veríssimo and Luís Rodrigues. *Distributed systems for system architects*. Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[231] Nanbor Wang. *Composing Systemic Aspects Into Component-Oriented DOC Middleware*. PhD thesis, Washington University, St. Louis, MO 63130, May 2004. Available at: http://www.zen.uci.edu/publications/.

[232] Nanbor Wang, Christopher D. Gill, Douglas C. Schmidt, Aniruddha Gokhale, Balchandran Natarajan, Joseph P. Loyall, Richard E. Schantz, and Craig Rodrigues. *Middleware for Communications*, chapter QoS-enabled Middleware. John Wiley & Sons, Ltd, 2001.

[233] Shengquan Wang, Sangig Rho, Zhibin Mai, Riccardo Bettati, and Wei Zhao. Real-Time Component-Based Systems. In *Proc. of the 11th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 428–437, 2005.

[234] Z. Wang and J. Crowcroft. Quality-of-service routing for supporting multimedia applications. *IEEE Journal on Selected Areas in Communications*, 14:1228–1234, 1996.

[235] Christopher Ward, Melissa J. Buco, Rong N. Chang, and Laura Z. Luan. A generic sla semantic model for the execution management of e-business outsourcing contracts. In *EC-WEB '02: Proceedings of the Third International Conference on E-Commerce and Web Technologies*, pages 363–376, London, UK, 2002. Springer-Verlag.

[236] Mark Weiser. The Computer for the 21st Century. *Scientific American*, September 1991.

[237] Andy J. Wellings, Roy Clark, E. Douglas Jensen, and Douglas Wells. The distributed real-time specification for java: A status report. In *Embedded Systems Conference*, pages 13–22, 2002.

[238] Andy J. Wellings, Roy Clark, E. Douglas Jensen, and Douglas Wells. A framework for integrating the real-time specification for java and java's remote method invocation. In *Symposium on Object-Oriented Real-Time Distributed Computing*, pages 13–22, 2002.

[239] D. Wichadakul, X. Gu, and K. Nahrstedt. A Programming Framework for Quality-Aware Ubiquitous Multimedia Applications. In *Proc. of ACM Multimedia 2000*, Juan Les Pins, France, December 2002.

[240] World Wide Web Consortium. *Web Services Glossary*, February 2004. W3C Working Group Note.

[241] Wen-Lin Yang. Optimal and heuristic algorithms for quality-of-service routing with multiple constraints. *Perform. Eval.*, 57(3):261–278, 2004.

[242] Rajendra Yavatkar and K. Lakshman. A CPU Scheduling Algorithm for Continuous Media Applications. In *NOSSDAV '95: Proceedings of the 5th International Workshop on Network and Operating System Support for Digital Audio and Video*, pages 210–213, London, UK, 1995. Springer-Verlag.

[243] X. Yuan. Heuristic algorithms for multiconstrained quality-of-service routing. *IEEE/ACM Trans. Network.*, 10(2):244–256, 2002.

[244] J. Zamorano-Flores. *Planificación estática de procesos en sistemas de tiempo real críticos*. PhD thesis, Universidad Politécnica de Madrid, 1995.

[245] John A. Zinky, David E. Bakken, and Richard E. Schantz. Architectural Support for Quality of Service for CORBA Objects. *Theory and Practice of Object Systems*, 3(1), 1997.