Universidad Carlos III de Madrid

Escuela Politécnica Superior

Ingeniería de Telecomunicación

PROYECTO FIN DE CARRERA

Implementación y evaluación de algoritmos de composición de aplicaciones distribuidas de tiempo real basadas en servicios

Autor: Jorge Diez Sánchez

Tutora: Dra. Iria Manuela Estévez Ayres

Marzo 2010

A mis padres

y a mi hermano

A Mónica

Comprender es el principio de aprobar.

Baruch Benedict Spinoza

Cuanto más entreno, más suerte tengo.

Gary Player

La imaginación es más importante que el saber.

Albert Einstein

Lo último que uno sabe es por donde empezar.

Blaise Pascal

Solo sé que no sé nada.

Socrates

Agradecimientos

Con el presente proyecto fin de carrera termino una de las etapas más enriquecedoras de mi vida y cuya finalización me llena de orgullo y satisfacción. Con esto concluyo un sueño que se inició hace muchos años cuando en los partidos del Baloncesto León, observaba con fascinación la "magia" con que las antenas de las unidades móviles hacían que un partido se pudiese estar viendo en la televisión. A partir de esos recuerdos de un niño, así como un gusto innato por las ciencias y nuevas tecnologías hicieron que me embarcara en este barco que ahora llega a puerto.

Dentro de este viaje, me gustaría agradecer la compañía de dos personas sin las cuales esto no hubiese sido posible: mis Padres. Gracias por darme la posibilidad de estudiar esta carrera. Gracias por apoyarme en cada uno de los duros momentos que he tenido que superar para llegar a donde ahora me encuentro. Gracias por "haber vuelto" de nuevo a la Universidad para hacer una nueva carrera junto a vuestro hijo y gracias por no haber desfallecido en las tempestades que nos hemos ido encontrando. ¡¡Simplemente gracias!! Espero algún día poder devolveros la mitad del apoyo que me habéis prestado.

Asimismo, me gustaría resaltar la compañía, en ese mismo barco, de mi hermano. Daniel, gracias por haber sido la persona más cercana, por haberme dirigido cuando estaba perdido y por haberme apoyado, tanto técnica como humanamente, en todo el viaje. Junto a ti las cosas han sido mucho más fáciles y tu ayuda incondicional ha sido motor para conseguir llegar hasta aquí.

Por supuesto tengo que destacar la contribución que ha tenido en mi persona la estancia en la Residencia Fernando Abril Martorell (*FAM*), considerándolo posiblemente lo mejor de mi estancia en esta etapa, ya que me ha formado como persona y me ha permitido conocer grandes personas. Gracias Tin, Duarte, Toufik, Fer por vuestra gran ayuda, compañía y lo que considero más importante, vuestra amistad. Gracias Rodri, Eli, Juan Carlos, Aida, Charly ... porque estos años universitarios no hubieran sido lo mismo sin vosotros. También he de mostrarme agradecido con mis amigos de León por su confianza en mi, y su apoyo en la distancia, gracias Javi, Pablo, Silvia, Inés, Bego, Kike, Miguel, Salado ... Y por supuesto gracias a la gente tan maravillosa que conocí en mi estancia en la DTU (CPH). Sin la ayuda

prestada durante ese año habría sido imposible terminar. Va por vosotros Giacomo, Tere, Santi, Nadia, Fabri, Hristo, Anna, Andrés, Guille, Jaime, etc.

Finalmente me gustaría agradecer a mi familia, en especial a mi Abuela. Por supuesto tengo que nombrar a mi tío Edu por su preocupación, su disponibilidad y las facilidades para que mis veranos fuesen mucho más amenos. Y quisiera tener un recuerdo especial con los que ya no están aquí para verlo, pero han contribuido en que consiga esta meta. ¡Muchas gracias!

No podia olvidarme de mi tutora Iria, por haberme ayudado a concluir de una forma amena y enriquecedora esta etapa de mi vida.

Para concluir y no por ello menos importante, sino todo lo contario, me gustaría dar un agradecimiento especial a Mónica. Porque gracias a su paciencia, ilusión, sabiduría, interés y especialmente su cariño y apoyo, finalmente podré decir que seré Ingeniero, lo cual sabe que es una de las mayores ilusiones de mi vida. Gracias por ser así, y por saber llevarme y apoyarme, lo cual reconozco que no es fácil. ¡Un trocito de esto te pertenece!

En definitiva, quiero agradecer a todas las personas que de una u otra manera han compartido conmigo estos años universitarios.

Gracias a todos.

Resumen

En el mundo actual en el que vivimos, se han extendido los entornos que trabajan con sistemas que garantizan el cumplimiento de tiempos de respuesta. Hasta ahora, al trabajar con entornos dinámicos, se aplicaba una aproximación clásica de diseño que nos aseguraba unas garantías temporales pero que podía implicar un gasto de recursos prohibitivo. Es por esto que se este dedicando un gran esfuerzo investigador para dotar de flexibilidad y dinamismo a sistemas de tiempo real.

Este Proyecto Fin de Carrera estudia aquellos trabajos que buscan la aplicación de técnicas del mundo de la orientación a servicios a dichos sistemas. En concreto, algoritmos de composición de aplicaciones de tiempo real basados en servicios, centrándonos en dos algoritmos apropiados para dicha composición.

Estos algoritmos se implementaron en el lenguaje de programación *Matlab* y su estudio condujo a la propuesta de mejora de los mismos. Finalmente, se desarrolló una herramienta en *Matlab* que permitía la simulación y validación de los distintos algoritmos.

Abstract

In the current world we are living in, there has been an increase in the amount of application domains where the development of systems with temporal guarantees is required.

Until now, when dynamic domains were used, a classic design approximation was implemented, which ensured us such temporal guarantees, but could entail a huge amount of resources expenses. It was this problem that led to a huge research effort aiming to give flexibility and dynamism to real time systems.

This Master Thesis studies those researches which propose the use of service oriented techniques in this kind of real time systems. Being more specific, algorithms related to composition of real time applications based on services will be used, paying special attention specifically to two algorithms that are suitable for this scenario.

These algorithms are implemented using Matlab's programming language. Their study has aided to propose an improvement of these algorithms. Finally, a Matlab application was developed for simulating and validating all the different algorithms.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

El objetivo de este proyecto fin de carrera consiste en el diseño, implementación y evaluación de un conjunto de algoritmos desarrollados en Matlab para la composición de aplicaciones de tiempo real basadas en servicios. En este primer capítulo se pretende describir de forma general el contexto en cual se encuadra el proyecto, así como los objetivos que se han planteado.

1.1. Introducción

Para poder describir de forma adecuada el contexto sobre el que desarrolla este proyecto, en primer lugar se han de definir los sistemas con los que se va a trabajar a lo largo del presente documento. Un sistema de tiempo real es aquél en el que para que las operaciones computacionales sean correctas no depende sólo de una correcta lógica e implementación de los programas, sino también del tiempo en el que dicha operación entregue su resultado [2][3]. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado [1]. Por lo tanto resulta esencial en estos sistemas que se cumplan las restricciones de tiempo. El tiempo en el que el proceso ha de completar su ejecución para evitar que el sistema sufra algún daño se denomina plazo (*deadline*). Asimismo, una de las consecuencias de tener que garantizar el comportamiento en el tiempo requerido es que hace que el sistema sea predecible.

Hoy en día la mayoría de los sistemas de tiempo real trabajan sobre entornos dinámicos en los cuales son poco predecibles los cálculos de las cargas computacionales. Para el diseño de este tipo de sistemas es necesario, tanto el dotarlos de una flexibilidad, como el buscar un dinamismo con predictibilidad temporal, debido a que estos sistemas no se ven gobernados por aproximaciones clásicas basadas en el cálculo de peores tiempos de ejecución y la asignación de recursos en el peor caso. Las aproximaciones clásicas nos aseguran un determinismo temporal, pero ante grandes aplicaciones nos llevan a un gasto de recursos prohibitivo.

A partir de estos principios, la comunidad de tiempo real está usando la mayor parte de sus esfuerzos en la adaptación a entornos dinámicos [4], tanto en la investigación de nuevos algoritmos para la gestión de los recursos, la iteración de las tareas y mensajes, como en el desarrollo de protocolos y arquitecturas que nos permitan la flexibilidad de los sistemas.

La gran evolución de los servicios requeridos en Internet, lo ha hecho llegar a ser un proveedor de aplicaciones, las cuales necesitan unas calidades de servicio (QoS). Esto ha llevado a la creación de contratos (SLA, *Service Level Agreement contract*), en los que usuario y proveedor establecen los niveles de los servicios, los cuales serán adecuados a las necesidades de cada aplicación. Asimismo el hecho de usar Internet como proveedor de aplicaciones, ha obligado a que los servicios dejen der ser centralizados para pasar a ser distribuidos en los distintos proveedores. Asimismo el hecho de la mejora de las conexiones de red y la evolución de los procesadores de los ordenadores han permitido la ejecución de sistemas distribuidos, los cuales ejecutan remotamente aplicaciones en otros ordenadores conectados en red [5]. También han tenido gran importancia en estos avances la aparición de nuevos paradigmas de distribución de los procesos a través de las redes, así como paradigmas de programación como el de Cliente-Servidor, el cual está implementado sobre un amplio número de sistemas. Por lo tanto, podemos comprobar la evolución tanto de los sistemas como de los servicios desde una configuración centralizada hacia una distribuida, que va en concordancia con la evolución de la tecnología.

A la hora de definir un servicio nos apoyamos en la definición dada por Steves Jones en *"Toward an acceptable definition of service"* [6]: *es un dominio de control de una envergadura acotada que contiene un conjunto de tareas que cooperan para alcanzar objetivos relacionados* . Por lo tanto, cuando hablemos en el presente trabajo de servicios nos esta-

remos refiriendo a una entidad *software* autocontenida que proporciona una determinada funcionalidad.

Respecto a los dispositivos *hardware* que los contienen, día a día las personas convivimos con una mayor cantidad de aparatos que tienen capacidad de cómputo y comunicación. Estos dispositivos ya forman parte de nuestra rutina e interactuamos con ellos sin pensar que estos dispositivos nos ofrecen mayor número de funciones de las que en un principio fueron dotadas [7]: los reproductores de música por ejemplo, no solo ofrecen la posibilidad de escuchar canciones, sino que además tienen la posibilidad de reproducir archivos de video, fotografías, grabar voz, lectura de $e-books$, juegos interactivos, conexión a internet, etc. Este hecho, que hace que el usuario no sea consciente de convivir con elementos heterogéneos, hizo evolucionar el paradigma de la computación distribuida, la cual ahora pretende integrar los dispositivos en el entorno (difuminándose y 'desapareciendo' en él [8]) pero teniendo presente que han de satisfacer las necesidades del usuario, atendiendo no sólo a la funcionalidad y características del dispositivo, sino también al entorno y situación personal del usuario [1].

1.2. Nuevos paradigmas

Teniendo en cuanta lo descrito en el párrafo anterior, los nuevos paradigmas de coputación, [9][10], aparecerán orientado a servicios (*Service Oriented Computing*, SOC)[11]. Su modelo estará basado en una computación distribuida, que se fundamenta en la existencia de proveedores y consumidores de servicios. Para favorecer el uso de estos servicios se hace necesarios unos registros que favorezcan su publicación así como la facilidad en la búsqueda de los mismos por parte de los consumidores. Asimismo se plantea necesario el uso de composición de servicios, como elemento para la creación de nuevos servicios y aplicaciones a partir de servicios ya existentes. Esto nos permitirá la existencia de entidades que pueden actuar como productoras de servicios, consumidoras de los mismos, o productoras/consumidoras, que surgen al hacer uso de la composición de servicios ya existentes para proveer uno más complejo. Esta nueva forma de creación de servicios favorecerá tanto la disminución del tiempo de desarrollo de un proyecto, así como la facilidad en la flexibilidad de los servicios seleccionados para su creación, pudiendo cambiar de proveedor

en el caso que las necesidades del proyecto cambien. Asimismo esta flexibilidad puede ser utilizada para proporcionar tolerancia a fallos en la aplicación, cambiando servicios que dejen de funcionar por otros que desarrollen la misma funcionalidad [11].

1.3. Sistemas de tiempo real

A día de hoy la computación orientada a servicios para sistemas de tiempo real se encuentra en una fase preliminar. Esto es debido a la no existencia, por el momento, de ninguna arquitectura completa que sea capaz de dar soporte en tiempo real al paradigma presentado en el apartado anterior. Así las cosas, la única manera que se ha utilizado para poder dotar a los sistemas de tiempo real de una cierta flexibilidad ha sido la utilización de tecnología basada en componentes [12][13], tanto a nivel de modelado, [14][15], como a nivel de lenguajes [16] y plataformas [17]. Por lo tanto puede decirse que la mayoría de los sistemas de tiempo real con los que se trabaja en la actualidad son de planificación estática, es decir, están formados por un conjunto de servicios fijos, seleccionados con antelación, los cuales son asignados a los recursos disponibles. Para asegurar que todas las restricciones temporales son satisfechas, se realizan comprobaciones previas al tiempo de ejecución.

Sin embargo los sistemas de tiempo real a los cuales se quiere aspirar en un futuro, necesitarán una mayor flexibilidad durante su ejecución [4][18], pudiendo modificar el sistema para ajustarse dinámicamente a la información que procederá tanto del sistema como del entorno donde se está ejecutando. Esto deberá llevar al sistema a la modificación de los servicios que está soportando para satisfacer en todo momento las funciones a las cuales da servicio. Esta flexibilidad que se persigue puede ser aportada, como se comentó en el apartado anterior, por el paradigma de computación orientada a servicios, ya que puede permitir la creación dinámica de aplicaciones a partir de servicios remotos. La posibilidad que se nos abre a la hora de componer dinámicamente servicios, no solo es interesante para el desarrollo de aplicaciones 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 [1]:

- Actualización dinámica: las nuevas versiones de los servicios pueden ser usadas si estas mejoran el rendimiento de las aplicaciones en las que está siendo usado.

- Tolerancia a fallos: los diferentes perfiles de un servicio serán usados para asegurar la supervivencia del sistema si uno de los nodos físicos sobre los que se está ejecutando se viese afectado por un fallo. Si se detectase se procedería a sustituir los perfiles de los servicios por unos que fuesen ejecutables.

- Equilibrado de carga: el sistema intentará el equilibrio entre los distintos nodos físicos del sistema para evitar deterioros en la prestación de los servicios ejecutados en cada uno de ellos y por lo tanto una disminución de las calidades ofrecidas por las aplicaciones.

Para representar lo anterior podemos definir un ejemplo práctico basado en cámaras de vigilancia. Se tendrán una serie de cámaras de vigilancia que enviarán información a diferentes centros de control. Debido a que la mayoría del tiempo no habrá incidencias, se podría tener en espera los flujos de datos desde las cámaras hacia los centros de control. En el momento que una cámara detecte alguna anomalía (movimiento, ruido, humo, etc.), reactivará su flujo de datos. Asimismo el centro de control podrá pedir información de más cámaras para evaluar el riesgo, o la gravedad de la alarma, así como para informarse de la situación en caso de que la cámara principal se quede sin servicio. 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 centros de control para mejorar las prestaciones del conjunto del sistema.

Finalmente otro de los campos donde se han realizado gran cantidad de estudios, es el de los servicios Web, donde se pretende extender el protocolo utilizado en éstos para intercambiar datos, SOAP [19][20], de tal forma que pueda soportar tiempo real. Al igual que los estudios anteriores, aun no se ha podido llegar a ninguna especificación [21].

Por todo lo expresado anteriormente, queda patente la necesidad de creación de una arquitectura de tiempo real, complementada con la definición de metodologías que puedan ser aplicadas sobre el paradigma estudiado. Teniendo en cuenta todo lo anterior se realizó la tesis de Iria Estévez Ayres [1], "que surgió bajo la idea de 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". Para conseguirlo se propuso la

aplicación de los conceptos de SOA al campo de los sistemas de tiempo real. Fue necesaria la definición de un modelo de aplicaciones basadas en servicios y de una arquitectura genérica que proporcionase soporte, así como la creación de los protocolos necesarios para ello. Siguiendo lo anterior, se consiguió integrar dicha arquitectura con mecanismos y protocolos de tiempo real ya existentes. Para todo ello se trabajó con sistemas de tiempo real no críticos.

1.4. Objetivos principales

Apoyándonos en los estudios realizados en la tesis de Iria Estévez Ayres, [1], los objetivos específicos del proyecto fin de carrera son los siguientes:

- Estudio de las principales características de los sistemas distribuidos de tiempo real y sus arquitecturas. También se cree necesario un estudio de las características de los entornos distribuidos.
- Estudio de los algoritmos de composición adecuados para aplicaciones de tiempo real basadas en servicios, los cuales se proponen en [1].
- Mejora y desarrollo de nuevos algoritmos de composición, que se basen en los estudios realizados en el punto anterior. Se buscará la creación de nuevos algoritmos que favorezcan la búsqueda de soluciones a la composición en un tiempo acotado. Estos, trabajarán sobre los casos en que los algoritmos de [1] se mostraban más desfavorables.
- Validación del comportamiento de los algoritmos propuestos:
  - Simulación de los algoritmos implementados para las distintas configuraciones.
  - Obtención y validación de las cotas temporales halladas teóricamente.
  - Estudio de las aplicaciones para las cuales resulta más idóneo cada algoritmo.
- Implementación de una herramienta de simulación que nos permita interactuar con los algoritmos implementados a lo largo del proyecto, facilitando la ejecución de los mismos, así como la realización de las pruebas necesarias para la presentación de resultados.

Capítulo 2

Estado del Arte

Este capítulo es un estudio sobre las tecnologías más importantes relacionadas con los Sistemas de Tiempo Real. Se estudiarán tanto las tecnologías sobre sistemas de tiempo real centralizados, así como los distribuidos. Se estudiará con especial interés la planificación de las tareas en ambos algoritmos, al ser la parte fundamental en la que se basa el proyecto. Asimismo se introducirán las redes a utilizar a la hora de trabajar con sistemas distribuidos. Finalmente se presentará el paradigma de la orientación a servicios.

2.1. Sistemas de tiempo real

Como ya definimos anteriormente el sistema de tiempo real es aquel en el que para que las operaciones computacionales sean correctas no depende solo de una correcta lógica e implementación de los programas, sino también del tiempo en el que dicha operación entregue su resultado [22][2]. Asimismo destacamos que el plazo es el tiempo que el proceso tiene para completar su ejecución y que el hecho de garantizar los plazos hace de estos sistemas predecibles. Hemos de añadir a estas características la importancia que estos sistemas obtengan un alto grado de utilización a la vez que cumple con los requerimientos de tiempo.

Dentro de los sistemas de tiempo real podemos distinguir la clásica concepción llamada tiempo real duro (*hard real time*) o sistemas de tiempo real inmediatos en los que es de vital importancia la conclusión de una operación antes de su *deadline*, ya que una conclusión

tardía puede provocar una situación crítica en el sistema. Asimismo podemos encontrar sistemas de tiempo real suaves (*soft real time*) que toleran la posible entrega fuera de plazo, respondiendo con una degradación de la calidad de servicio del mismo.

Una vez superado el plazo, el resultado de una operación podrá ser clasificada [10][23] como firmes, flexibles o críticas. Si una respuesta fuera de tiempo no tiene ningún valor, el plazo será clasificado como firme, en caso contrario, si tiene utilidad, como flexible. En caso de que la superación de un plazo firme suponga consecuencias catastróficas para el sistema, éste se denominará crítico. El sistema de tiempo real que ejecuta por lo menos una tarea con un plazo crítico es un sistema de tiempo real crítico, sirva de ejemplo el laser de una operación de ojos. En caso contrario se conocerá como sistema de tiempo real flexible, como puede ser un sistema de procesamiento de datos multimedia.

2.1.1. Tareas de Tiempo Real

Hoy en día las operaciones ocurren en paralelo, por lo que cualquier sistema que interactúe con el mundo real ha de imitar ese comportamiento. Para ello se recurre a la abstracción la cual consiste en la construcción del sistema como un conjunto de operaciones secuenciales que cooperan e interactúan entre sí y con el entorno [24][25].

Otra de las clasificaciones que se pueden realizar teniendo en cuenta las características temporales es: operaciones periódicas, aquellas en las que su activación se produce con una periodicidad fija, como puede ser el muestreo a una determinada tasa; operaciones aperiódicas, aquellas en las que no existe una periodicidad fija, como puede ser la lectura de unos datos al realizarse un cambio de contexto. Dentro de este último grupo existirán un tipo de operaciones caracterizadas por tener períodos de activación variables, denominadas operaciones esporádicas. Para este determinado caso, el parámetro que las caracteriza será la tasa de llegada al sistema.

En un sistema de tiempo real, estas operaciones o procesos suelen ser llevadas a cabo por sistema software. Habitualmente, un sistema software ejecuta diferentes actividades, pudiendo cada una de ellas tener diferentes restricciones temporales. Dentro de estas tareas, algunas 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.

Por lo tanto, a la hora de implementar un sistema, hay que conocer las características y exigencias de cada una de las tareas. Un conjunto de N tareas periódicas Π se puede caracterizar como:

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

siendo:

- 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. Dicho periodo es el tiempo entre dos invocaciones sucesivas de la misma. Cada tarea periódica puede producir una secuencia infinita de invocaciones separadas T_i unidades de tiempo.
- ϕ_i es el desplazamiento de la tarea τ_i, con $\phi_i < T_i$. Dicho desplazamiento corresponde al momento de invocación de la tarea respecto a t = 0. 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. Antiguamente, se usaba como prioridad el subíndice de la tarea, pero esta notación deja de ser válida cuando se introdujeron nuevas técnicas en las que 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 caracterizar las tareas esporádicas que se han nombrado previamente se puede seguir utilizando la misma notación, teniendo en cuenta el período mínimo de activación, $T_{min,i}$, en lugar del período y dejando sin definir el desplazamiento temporal:

$$\sigma_i = (Ci, Di, T_min, i, pi), i = 1 \dots M \tag{2.2}$$

2.1.2. Planificación de sistemas de tiempo real

El principal objetivo de un sistema de tiempo real será la asignación de un correcto reparto de recursos entre las diferentes tareas del mismo, con el fin de asegurar que se

cumplan sus requisitos temporales. Para llevar a cabo esta tarea será insuficiente someter a diversas pruebas al sistema, ya que con estos solamente se puede demostrar la existencia de errores pero no la ausencia completa de los mismos. Para intentar probar que el sistema es correcto, surge la Teoría de Planificación [26] como la base teórica necesaria para corroborar que un sistema de tiempo real cumple con las condiciones impuestas.

Un *algoritmo de planificación* se definirá como un conjunto de reglas que determinan las decisiones que se han de tomar a lo largo de la vida del sistema de forma que todas las tareas cumplan sus requisitos temporales o de otro tipo, como pueden ser las restricciones de precedencia, usando el mínimo de recursos. El término planificación se referirá al proceso de seleccionar una tarea concreta para su ejecución en un determinado instante temporal.

La problemática principal de la planificación de estos sistemas se subdividirá en dos apartados: 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, se pasará al segundo apartado que será encontrar dicha planificación. Al primer problema se le denomina análisis de planificabilidad y al segundo construcción de la planificación.

Tal como se ha nombrado previamente, el análisis de planificabilidad para un determinado algoritmo se basará en la ejecución de diversas pruebas. 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. Por ello se clasificarán en tres grupos [27]:

- **Tests Exactos**. Son los más complejos computacionalmente hablando. Tendrán un resultado positivo siempre que exista una planificación posible para el sistema y negativo en caso contrario. Desafortunadamente, la mayor parte de este tipo de tests son computacionalmente intratables [28], o comportan una carga computacional excesiva que reduce su aplicación a las fases off-line del sistema.

- **Tests Suficientes**. Son más sencillos de implementar que los test exactos. Su resultado positivo asegura la existencia de una planificación posible, pero si es negativo no aseguran que no la haya. Cabe destacar que este tipo de test puede, por lo tanto, rechazar conjuntos de tareas planificables.

- **Tests Necesarios**. Estos test 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, existen principalmente dos familias de test aplicados a sistemas de tiempo real, basada cada una de ellas en conceptos diferentes [29]:

- **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:

$$U_i = \frac{C_i}{T_i} \tag{2.3}$$

- ***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 llevan a cabo previamente a la ejecución del sistema (también conocidos como algoritmos *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 estos algoritmos podemos distinguir entre aquellos en los que las prioridades de las tareas son determinadas *off-line*, es decir, estáticas y los que, dado un cierto criterio, pueden modificar las prioridades de forma dinámica.

Existe una nueva división de los algoritmos dinámicos relacionada con el uso del procesador dependiendo de la prioridad de las tareas. Se denominaran algoritmos dinámicos sin apropiación a aquellos en los que la decisión de quién entra en el procesador se pospone hasta la completa ejecución de las tareas. Sin embargo, si es posible que una tarea vea interrumpida su ejecución, debido a que otra de mayor prioridad pase a estar preparada, el algoritmo será con apropiación del procesador.

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

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

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

Tabla 2.1: Clasificación algoritmos planificación y ejemplos para planificación centralizada

del inicio o conclusión de las demás tareas del sistema, y no hay relaciones de precedencia o de exclusión. Debido a ello, las tareas no se pueden suspender en 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

Tal como se ha definido previamente, la planificación estática u $off-line$ se caracteriza porque todas las decisiones de planificación se toman antes de la ejecución del sistema. Dicha planificación se almacenará 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 lee de forma secuencial la lista y activa las tareas en los instantes apropiados.

El funcionamiento del planificador es muy sencillo, robusto y eficiente, ya que su ejecución casi no supone carga computacional para el sistema. Además como ya se ha indicado, este método es determinista y predecible [30], al saber con exactitud lo que el procesador

está ejecutando en cada momento. Sin embargo, como estudiaremos posteriormente esta aproximación produce sistemas inflexibles y difíciles de mantener [22].

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 [25]. Un problema NP-completo se puede definir como [31][32] [33][34]:

Definición 2.1. *Reingold et al, 1977 y Wilf 1986 Un problema se dice que es NP-Completo si es duro y es NP. Un problema NP-Completo tiene la característica de que todo problema en NP se reduce polinomialmente a él.*

por lo que ahora habrá que definir que es un problema NP y un problema *duro*

Definición 2.2. *Stinson, 1987 NP es acrónimo en inglés de Polinómico no determinista (Non-Deterministic Polynomial-time) y denota la colección de todos los problemas de decisión los cuales tienen algoritmos de solución no-determinísticos en tiempo polinomial*

Definición 2.3. *Reingold et al, 1977 Se dice que un problema es NP-Duro si todo problema en NP se puede transformar polinomialmente a él.*

Es decir, a la hora de resolver la planificación, 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 [3][35][30].

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. 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 [36] es la dificultad de incluir en la planificación tareas no periódicas, debido al carácter estático del plan de ejecución.

2.1.3.2. Planificación dinámica

A partir de los años 80, se empieza a usar una nueva aproximación más adecuada, es la planificación en tiempo de ejecución o planificación dinámica. En ella no existirá un plan de ejecución previo, sino que el planificador elegirá en cada momento la tarea más urgente, dado que a cada tarea le será asigna una prioridad. Si dicha prioridad de una tarea no varía durante el proceso, se hablará de prioridades fijas y, en caso contrario, de prioridades dinámicas.

Existen diversos algoritmos de planificación para tareas independientes en sistemas monoprocesador, entre los que destacan los presentados por Liu y Layland [37]: 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 [38]. 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 [37] 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 ($Ti = Di$).
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, 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 estas suposiciones, se enunció el método de planificación de prioridad a la tarea más frecuente (RMS). De acuerdo con el algoritmo RMS, las prioridades se asignarán respecto a la frecuencia de las tareas; cuanto mayor sea la frecuencia de una tarea (menor período), mayor será la prioridad:

$$\forall \tau_i, \tau_j \in \Pi : T_i < T_j : p_i > p_j \tag{2.4}$$

A parte de este método, se enunciaron las siguientes definiciones y teoremas:

Definición 2.4. *(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.*

A partir de la definición previa, 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.*

Según este último teorema 2.1, si todas las fases de las tareas del sistema son nulas ($\phi_i = 0 \ \in i = 0 \ldots n$), el instante crítico para todas las tareas del sistema tendrá lugar durante el inicio del mismo.

Del mismo modo, al afirmar que una planificación es correcta o 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. Ello vendrá dado por el siguiente teorema:

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.*

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. Ellos definen el factor de utilización del

procesador como la fracción de tiempo que tarda la tarea en ejecutarse en el procesador. Como $\frac{C_i}{T_i}$ es la fracción de tiempo del procesador usado para la ejecución de la tarea τ, entonces para n tareas, el factor de utilización será:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \tag{2.5}$$

Aunque el factor de utilización del procesador puede mejorar incrementando los valores de los C_i, o decrementando los valores de los T_i, se verá limitado por el hecho que todas las tareas tienen que satisfacer sus *deadlines* en los momentos críticos. No tiene ningún interés saber como de pequeño puede llegar a ser el factor de utilización, sin embargo adquiere gran importancia el conocer el máximo factor de utilización. Al trabajar con asignaciones basadas en prioridades, se dice que un conjunto de tareas hará un uso completo del procesador, si las prioridades asignadas son viables para el conjunto y si un incremento en el tiempo de computación de alguna de las tareas hiciese las asignaciones de prioridades inviables. Para un algoritmo que trabaje con conjuntos de tareas con prioridades fijas, la cota máxima más pequeña del factor de utilización será el mínimo de los factores de utilización sobre todos los conjuntos de tareas que llenan el procesador.

Como la asignación de prioridades $rate - monotonic$ es óptima, el factor de utilización logrado por una asignación de prioridades $rate - monotonic$ para una conjunto de tareas dado es mayor o igual, que el factor de utilización para otra asignación de prioridades cualquiera para ese mismo conjunto de tareas. Por lo tanto, el factor de utilización máximo en el peor caso que debe ser calculado es el ínfimo de los factores de de utilización correspondientes a la asignación $rate - monotonic$ sobre todas los posibles periodos y tiempos de computación requeridos para las tareas. En el caso de dos tareas quedaría reflejado en el siguiente teorema.

Teorema 2.3. *(Liu y Layland, 1973) Para un conjunto de dos tareas con asignación de prioridades fija, el factor de utilización del procesador máximo en el peor caso es:*

$$U = 2(2^{\frac{1}{2}} - 1) \tag{2.6}$$

A partir del cual Liu y Layland son capaces de formular el siguiente teorema,

Teorema 2.4. *(Liu y Layland, 1973) Para un conjunto de n tareas con asignación de prioridades fija, el factor de utilización del procesador máximo en el peor caso será:*

$$U = n(2^{\frac{1}{n}} - 1) \tag{2.7}$$

Del teorema anterior podemos sacar la conclusión de que teniendo un conjunto de N tareas con prioridades fijas, el conjunto será planificable para cualquier relación de fases si:

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

Con ello se afirma que si la utilización del procesador es menor a un determinado umbral, entonces se podrá decir que el conjunto de tareas que lo componen será planificable. Si queremos calcular ese umbral para dos tareas ($N = 2$), el umbral será igual a $0,8383$, siendo para valores más altos de N:

$$\lim_{N \to \infty} N(2^{\frac{1}{N}} - 1) = ln2 \simeq 0,69 \tag{2.9}$$

Cabe destacar que el cumplimiento de esta condición es suficiente pero no necesario, ya que pueden existir conjuntos de tareas planificables pero que fallen a la hora de cumplir la condición antes expuesta.

Las suposiciones en las que se basa RMS generan un modelo de sistema poco realista que reduce su aplicabilidad. En 1989, J. Lehoczky, L. Sha, y Y. Ding [39] presentan una técnica para el análisis exacto de planificabilidad bajo RMS. Ésta se basa en la comprobación que la carga solicitada al procesador sea menor que el tiempo transcurrido entre la activación de la tarea y su finalización. Al no ser viable realizar comprobaciones en todos los instantes temporales, ellos mismos llegan a la conclusión que sólo han de comprobarse 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 2.5. *(Lehoczky et al,1989) Sea* $\prod = \{\tau_i, i = 1 \ldots n\}$ *un conjunto de tareas ordenadas en orden decreciente de prioridades. El conjunto de puntos de planificación para una tarea* $\tau_i \in \prod$ *coincidirá con los momentos en los que las tareas de mayor prioridad que ella son activadas y se define como:*

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

Lo cual nos lleva a afirmar que la tarea cumplirá todos los plazos de respuesta para cualquier relación de fases si:

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

Y por lo tanto la utilización en el peor de los casos será:

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

Por lo que se puede reescribir la inecuación 2.12 como sigue:

$$\min_{t \in S_i} U_i(t) \leq 1 \tag{2.13}$$

Y a partir de esta ecuación uno es capaz de deducir 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. Con ello se concluye que el sistema será planificable *si y solo si* se cumple dicha condición para todas las tareas:

$$\max_{\forall i = [1 \dots n]} \left\{ \min_{t \in S_i} \sum_{j=1}^{i} \frac{C_j}{t} \left\lceil \frac{t}{T_j} \right\rceil \right\} \leq 1 \tag{2.14}$$

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 se comentó en párrafos anteriores las suposiciones en las que se basa RMS generan un modelo de sistema poco realista que reducía su aplicabilidad. Cabe destacar que la restricción impuesta a las tareas de que el plazo ha de ser igual al periodo ($D_i = T_i \quad \forall i$) es una restricción muy severa. Como ejemplo de sistema que no podría cumplir esta restricción está el de 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. Trabajando sobre ello Leung y Whitehead [40] definieron el método de prioridad a la tarea más urgente (*Deadline Monotonic Scheduling*, DMS) el cual se define en el siguiente teorema.

Teorema 2.6. *(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.*

Se puede observar evaluando lo anterior que RMS es un caso particular del método DMS cuando Di = Ti. Asimismo en el mismo documento se demuestra que en el caso que los plazos de finalización sean menores o iguales que los períodos ($D_i \leq T_i$) la asignación DMS será óptima. Apoyándose en este trabajo Audsley [24] desarrolló un test de planificabilidad suficiente basado en utilizaciones para DMS:

Sea $\prod = \tau_i, i = 1 \ldots N$ un conjunto de tareas ordenadas en orden decreciente de prioridades, entonces la interferencia que la tarea τ_i experimentará debido a tareas con igual o mayor prioridad que ella estará acotada por la siguiente expresión:

Teorema 2.7. *(Audsley, 1991)*

$$I_i = \sum_{j \in h_p(i)} \left\lceil \frac{D_i}{T_j} \right\rceil C_j \tag{2.15}$$

donde $h_p(i)$ es el conjunto de tareas que tienen mayor o igual prioridad que y por lo tanto pueden interferir en la ejecución de esta.

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

$$\frac{C_i}{D_i} + \frac{I_i}{D_i} \leq 1, \forall i = 1 \ldots N \tag{2.16}$$

El análisis exacto se obtiene extendiendo el test desarrollado por Lehocky et al [39] (Teorema 2.5) al caso DMS, redefiniendo la utilización en el peor caso (ecuación 2.12) como:

$$U_i(t) = \frac{C_i \left\lceil \frac{t}{T_i} \right\rceil + \sum_{j=1}^{i} \left(C_j \left\lceil \frac{t}{T_j} \right\rceil \right)}{t} \tag{2.17}$$

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 [41] y por Audsley et al en [24] 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 [42][43], 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:

$$w = C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w}{T_j} \right\rceil C_j \tag{2.18}$$

La planificabilidad del sistema viene condicionada a que exista una solución a la ecuación anterior (2.18), $\exists R_i \forall i = 1 \ldots N$ 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 [44][45]:

$$\begin{cases} w^0 & = & C_i \\ w^{n+1} & = & C_i + \sum_{\tau_j \in (\tau_i)} \left\lceil \frac{w^n}{T_j} \right\rceil C_j \end{cases} \tag{2.19}$$

Planificación dinámica basada en prioridades dinámicas: C. Liu y J Layland realizaron un estudio de otros algoritmos que fueron presentados en el mismo trabajo en el cual definieron el RMS [37]. Bajo las mismas suposiciones que RMS, introdujeron el algoritmo DDS (*Deadline Driven Scheduling Algorithm*), más conocido por su denominación EDF (*Earliest Deadline First*), el cual se basa 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:

$$\forall \tau_i, \tau_j \in \prod_R : d_i < d_j : p_i > p_j \tag{2.20}$$

En este algoritmo las prioridades son asignadas a las tareas de acuerdo a sus *deadlines*. A una tarea se le asignará la prioridad más alta si su *deadline* es el más cercano, y se le asignará la prioridad más baja en el caso que su *deadline* sea el más alejado en el tiempo. En cualquier instante, la tarea con la prioridad más alta y que aun no haya entrado en el procesador será la primera en ser ejecutada. Este es un método dinámico de asignar

prioridades a las tareas, en oposición al método estático que se usaba para asignar las prioridades en RMS, en el cual las prioridades no cambian a lo largo del tiempo de ejecución.

Teorema 2.8. *(Liu y Layland, 1973) Cuando el algoritmo Deadline Driven Scheduling es usado para planificar un conjunto de tareas en un procesador, no hay tiempo de procesador sin utilizar previo a un overflow*

Este último teorema es usado para establecer el siguiente:

Teorema 2.9. *(Liu y Layland, 1973) Para un conjunto dado de m tareas, el algoritmo Deadline Driven Scheduling es planificable si y solo si*

$$\frac{C_1}{T_1} + \frac{C_2}{T_2} + \frac{C_m}{T_m} \leq 1 \tag{2.21}$$

el cual puede ser expresado de la siguiente manera:

$$\sum_{i=1}^{N} \frac{C_i}{T_i} \leq 1 \tag{2.22}$$

Para concluir cabe resaltar que este algoritmo es óptimo en el sentido que si un conjunto de tareas pueden ser planificadas por cualquier otro algoritmo, entonces podrán ser planificadas asimismo por el algoritmo *Deadline Driven Scheduling*. Aunque esto ya fue intuido por Liu y Laylan en 1973 en su artículo [37], es en 1974 Dertouzos [46] quien lo demostrará.

Unos años después A.Mok y M. Dertouzus propusieron otro algoritmo, el método de prioridad a la tarea con la menor holgura (*Least Laxity First* o LLF). Este nuevo algoritmo será también un algoritmo óptimo. La holgura de una tarea en un instante de tiempo viene definida como la diferencia entre su plazo de finalización absoluto y el tiempo estimado de finalización en el peor caso. El algoritmo *Least Laxity First* 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. Debido a esto, 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. Cabe destacar el análisis de demanda de procesador [47] [48] (*Processor Demand Analysis*), la planificación de tareas aperiódicas basándose en estructuras denominadas servidores (por ejemplo el método *Total Bandwidth Server* [49] [50], TBS, o el método *Constant Bandwidth Server* [51][52], CBS) técnicas para la correcta gestión de las sobrecargas en tiempo de ejecución (como el algoritmo Capacity Sharing [53] o CASH, el algoritmo Greedy Reclamation of Unused Bandwidth [54], GRUB, o la planificación elástica [55] [56][57]), etc. En el articulo [58] puede encontrarse un resumen de las técnicas de análisis de planificabilidad para algoritmos de planificación basados en prioridades dinámicas.

2.1.3.3. Plazos superiores a los períodos

Una situación habitual en los sistemas distribuidos y que requiere una especial atención es el caso en el que las tareas posean un plazo (*deadline*) superior a su período donde una activación en respuesta a un evento debe recorrer distintos recursos hasta su finalización. Así pues el análisis del primer instante crítico, como el realizado en el método desarrollado por Audsley et al [59], 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 [60]. Por lo tanto se plantea como una solución el extender el análisis del instante crítico sobre todas las activaciones de una tarea que ocurran dentro de su período de ocupación de peor caso. Esto fue demostrado en 1990 por Lehoczky [61]. Basándonos en ello y en el método propuesto por Audsley, Tindell et al [62][63] 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:

$$w_i^{n+1}(p) = B_i + (p+1)C_i + \sum_{\tau_j \in h_p(\tau_i)} \left\lceil \frac{w_i(p)^n + J_i}{T_j} \right\rceil C_j \tag{2.23}$$

Para resolver esta ecuación se hace de manera iterativa, iniciando cada iteración sobre p con el valor $w_i^{(0)}(p) = (p+1)Ci$, y deteniendo su análisis cuando $w_i(p) \leq (p+1)Ti$.

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

$$R_i = \max_{\forall p}\{w_i(p) + J_i - qT_i\} \tag{2.24}$$

y se podrá concluir que el conjunto de las tareas será planificable si todos los tiempos de respuesta en el peor caso son inferiores a sus plazos (*deadlines*)

2.2. Sistemas distribuidos

Con el paso del tiempo, al ir mejorando la tecnología de los procesadores, se hizo posible la integración de varios de ellos para así poder delegar diferente funcionalidad a cada uno, formando parte de un mismo sistema. Por lo tanto se ha pasado de una arquitectura centralizada a una arquitectura distribuida más compleja, pero que nos aporta más fiabilidad.

Según Coulouris et al [64] se define un sistema distribuido como:

Definición 2.5. *Un sistema distribuido es 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.*

Por lo que se podría definir un sistema distribuido [1] 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. Adaptando lo anterior a nuestro estudio, un sistema de tiempo real distribuido será un sistema distribuido con restricciones temporales, es decir, un sistema distribuido donde existen actividades con requisitos de tiempo real.

En [64] Coulouris establece que son seis las características principales responsables de la utilidad de los sistemas distribuidos [65]:

Compartición de Recursos El término *recurso* es bastante abstracto, pero es el que mejor caracteriza el abanico de entidades que pueden compartirse en un sistema distribuido. El abanico se extiende desde componentes hardware hasta elementos software.

Los recursos en un sistema distribuido están físicamente encapsulados en una de los ordenadores y sólo pueden ser accedidos por otros ordenadores mediante las comunicaciones (la red). Para que la compartición de recursos sea efectiva, ésta debe ser manejada por un programa que ofrezca un interfaz de comunicación permitiendo que el recurso sea accedido, manipulado y actualizado de una manera fiable y consistente. Por lo tanto surge la figura del gestor de recursos. Un gestor de recursos es un módulo software que maneja un conjunto de recursos de un tipo en particular. Cada tipo de recurso requiere algunas políticas y métodos específicos junto con requisitos comunes para todos ellos. Éstos incluyen la provisión de un esquema de nombres para cada clase de recurso, permitir que los recursos individuales sean accedidos desde cualquier localización; la traslación de nombre de recurso a direcciones de comunicación y la coordinación de los accesos concurrentes que cambian el estado de los recursos compartidos para mantener la consistencia.

Un sistema distribuido puede verse de manera abstracta como un conjunto de gestores de recursos y un conjunto de programas que usan los recursos. Los usuarios de los recursos se comunican con los gestores de los recursos para acceder a los recursos compartidos del sistema. Esta perspectiva nos lleva a dos modelos de sistemas distribuidos: el modelo cliente-servidor y el modelo basado en objetos.

Apertura (*openness*) Un sistema es abierto si el sistema puede ser extendido de diversas maneras. Un sistema puede ser abierto o cerrado con respecto a extensiones hardware (añadir periféricos) o con respecto a las extensiones software (añadir características al sistema operativo, protocolos de comunicación y servicios de compartición de recursos, etc.). La apertura de los sistemas distribuidos se determina primariamente por el grado hacia el que nuevos servicios de compartición de recursos se pueden añadir sin perjudicar ni duplicar a los ya existentes.

La principal característica que ha de cumplir un sistema distribuido con respecto a la apertura es:

- Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo, posiblemente proveniente de vendedores diferentes. Pero la conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

Concurrencia Se dice de la posibilidad de que varias aplicaciones puedan acceder al mismo tiempo a un recurso compartido. Se deben sincronizar los accesos para mantener estados consistentes. Los procesos concurrentes pueden ser ejecutados realmente de forma simultánea, sólo cuando cada uno es ejecutado en diferentes procesadores. En cambio, la concurrencia es simulada si sólo existe un procesador encargado de ejecutar los procesos concurrentes, simulando la concurrencia, ocupándose de forma alternada en uno y otro proceso a intervalos de tiempo. De esta manera se simula que se están ejecutando a la vez.

Debido a que los procesos concurrentes en un sistema pueden interactuar entre otros también en ejecución, el número de caminos de ejecución puede ser extremadamente grande, resultando en un comportamiento sumamente complejo. Las dificultades asociadas a la concurrencia han sido pensadas para el desarrollo de lenguajes de programación y conceptos que permitan hacer la concurrencia más manejable.

Escalabilidad Los sistemas distribuidos operan de manera efectiva y eficiente a muchas escalas diferentes. La escala más pequeña consiste en dos estaciones de trabajo y un servidor de ficheros, mientras que un sistema distribuido construido alrededor de una red de área local simple podría contener varias estaciones de trabajo, varios servidores de ficheros, y otros servidores de propósito específico.

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema se incrementa. La necesidad de escalabilidad no es solo un problema de prestaciones de red o de hardware, sino que está íntimamente ligada con todos los aspectos del diseño de los sistemas distribuidos. El diseño del sistema debe reconocer explícitamente la necesidad de escalabilidad o de lo contrario aparecerán serias limitaciones.

Tolerancia a Fallos Los sistemas a veces fallan. Cuando se producen fallos en el software o en el hardware, los programas podrían producir resultados incorrectos o podrían pararse

antes de terminar la computación que estaban realizando. El diseño de sistemas tolerantes a fallos se basa en dos cuestiones, complementarias entre sí: Redundancia hardware (uso de componentes redundantes) y recuperación del software (diseño de programas que sean capaces de recuperarse de los fallos).

En los sistemas distribuidos la redundancia puede plantearse en un grado más fino que el hardware, pueden replicarse los servidores individuales que son esenciales para la operación continuada de aplicaciones críticas o no criticas, lo que evitaría errores en el caso de la inutilización de un servidor, como ocurriría en el caso de sistemas centralizados.

Los sistemas distribuidos también proveen un alto grado de disponibilidad en la vertiente de fallos hardware. La disponibilidad de un sistema es una medida de la proporción de tiempo que está disponible para su uso. Un fallo simple en una máquina multiusuario resulta en la no disponibilidad del sistema para todos los usuarios. Cuando uno de los componentes de un sistema distribuidos falla, solo se ve afectado el trabajo que estaba realizando el componente averiado. Sin embargo el proceso podría ejecutarse en otra máquina.

Transparencia La transparencia se define como la ocultación al usuario y al programador de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes. La transparencia ejerce una gran influencia en el diseño del software de sistema.

El manual de referencia RM-ODP [ISO-1996] [66] [67] identifica ocho formas de transparencia. Estas proveen un resumen útil de la motivación y metas de los sistemas distribuidos. Las transparencias definidas son:

- Transparencia de Acceso: Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.
- Transparencia de Localización: Permite el acceso a los objetos de información sin conocimiento de su localización
- Transparencia de Concurrencia: Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.

- Transparencia de Replicación: Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan por que conocer la existencia de las replicas.
- Transparencia de Fallos: Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.
- Transparencia de Migración: Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.
- Transparencia de Prestaciones. Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varia.
- Transparencia de Escalado: Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

Las dos más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta fuertemente a la utilización de los recursos distribuidos. A menudo se las denomina a ambas transparencias de red. La transparencia de red provee un grado similar de anonimato en los recursos al que se encuentra en los sistemas centralizados.

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. Por lo tanto se puede decir que 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. Las redes de comunicación son un problema adicional al que se tienen que enfrentar una red distribuida frente a una centralizada, al tener que realizar la planificación de los canales de comunicaciones y la asignación de tareas a los procesadores. A parte de sus características anteriores, cabe destacar la existencia de dos enfoques a la hora del desarrollar sistemas distribuidos [10]:

- Sistemas gobernados por eventos: las acciones son activadas por la ocurrencia de eventos, que caracterizan el flujo de control del programa; los tiempos exactos de activación no son conocidos en tiempo de diseño.

- 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. 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.

Con lo descrito anteriormente se puede resaltar que 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.

Sin embargo el uso de tiempos implica 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 la planificación de cada recurso (nodo o red) puede ser independiente, sin necesidad de realizar un análisis de planificabilidad iterativo.

Asimismo hemos de 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 [29]:

- 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

Tanto la planificación como el análisis en sistemas distribuidos y multiprocesador es a día de hoy un campo en el que la investigación aun está inmersa sin haber sido capaz de

sacar gran cantidad de conclusiones y resultados. Al contrario de lo visto en el apartado 2.1 en el cual se mostraban numerosos y grandes trabajos sobre sistemas monoprocesador, los sistemas distribuidos es un campo abierto a la investigación, en lo que se refiere a la búsqueda de mejores soluciones y más generales, para poderlas aplicar a un número mayor de sistemas de este tipo. Se ha demostrado[68] [69] que muchos problemas de planificación de tiempo real en multiprocesadores son NP-completos (definiciones 2.1, 2.2 y 2.3).

Aun así, los escasos resultados obtenidos al intentar alcanzar la resolución de estos problemas demuestra la complejidad a la que uno se enfrenta cuando se está tratando con sistemas multiprocesador y distribuidos. Asimismo no se pueden dar por supuesto conclusiones obtenidas para los sistemas monoprocesador, ya que se ha demostrado que en ocasiones son contraproducentes para estos últimos sistemas. Sirva como ejemplo el estudio realizado por Stankovic et al [69], en el que se demuestra como el uso de apropiación de procesador no siempre es beneficioso como se afirmaba en sistemas monoprocesador; 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 [35], el algoritmo HOPA [70] (*Heuristic Optimized Priority Assignment*), o los algoritmos WCDO [71] (*Worst Case Analysis for Dynamic Offsets*) y WCDOPS [72](*Worst Case Analysis for Dynamic Offsets with Priority Schemes*).

2.2.1.1. Modelo y política de planificación para el análisis de sistemas distribuidos

Aunque existen diferentes modelos para la realización de la planificación de un sistema distribuido, en este documento solo se va a desarrollar el que vamos a usar para la realización del proyecto fin de carrera. En el modelo presentado, la red se modela como un recurso más sobre el que debe ejecutarse una acción, en este caso la transmisión de un mensaje.

El modelo que se ha utilizado es el **Modelo transaccional de tiempo real**. Es un

modelo computacional, definido por Tindell en [73], 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.

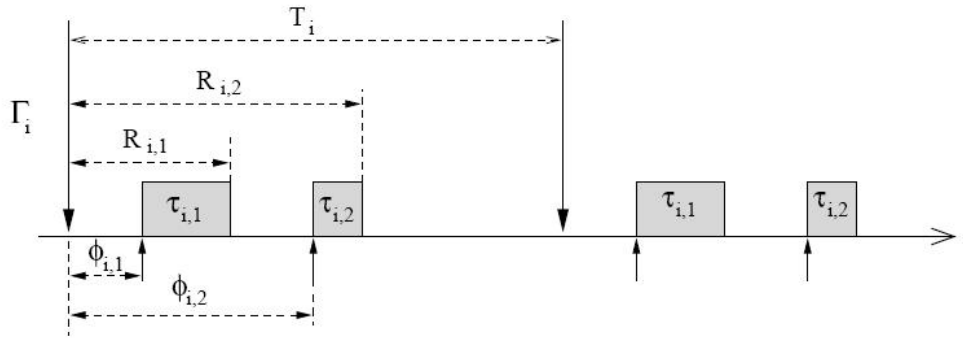

Figura 2.1: Ejemplo de transacción de tiempo real de [1]

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

$$\Gamma_i = \{\tau_{i,j}\left(C_{i,j}, D_{i,j}, T_i, \phi_{i,j}, J_{i,j}, p_{i,j}\right), j = 1 \ldots N\} \tag{2.25}$$

donde $\phi_{i,j}$ representa el desplazamiento temporal u $offset$ y $J_{i,j}$ representará el retraso máximo 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 Modificación de Fase [74], el análisis holístico de Tindell [73][27] y el protocolo Release Guard Protocol [75].

2.2.1.2. Análisis de planificabilidad

Como ya se ha comentado en párrafos anteriores, 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. Los diferentes análisis son los siguientes:

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.

Efecto del retraso en un sistema distribuido A la hora de realizar un análisis de planificabilidad de un sistema distribuido hemos de tener en cuanta tanto la red como la precedencia en la activación de las tareas y mensajes de una secuencia o transacción. En el modelo propuesto por Tindell [62], 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 se active con retraso cero, $J_1 = 0$, las siguientes pueden experimentar un retraso máximo que puede ser equivalente a la máxima variación del tiempo de respuesta de la acción precedente.

Planificación holística bajo la aproximación de tareas independientes Tindell y Clark [62] desarrollaron en su artículo [62] esta técnica para sistemas distribuidos. Para cada acción calcula el tiempo de respuesta en el peor caso aplicando la metodología propuesta por Tindell para sistemas monoprocesador [63] (apartado 2.1.3.3), teniendo solo en cuanta el recurso (red o nodo) donde se ejecuta la acción. En esta aproximación se supone que las tareas son independientes entre ellas, por lo que la estimación de los tiempos de respuesta de las tareas es más pesimista que si tuvieran en cuenta las relaciones de precedencia. Por lo tanto a la hora de definir el instante de activación p-ésima de una acción a_i se define exactamente igual que en los sistemas monoprocesador (ecuación 2.23 del apartado 2.1.3.3):

$$w_i^{n+1}(p) = B_i + (p+1)C_i + \sum_{a_j \in h_p(a_i)} \left\lceil \frac{w_i^n(p) + J_i}{T_j} \right\rceil C_j \tag{2.26}$$

Por lo que el tiempo de respuesta en el peor caso de la tarea τ_i será el máximo de los tiempos de respuesta obtenidos:

$$R_i = \max_{\forall p} \{w_i(p) + R_{i-1} - qT_i\} \tag{2.27}$$

2.3. Paradigma de servicios

Como se introdujo en el primer capítulo, un servicio es [6]: *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 que no tiene dependencias con otros servicios y que está desarrollada, implementada y desplegada individualmente, con sus interfaces bien definidos y que proporciona una determinada funcionalidad, siendo además sin estado.

Los servicios debido a su funcionalidad aportan un valor añadido. Las especificaciones del servicio han de depender de las características del entorno en el que este va a ser utilizado. Por ejemplo no tendrá sentido dotar de grandes medidas de seguridad a un servicio orientado a informar sobre los resultados de los campeonatos deportivos locales. Asimismo una de las características más importantes que debe respetar un servicio es que su implementación no esté limitada a un determinado lenguaje o plataforma de ejecución. Lo que idealmente se debe perseguir es que diferentes servicios implementados sobre lenguajes distintos puedan interaccionar a través de interfaces.

En la figura 2.2 se puede observar el modelo conceptual de una arquitectura orientada a servicios. Se muestra como los proveedores de servicios han de registrar los servicios. El acoplamiento entre proveedores y consumidores se consigue a través de interfaces bien diseñadas así como de la utilización de metadatos, que incluirán su funcionalidad y características más importantes, como por ejemplo su calidad de servicio. Los consumidores, basándose en los metadatos de los servicios, establecerán un contrato con el proveedor del servicio, que obligará a ambos a cumplir una serie de requisitos (como puede ser la calidad de servicio de entrada y/o de salida, peticiones máximas, etc.).

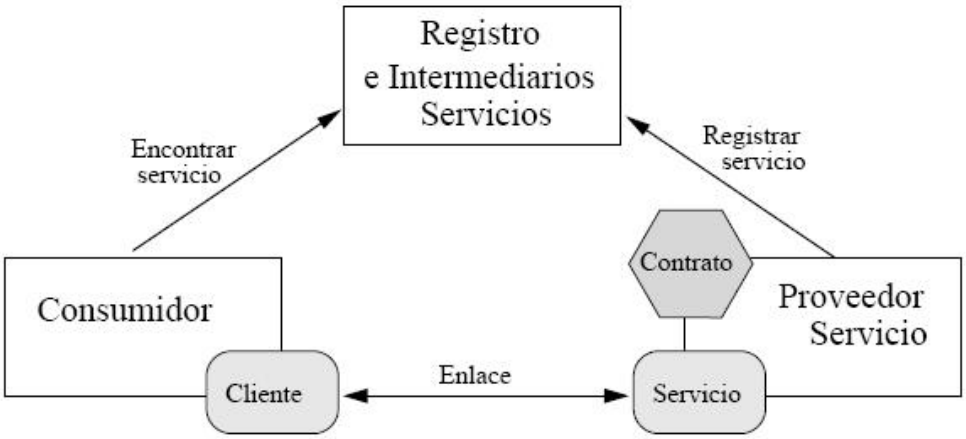

Figura 2.2: Esquemático de una arquitectura orientada a servicios de [1]

La composición de servicios consiste en la combinación de las funcionalidades de servicios ya existentes para la creación de un servicio complejo. A la hora de definir un servicio compuesto en [76] se divide en seis dimensiones:

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

2.3.1. Servicios Web

A principios de esta década nacen los Servicios Web [76], tecnología de soporte a la computación distribuida. Según el *World Wide Web Consortium* (W3C), un servicio Web

es [77] *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.*

Por lo tanto 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. A día de hoy no se puede afirmar que son las tecnologías citadas las que definen el concepto de servicio Web, ya que no existe un consenso respecto al uso de estas tecnologías concretas.

Como se describió anteriormente, la composición de servicios Web es idéntica a la composición de servicios, ya que consiste asimismo 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 [76], es necesario contar con middleware de composición de servicios, que provea abstracciones e infraestructura para la definición y ejecución de servicios compuestos.

Capítulo 3

Modelos Teóricos

En este capítulo se describen los modelos utilizados en el desarrollo de la tesis de Iria Estévez Ayres [1] en la cual está basado este proyecto. En primer lugar se describirá el modelo de las aplicaciones basadas en servicios. Posteriormente, se realizará un estudio sobre el modelo de sistema usado, y el modelo de servicios, para finalizar con el estudio de los algoritmos desarrollados a lo largo de la elaboración de la tesis.

3.1. Modelo del sistema

Uno de los objetivos planteados en la tesis de Iria Estévez Ayres [1] era 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. Lo que se persiguía era que el sistema tuviera la posibilidad de cambiar en tiempo de ejecución las implementaciones de los servicios, perfiles, que empleaba una determinada aplicación, sin que tuviese efectos en las demás aplicaciones existentes en el sistema.

La abstracción que se tomó como partida es 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.

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 [73]

y explicado en la sección 2.2.1.1. 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. En la tesis, se empleó 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.

Una vez que se eligió el modelo para el diseño se hubo de elegir la aproximación usada: sistema gobernado por eventos o sistema gobernado por tiempo. En la tesis se siguió una aproximación hibrida que aúne características de ambos enfoques, la flexibilidad y eficiencia del paradigma gobernado por eventos, y la predictibilidad y seguridad del gobernado por tiempo. Las aplicaciones presentes en el sistema estaban gobernadas por tiempo, y las acciones involucradas tienen instantes de activación predefinidos y siempre mayores que los tiempos de respuesta en el peor caso de las acciones precedentes. Por otra parte, fue 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 permite que las tareas asociadas a éste puedan ser activadas por tiempo, pertenecientes a una transacción, o activadas por eventos del sistema.

3.2. Modelo de Aplicaciones

Como se comentó en el primer capítulo, los sistemas software distribuidos se han vuelto más dinámicos permitiendo distribución, auto-reconfiguración, portabilidad y migración de aplicaciones. Asimismo se señaló que a consecuencia de esto surge la aparición de nuevos paradigmas de desarrollo de aplicaciones basados en el uso de múltiples servicios dispersos en el entorno [78][79]. Estos nuevos paradigmas permiten aportar mayor flexibilidad tanto en el desarrollo como en la ejecución de las aplicaciones. 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.

La tesis de Estévez Ayres [1] 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.

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

- Aplicaciones distribuidas, aunque el modelo permite realizar composición centralizada en un único nodo físico.
- Aplicaciones flexibles, capaces de ejecutarse como parte de un entorno que cambia de forma dinámica pudiendo ajustarse a éste.
- Aplicaciones 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 deban ser deterministas. Por otra parte, los modelos que se presentarán en la tesis, son más adecuados para aplicaciones de tiempo real no críticas, aunque podrían ser aplicados en otros entornos.
- Aplicaciones con 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 puede 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.

Las aplicaciones, en principio, son distribuidas, es decir, los perfiles están dispersos en la red en diferentes nodos físicos, pudiendo varios perfiles residir en el mismo nodo y varias aplicaciones pueden compartir el mismo perfil. La comunicación entre perfiles que residen en nodos diferentes se realiza mediante paso de mensajes, regulados por los protocolos de red subyacentes. Mientras que la comunicación entre perfiles residentes en el mismo nodo físico se realiza mediante búferes de comunicación.

La multiplicidad de perfiles permite al sistema elegir, de entre el conjunto de combinaciones de perfiles posibles, aquélla que más se adecúe a una métrica especificada previamente.

Así, según lo tesis de Iria Estévez Ayres [1] un entorno distribuido que permita composición de aplicaciones de tiempo real debe 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.
- 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.
- 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. Asimismo, también es el encargado de dar el soporte necesario para la reconfiguración de las aplicaciones, en entornos donde se permita la composición dinámica.

  En estos entornos, la multiplicidad de perfiles permite al sistema seleccionar, en tiempo de ejecución, de entre todas las implementaciones disponibles de un servicio, la que más se ajuste a sus necesidades para una aplicación determinada.

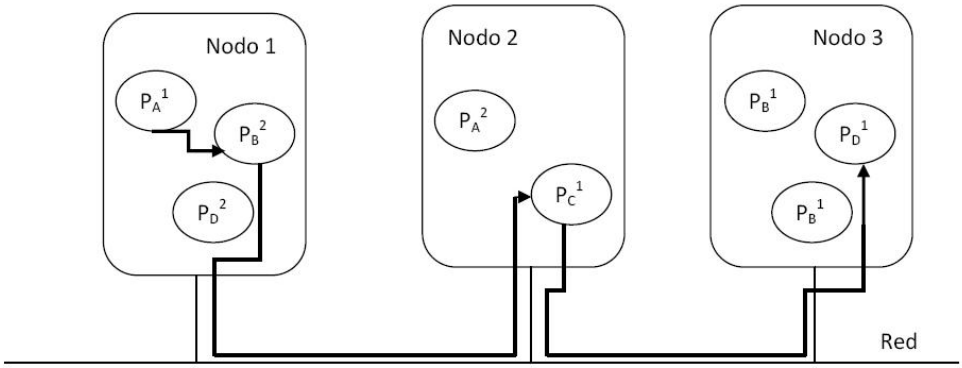

Figura 3.1: Ejemplo de aplicación distribuida

Como podemos ver en la figura 3.1 existe una aplicación compuesta por el conjunto de servicios S_A, S_B, S_C, S_D, en el orden presentado. Como se puede apreciar en la figura existen diferentes perfiles que representan a cada uno de estos servicios y que se encuentran localizados en diferentes nodos en la red. Se puede observar que el servicio S_A está implementado por los perfiles P_A^1 en el nodo 1 y P_A^1 en el nodo 2. En el caso del servicio S_B se puede apreciar que está implementado en el perfil P_B^2 en el nodo 1 y por los perfiles P_B^1 y P_B^2 en el nodo 3. En el caso del servicio S_D tiene el perfil P_D^1 en el nodo 3 y P_D^2 en el nodo 1. Sin embargo para el servicio S_C sólo existe un perfil en el nodo 2.

Posteriormente es el sistema el responsable de elegir los perfiles más adecuados en ese momento determinado para la composición de la aplicación. Como se puede observar en el ejemplo, el sistema en este caso elige los perfiles P_A^1,P_B^2,P_C^1,P_D^1 que resuelven la aplicación planteada.

$$P_A^1 \xrightarrow[R_1,R_1]{msg1} P_B^2 \xrightarrow[R_1,R_2]{msg2} P_C^1 \xrightarrow[R_2,R_3]{msg3} P_D^1 \tag{3.1}$$

Si, en un momento determinado el perfil P_A^1 falla, el sistema puede reconfigurar la aplicación evitando un fallo general de la misma, eligiendo, por ejemplo el perfil P_A^2. Asimismo pueden cambiar también los mensajes entre los diferentes nodos. Con lo que la aplicación resultante es:

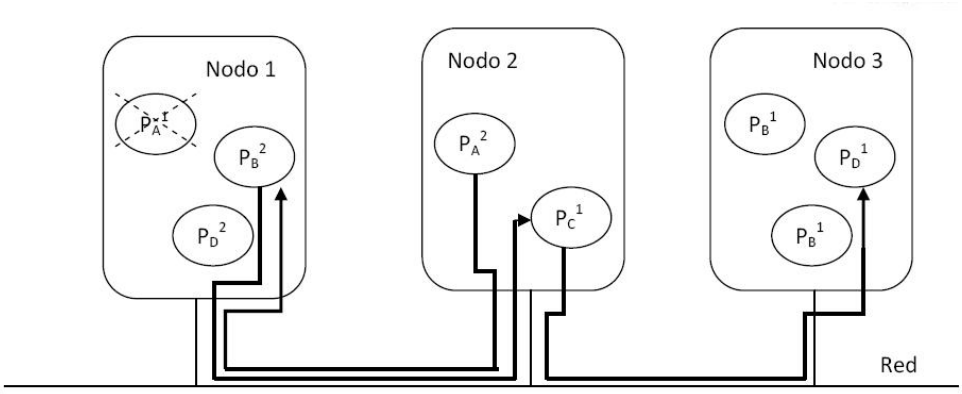

Figura 3.2: Ejemplo de reconfiguración en una aplicación distribuida

$$P_A^2 \xrightarrow[R_2,R_1]{msg1'} P_B^2 \xrightarrow[R_1,R_2]{msg2} P_C^1 \xrightarrow[R_2,R_3]{msg3} P_D^1 \tag{3.2}$$

El modelo desarrollado en la 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.

Por lo tanto se puede concluir respecto al modelo de aplicación utilizado en la tesis, que se basa en la composición de servicios heterogéneos dispersos en una red de tiempo real, en la cual los servicios se invocarán para cada aplicación en una determinada secuencia y se comunicarán entre ellos a través de mensajes. Finalmente una aplicación en ejecución queda definida por los perfiles seleccionados para cada servicio y los mensajes intercambiados entre ellos.

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 el mismo procesador. Sea S la representación del sistema consistente en un conjunto de n servicios:

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

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

$$P_i = \{P_i^1, P_i^2, \ldots, P_i^m\} \tag{3.4}$$

Cuando se desee utilizar un determinado servicio, se selecciona 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 debe tener en cuenta el estado del sistema en ese momento.

La aproximación de servicio implementado por diferentes perfiles puede 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, puede elegir otro perfil, P_i^k , que, implementando la misma funcionalidad, permita que el sistema sobreviva.

Asimismo, como las prestaciones de un perfil varian 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

Cada una de las invocaciones a un perfil se materializa en una única tarea. 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 se basó en el modelo comúnmente empleado de caracterización de tareas (introducido en el apartado 2.1.1). Así, se caracterizará cada invocación periódica del perfil P_i como:

$$\tau_{i,j}^t = (C_{i,j}^t, D_{i,j}^t, T^t i, j, \phi_{i,j}^t, p_{i,j}^t) \tag{3.5}$$

y cada invocación esporádica como:

$$\sigma_{i,j} = (C_{i,j}^t, D_{i,j}^t, T_{min\{i,j\}}^t, p_{i,j}^t) \tag{3.6}$$

Asimismo se pueden clasificar las tareas en cuatro grupos dependiendo de su interacción con el entorno. Una tarea que necesita datos es una consumidora; una tarea que genera datos es una productora; una tarea que necesita y genera datos, una consumidora/productora; 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.

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.

Respecto a la clasificación anterior de las tareas dependiendo su iteración con el entorno, apoyándose en la tesis de Calha [80], en la tesis de Iria Estévez Ayres [1] se consideran tres tipos de tareas: productoras, consumidoras y productoras/consumidoras.

Tareas productoras Como se muestra en la figura 3.3 es una tarea que produce un mensaje. 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} . Asimismo hay que imponer la restricción de que los períodos de las tareas y sus plazos relativos tengan el mismo valor, $T_t = T_{msgP} = D_t = D_{msgP} = T_{DS}$. Donde T_{DS} es el periodo del flujo de los datos. Por lo tanto lo único que nos queda por determinar es el desplazamiento del mensaje con respecto al de la tarea que lo genera, cuyo valor será:

$$\phi_{mensP} = \phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{3.7}$$

donde T_{EC} es la duración del ciclo elemental, el cual se podría definir como el slot de tiempo en el que está dividido tanto el tiempo de ejecución de las tareas como el de los mensajes. Por lo tanto lo que se persigue con la fórmula anterior es hallar el tiempo en el que se podrá mandar el mensaje, que vendrá definido por su retardo, más el número entero de ciclos elementales que necesitará la tarea para transmitirse.

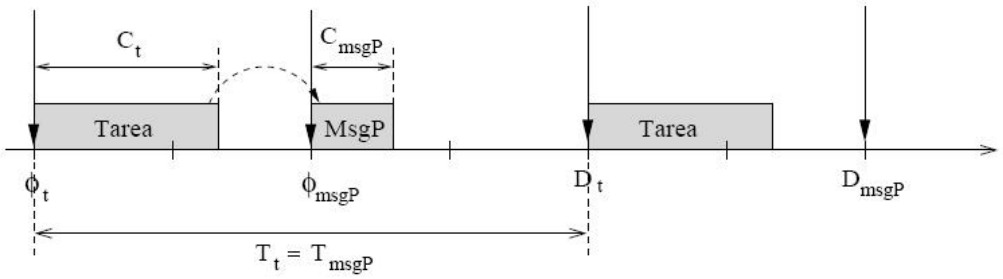

Figura 3.3: Tarea productora

Tareas consumidoras Se define como tal la tarea que consume un mensaje. Al igual que en las tareas productoras se ha de imponer que los períodos de las tareas y sus plazos relativos tengan el mismo valor, $T_t = T_{msgC} = T_{DS} = D_t$. Los parámetros que debemos calcular de la figura 3.4 serán:

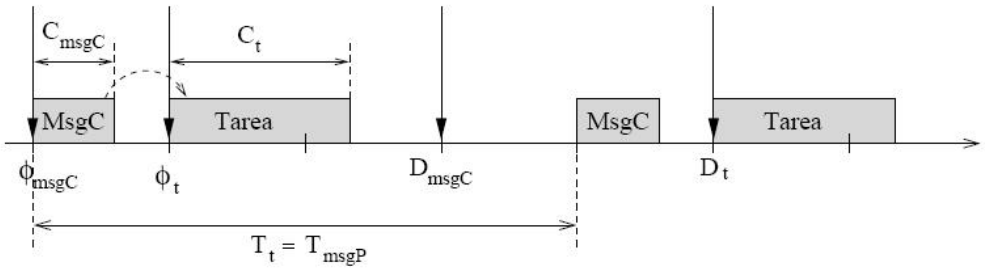

Figura 3.4: Tarea consumidora

el deadline del mensaje consumido:

$$D_{mensC} = T_{DS} - \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{3.8}$$

y el desplazamiento de la tarea con respecto al mensaje recibido:

$$\phi_t = \phi_{mensC} + \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{3.9}$$

Tareas productoras-consumidoras Aquella tarea que consume un mensaje Msg_C y produce con su finalización otro mensaje Msg_P. Las restricciones generales para esta tarea son similares a las de las demás tareas, $T_t = T_{msgC} = T_{msgP} = D_t = T_{DS}$, y los valores que se deben calcular son:

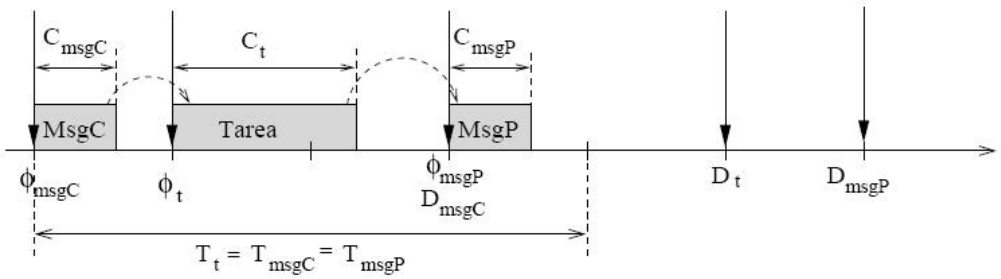

Figura 3.5: Tarea productora-consumidora

el deadline del mensaje consumido:

$$D_{mensC} = T_{DS} - \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{3.10}$$

el desplazamiento de la tarea con respecto al mensaje recibido:

$$\phi_t = \phi_{mensC} + \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{3.11}$$

y el desplazamiento del mensaje con respecto a la tarea consumida:

$$\phi_{mensP} = \phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{3.12}$$

Para concluir se necesita analizar para la aplicación del modelo de la tesis los puntos de sincronización. Estos puntos son aquellos 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. Como se puede observar en la figura 3.6, hay dos puntos de sincronización. El primer punto (PS_1) corresponde a la concurrencia de los servicios S_{21} y S_{22} que llegue a un consumidor-productor. El segundo punto de sincronización es a la conclusión de los servicios S_{31} y S_{23} en el cual desembarcan en un único consumidor.

Conociendo lo anterior, para una tarea consumidora de n mensajes, $\{msg_C\}_{i=1}^{n}$, suponiendo que su ejecución empieza al recibir el último de los mensajes que esperaba, las ecuaciónes 3.9 y 3.11 quedarón modificadas de la siguiente manera:

$$\phi_t = \max_{\forall i=[1...n]} \left\{ \phi^i_{mensC} + \left\lceil \frac{C^i_{mensC}}{T_{EC}} \right\rceil T_{EC} \right\} \tag{3.13}$$

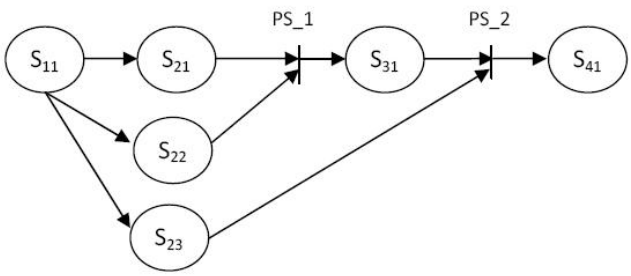

Figura 3.6: Aplicación con dos puntos de sincronización

3.4. Algoritmos usados

En este última parte se estudia cómo se realizó en la tesis la composición de aplicaciones de tiempo real distribuidas basadas en servicios. En primer lugar, se planteó 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 co-dependientes, 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 propuso en primer lugar un algoritmo exhaustivo para la composición de aplicaciones basado en la búsqueda de una combinación de perfiles que minimizase una determinada figura de mérito global que refleja los deseos del usuario y las restricciones impuestas por el propio sistema. Sin embargo, los algoritmos exhaustivos al crecer el número de combinaciones hacen este demasiado costoso para poder ser realizado en tiempo de ejecución. Por lo tanto para poder realizar la composición en tiempo de ejecución del sistema, será 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 basó en el uso de una figura de mérito relativa que refleja la figura de mérito global que se emplea en el exhaustivo, y, además, en el uso de heurísticos que acotan el número de combinaciones a explorar.

3.4.1. Algoritmo exhaustivo

Este algoritmo está basado en la comprobación de todas las combinaciones posibles de los diferentes perfiles que pueden tener los servicios solicitados por una aplicación. Como se dijo anteriormente, este algoritmo tiene el inconveniente del coste que implican a la hora de trabajar con gran número de combinaciones posibles, que hacen que no sean eficaces en

tiempo de ejecución. La gran ventaja de este algoritmo es la selección de los perfiles que más se ajusten a la demanda del usuario así como las restricciones del propio sistema.

El algoritmo exhaustivo presentado en la tesis 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 [81], o mediante el uso de heurísticos desarrollados a tal efecto [60][70].

El algoritmo que se ejecuta dada una petición de un cliente y el estado actual del sistema, es el expuesto en [1] y que se explica a continuación:

1. Se define el nivel 0 de la aplicación como el del primer servicio de la misma.

2. De entre todos los posibles perfiles de los servicios involucrados se descartan aquéllos cuyo tiempo de ejecución en el peor caso (C_i), es mayor que el plazo deseado por la aplicación o los que la suma de su utilización, $U_i = \frac{C_i}{T}$, con la utilización actual del nodo en el que reside sea mayor que 1.

3. Se enumeran los servicios, del grafo completo de la aplicación desde el nivel 1 hasta el nivel N, que será la profundidad máxima de la aplicación.

4. Desde el 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 procede al paso 5.

   Nota: 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 elegida una rama del grafo, se procede a calcular 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.

7. Se devuelve al principio, almacenando la nueva planificación, la figura de mérito y la combinación.

8. Se seguirá iterando sobre todas las posibles combinaciones, 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. Cuando se termine de iterar sobre todos las posibles combinaciones, ya se habrá obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura de mérito dada y la nueva planificación del sistema.

Se puede desprender de los pasos anteriores dos conclusiones importantes. En primer lugar cabe destacar que en caso que la red o alguno de los nodos físicos no fuese planificable, la combinación que se está evaluando no será planificable. En segundo lugar, 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 modelo de servicio 3.3, donde se supone que el sistema se estructura en ciclos elementales (T_{EC}), 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 definida en la ecuación 3.13.

3.4.2. Algoritmo mejorado

Tal y como se adelantó el algoritmo exhaustivo 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. Basándose en lo anterior se presenta el algoritmo mejorado como un algoritmo basado en la disminución del número de ramas a evaluar mediante el uso de heurísticos de poda. El algoritmo mejorado aplica una figura de mérito relativa a los perfiles de cada nivel de la aplicación, en función de la cual ordena los perfiles de ese nivel de mejor a peor. Acto seguido, divide 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 evalua 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 evaluan depende del heurístico empleado. Por otra parte, hay que destacar que la figura de mérito relativa empleada ha de estar fuertemente vinculada a la figura de mérito global.

A continuación se procede a explicar el algoritmo mejorado implementado en [1]. En primer lugar se procede con la fase de preparación. Esta fase se ejecuta en el nivel cero, es decir antes de evaluar ninguna combinación.

1. Primeramente como en el algoritmo exhaustivo, se eliminan los perfiles cuyo tiempo de ejecución en el peor caso sea mayor al plazo deseado de la aplicación y que su instanciación en el nodo físico al que pertenecen no sobrepase una utilización superior a la unidad.

2. Para todos los posibles perfiles se calcula la figura de merito relativa. Nótese que el cálculo de la figura de merito relativa es independiente para cada perfil del sistema.

3. Se calcula para cada nivel la media (μ_s) y la desviación típica (σ_s) de las figuras de merito 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, $nMaxSubc$.

Una vez introducida la fase previa del algoritmo, se explica el algoritmo mejorado, el cual sigue una estructura similar al exhaustivo:

1. En cada uno de los niveles:

   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 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 1*a*.

2. Una vez seleccionada una rama del grafo (último nivel), se calculan los parámetros temporales de la aplicación y, si es planificable, la figura de mérito completa. En caso de error se recoge el nivel exacto donde se produce este. 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 es adecuada, devolviendo la función el nivel donde se produce el error.

3. Si la aplicación es planificable, se comprueba la planificabilidad de las tareas pre-existentes en todos los nodos involucrados. 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.

4. Devuelve a su nivel padre la figura de mérito encontrada para dicha combinación.

5. 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.

6. En el caso de que no haya error, o de que el error se produzca en un nivel inferior, se siguen evaluando perfiles (paso 1*b*) del bloque actual.

7. 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 1*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.

3.4.2.1. Heuristicos desarrollados

A continuación se describen los heurísticos desarrollados con la tesis:

Primera combinación valida Este heurístico refleja el deseo por parte de un cliente de que el sistema escoja la primera combinación válida que encuentre. 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. Los perfiles de cada nivel están 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.

Tamaño de bloque fijo Dado un tamaño de bloque fijo, este heurístico realiza la poda siempre que el número de perfiles sea mayor que el tamaño de bloque especificado. En este caso se evaluan bloques de tamaño fijo para 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.

Dispersion como medida en la realización de la 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 y, además, se supera el número de perfiles en cada nivel, se realiza la poda. Aunque no se alcance este valor de desvío, si el número de perfiles en el nivel supera un determinado valor, 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.

Tamaño de bloque variable Dado que no todos los niveles presentan 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 calcula 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.

Capítulo 4

Decisiones de diseño

En este capítulo se van a evaluar las decisiones que se han tomado a la hora de la implementación de los modelos que se van a utilizar el desarrollo del presente proyecto. Apoyándonos en el estudio realizado en el capitulo anterior, se describirán las diferentes decisiones tomadas a la hora de elegir los modelos de sistemas, aplicaciones y servicios. Además se describirán cómo se han implementado los perfiles de servicio. Finalmente se realizará un estudio de los algoritmos que se han implementado y evaluado en el presente proyecto fin de carrera.

4.1. Diseño de los modelos

Sobre el modelo del sistema del proyecto se han mantenido las principales características presentadas en el apartado 3.1. Aun así se ha excluido uno de los objetivos que se planteaban en la tesis de Estévez Ayres ([1]) que era dotar de flexibilidad a la ejecución de sistemas de tiempo real distribuidos. Lo que se perseguía era que el sistema tuviese la posibilidad de eliminar, añadir o modificar 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 se ha considerado como fuera de los límites del proyecto, considerándose un posible trabajo futuro, por lo tanto se ha prescindido de la flexibilidad de las aplicaciones ya que lo que se busca en este trabajo

es la implementación de algoritmos para la composición inicial de aplicaciones basadas en servicios.

Continuando con el modelo de sistema, se hubo de elegir la aproximación a la hora de activar el sistema. Se decidió seleccionar que el sistema fuera gobernado por tiempo, en vez de hibrido (eventos y tiempos) como fue necesario en la tesis. Esto es debido a que al no desarrollar un sistema flexible no serán necesarios contemplar ejecuciones debidas a eventos de sensores, y por lo tanto nuestro sistema será predecible y periódico en todo momento.

A la hora del modelo de aplicación, se han tomado las características previamente definidas en el apartado 3.2, es decir, el modelo de aplicación se basa en la composición de servicios heterogéneos dispersos en una red de tiempo real, en la cual los servicios se invocarán para cada aplicación en una determinada secuencia. Estos servicios se comunicarán entre ellos a través de mensajes (exclusivamente mensajes, en ningún caso búferes). Por lo tanto finalmente una aplicación en ejecución quedará definida por los perfiles seleccionados para cada servicio y los mensajes intercambiados entre ellos.

Finalmente en el siguiente apartado se explican las decisiones que se han tomado para el diseño de los perfiles, relativas al modelo de servicios.

4.1.1. Perfiles de servicio

En nuestro proyecto solo se va a trabajar con tareas periódicas que como se definió en el apartado 2.1.1 está caracterizada como:

$$\tau_i^t = (C_i^t, D_i^t, T^t i, p_i^t, num_{op}) \tag{4.1}$$

donde C_i^t será el tiempo de computación de la tarea o también denominado tiempo de ejecución en el peor caso de la tarea; D_i^t será el *deadline* de la tarea o tiempo límite de finalización, que coincidirá con el *deadline* del servicio; $T^t i$ se referirá al periodo de la tarea; p_i^t es la prioridad de la tarea y num_{op} será el parámetro que se refiere al código de la operación que se quiere realizar sobre ese nodo.

Como se puede observar se tratan todas las tareas sin offset, por lo que no será necesario definir en las tareas el retardo en la ejecución como se hacía en la tesis (apartado 3.5).

A la hora de encuadrar las tareas dependiendo su interacción con el entorno, nosotros vamos a tener tareas de tres tipos. Tareas que necesitan datos, consumidoras; tareas que generan datos son productoras; tareas que puede ser ambos a la vez, es decir, consumidoras/productoras. Para ilustrarlo nos vamos a servir de un ejemplo. En la figura 4.1, podemos ver una aplicación sencilla, en la que podemos distinguir los tres tipos:

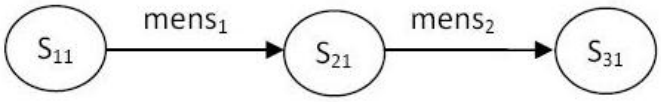

Figura 4.1: Aplicación sencilla

en primer lugar el servicio S_{11} estaría implementado por una tarea productora, ya que será la encargada de la producción del mensaje $mens_1$. Posteriormente la tarea correspondiente al servicio S_{21} será una tarea productora-consumidora, al consumir por un lado el mensaje generado por el servicio anterior y producir el $mens_2$ que finalmente será consumido por el servicio S_{31}, el cual será implementado por una tarea consumidora.

En el caso de los mensajes son considerados entidades independientes, cuya transmisión será gobernada por la red. A la hora de caracterizar un mensaje, se definió como:

$$\tau_i^m = (C_i^m, D_i^m, pos, t_{resp\_mens}) \tag{4.2}$$

donde C_i^m será el tiempo de computación del mensaje o también denominado tiempo de ejecución en el peor caso; D_i^m será el *deadline* del mensaje o tiempo límite de finalización que coincidirá con el tiempo de respuesta de la aplicación; num_{mens} representa la posición del mensaje en el nodo y t_{resp_mens} que representará el tiempo de respuesta del mensaje.

Con respecto a los deadlines calculados para los diferentes tipos de tareas, estos han de ser actualizados para el uso de nuestras aplicaciones, ya que como hemos indicado anteriormente ninguna de las tareas tendrá retardo de ejecución, $offset$, por lo que las ecuaciones quedarán de la siguiente manera:

Tareas productoras Como en la tesis, los parámetros previamente especificados han de ser 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_{mensP} . Habrá también que imponer la restricción de que

los períodos de las tareas y sus plazos relativos tengan el mismo valor que el *deadline* del nodo o servicio, $T_t = D_t = T_{nodo}$ y los periodos de los mensajes y sus plazos sean iguales al *deadline* de la aplicación $D_{mensP} = T_{mensgP} = D_{aplic}$. Donde D_{nodo} representará el *deadline* de un servicio de la aplicación y será igual para todos los servicios. D_{aplic} representa el *deadline* de la aplicación completa y es función del número de nodos que tenga la misma. Por lo tanto lo único que nos queda por determinar es el desplazamiento del mensaje con respecto al de la tarea que lo genera, cuyo valor será:

$$\phi_{mensP} = \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{4.3}$$

donde T_{EC} como ya definimos anteriormente es la duración del ciclo elemental, el cual se podría definir como el slot de tiempo en el que está dividido tanto el tiempo de ejecución de las tareas como el de los mensajes. Se puede observar la diferencia con la ecuación 3.7 en que se añade el retardo en la ejecución de la propia tarea.

Con relación a la definición de T_{EC} como ciclo elemental, existe otro tiempo relacionado con él que es el T_{LSW} que será la longitud de ventana de sincronización o también se podría definir como el tiempo máximo de ocupación de un ciclo elemental. En nuestro caso hemos trabajado con valores de T_{EC} igual a diez unidades de tiempo, mientras que el T_{LSW} son ocho unidades de tiempo.

Tareas consumidoras Al igual que en las tareas productoras se ha de imponer que los períodos de las tareas y sus plazos relativos tengan el mismo valor, $T_t = D_t = T_{nodo}$, así como sucedía en los mensajes $D_{mensP} = T_{mensgP} = D_{aplic}$. Para el caso del *deadline* del mensaje consumidor se podrá calcular con la ecuación 3.8, donde T_{DS} será igual al *deadline* de la aplicación (D_{aplic}). El parámetro a calcular será:

el desplazamiento de la tarea con respecto al mensaje recibido:

$$\phi_t = \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{4.4}$$

en este caso se puede apreciar la ausencia del retardo de ejecución con respecto a la ecuación mostrada en la tesis 3.9.

Tareas productoras-consumidoras Las restricciones generales para esta tarea serán similares a las de las demás tareas, $T_t = D_t = T_{nodo}$ y para los mensajes $D_{mensP} = T_{mensgP}$

$= D_{aplic}$. Como se puede observar, el *deadline* del mensaje consumidor se podrá calcular con la ecuación 3.10 mostrada en el capitulo anterior. Los valores que se deberán calcular son:

el desplazamiento de la tarea con respecto al mensaje recibido:

$$\phi_t = \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC} \tag{4.5}$$

y el desplazamiento del mensaje con respecto a la tarea consumida:

$$\phi_{mensP} = \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC} \tag{4.6}$$

donde también podemos apreciar que difieren de las ecuaciones 3.11 y 3.12

4.2. Algoritmos implementados

4.2.1. Introducción y nomenclatura

En este último apartado del presente capítulo se presentan los diferentes algoritmos diseñados en la elaboración del proyecto. En primer lugar hay que resaltar que todos estos algoritmos han sido implementados en lenguaje *Matlab* y que por lo tanto para su diseño nos hemos basado en el sistema de *functions* o funciones con las cuales trabaja. Una visión más exhaustiva de cada una de las *functions* se puede observar en el siguiente capítulo.

Asimismo cabe destacar la nomenclatura que se ha decidido usar en el proyecto y que difiere en algunos puntos con la usada en el capítulo anterior, y por consiguiente en la tesis. En primer lugar cabe destacar que al igual que en la tesis la misión principal del programa es componer aplicaciones. Estas aplicaciones serán evaluadas por diferentes algoritmos dependiendo sobre las características del sistema que se ejecuten. Los sistemas serán algoritmos de composición y son las *functions* principales del programa, las cuales harán un seguimiento de la aplicación desde su preparación previa hasta el momento en que queda resuelta su planificabilidad. Hemos diseñado cuatro sistemas o algoritmos de composición que se explican a continuación. Las aplicaciones estarán compuestas por servicios, también denominados en el proyecto como operaciones o nodos, los cuales han de estar previamente definidos a la hora de llamar al sistema, y que formarán parte de los

parámetros especificados a priori. Por lo tanto a la hora de trabajar con los algoritmos, se parte de la premisa de que la entidad encargada de componer las aplicaciones posee toda la información relativa al estado actual del sistema (servicios existentes, sus perfiles asociados, la carga en cada nodo físico y la carga total del sistema). Asimismo de cada perfil se conoce su tiempo de ejecución en el peor caso, pero como lo que se pretende con la ejecución de los algoritmos es comprobar la planificabilidad, se deberá calcular su tiempo de respuesta. Es labor del algoritmo 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 (con respecto al T_{EC}). Asimismo cabe resaltar que existirán tres periodos/deadlines. El denominado *deadline* de nodo, que se referirá al *deadline*/periodo de cada servicio, el *deadline*/periodo de red el cual se define para su uso en la configuración de los mensajes en la red y el *deadline* de aplicación que resolverá el tiempo máximo de respuesta de una aplicación y que tendrá relación con el número de nodos que tenga la aplicación. A la hora de elegir el *deadline*/periodo del nodo se ha hecho de forma aleatoria para cada aplicación.

En relación a las representaciones en forma de diagramas de flujo que se van a utilizar para facilitar las explicaciones, cabe destacar el detalle de los diferentes colores y sus significados. En la tabla 4.1 quedan definidas las equivalencias por orden alfabético.

Color	**Representación**
Amarillo	Decisión if/else
Azul	Función
Gris	Acción a realizar
Morado	Bucle *for*
Naranja	fichero aplicación
Rojo	Bloque
Verde claro	Inicio/Fin diagrama
Verde oscuro	Sistema

Tabla 4.1: Equivalencias entre los colores y sus representaciones en los diagramas de flujo

Otra diferencia que se debe resaltar es como se divide la arquitectura. A diferencia de la tesis donde las aplicaciones se iban dividiendo por niveles, padres e hijos, en este proyecto los diferentes servicios padres se van a denominar como nodos en serie, denominando a los hijos como nodos en paralelo. Sin embargo si se va a conservar la denominación de niveles para los diferentes nodos en serie, partiendo del nivel cero y terminando en el nivel N.

4.2.2. Sistema Generalista

Este será el sistema primario del proyecto, ya que a partir de él se han ido desarrollando todos los demás sistemas. Este sistema será el empleado cuando se desea ejecutar el algoritmo exhaustivo para el cálculo de la mínima figura de merito global y la planificabilidad del sistema.

4.2.2.1. Algoritmo Exhaustivo

Al igual que en la tesis y basándose en sus especificaciones se ha desarrollado este algoritmo que realizará la comprobación de todas las combinaciones posibles de los diferentes perfiles que pueden tener los servicios solicitados por una aplicación. A diferencia del algoritmo presentado en el capítulo anterior 3.4.1, en nuestra implementación sí se ha tenido en cuenta la asignación de prioridades, siendo esta realizada de forma estática en tiempo de ejecución. Para la definición de las distintas prioridades de las tareas, se ha tenido en cuenta que se está trabajando con RMS (*Rate Monotonic Scheduling*), método de asignación de prioridades estático que será función del periodo de cada tarea. Por lo tanto de acuerdo al uso de RMS, las prioridades se asignaran monotónicamente con respecto a la frecuencia de las tareas; cuanto mayor sea la frecuencia de la tarea (menor periodo/deadline), mayor será la prioridad (véase ecuación 2.4).

El algoritmo exhaustivo que se ejecuta dada una petición de un cliente, viene representado junto con el sistema genérico en la figura 4.2 bajo un diagrama de bloques. El funcionamiento general de este algoritmo se explica a continuación, dejando su desarrollo específico para el capitulo siguiente (apartado 5.1.1.3):

1. Una vez cargadas todas las características previas tanto de los nodos, como de los servicios y perfiles, se procede a generar todas las combinaciones posibles. Estas se generarán teniendo en cuenta todos servicios de la aplicación y sus posibles perfiles.

2. Mediante un bucle se va seleccionando todas las combinaciones antes generadas, que representarán cada uno de los posibles caminos dentro del grafo de la aplicación.

3. Trabajando con bucles se irá evaluando la planificabilidad y tiempos de respuesta de los distintos servicios:

   a) Se evalúa un servicio en serie. En caso de que no fuese planificable ese servicio se descarta la combinación y se va al paso 2. Si no existen servicios en paralelo se volvería de nuevo a este paso (Paso 3*a*) hasta haber evaluado todos los servicios de la combinación elegida, tras lo cual se pasará al paso 4.

   b) Se evalúan todos los servicios en paralelo. En caso de no ser alguno planificable se pasa al paso 2.

   c) Si han existido paralelos y tienen el punto de sincronización en el próximo nivel, se calculará el camino máximo del paralelo. Si existen más nodos en serie por evaluar, pasamos a los nodos del siguiente nivel (Paso 3*a*)

4. Una vez completada la evaluación de todas las combinaciones, se puede confirmar que la combinación es planificable en el sistema. En caso de tener la mínima figura de merito para la aplicación se almacenará como la combinación óptima.

5. Si no se han terminado de evaluar todas las combinaciones se vuelve al paso 2.

6. Cuando se termine de iterar sobre todas las posibles combinaciones, ya se habrá obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura de mérito dada y la planificación del sistema. Por lo tanto se puede proceder a introducirla en el sistema real.

Comparando el algoritmo ahora expuesto con el desarrollado en la tesis, se pueden apreciar dos mejoras. En primer lugar, este algoritmo realiza las comprobaciones de la planificabilidad simulando el sistema, por lo que no se ve obligado a estar modificándolo en cada

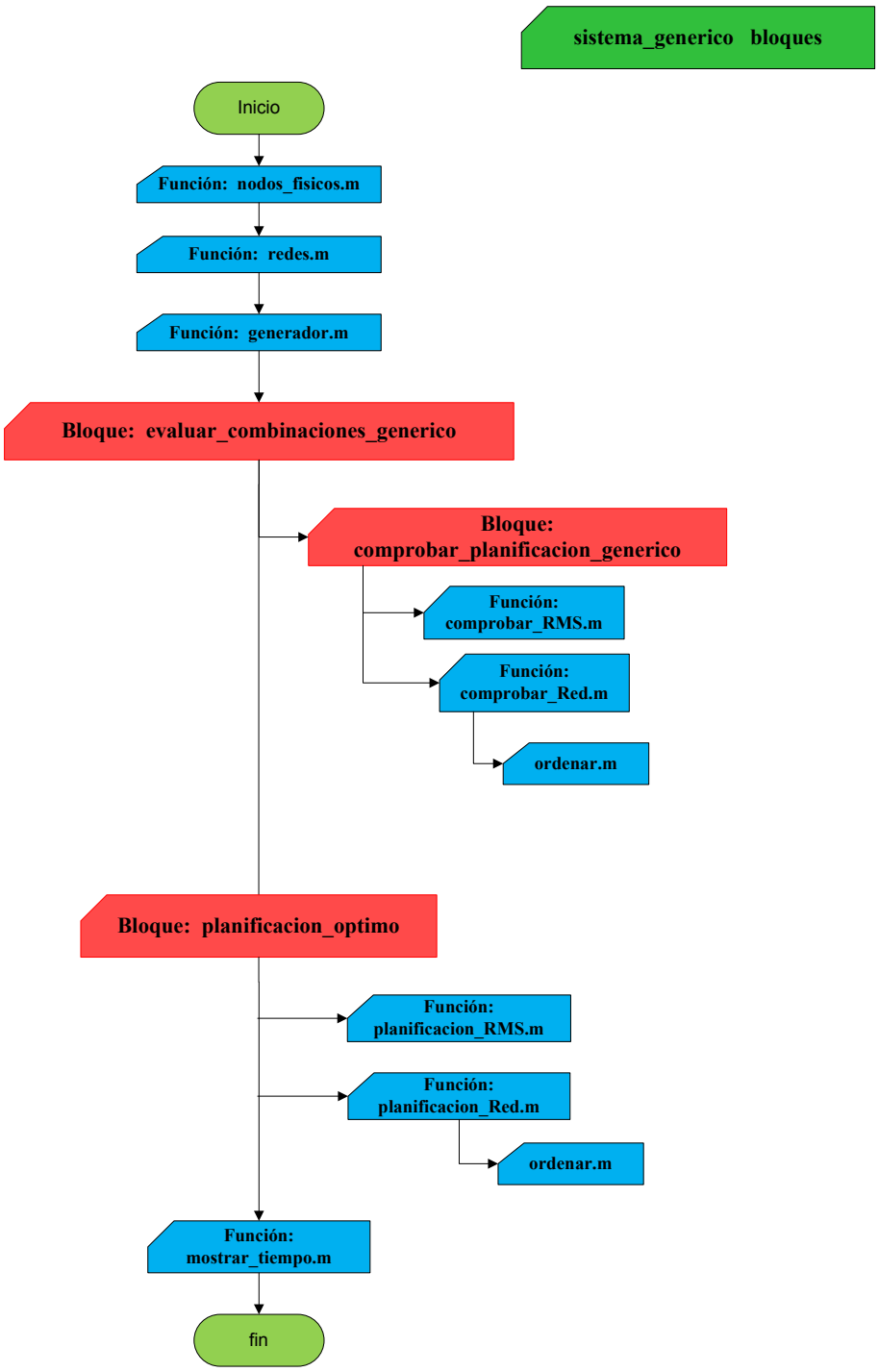

Figura 4.2: Diagrama explicativo de bloques del sistema generalista

evaluación. Primeramente realiza las comprobaciones necesarias sobre simulaciones del sistema, para una vez elegido la combinación óptima ya ejecutarla en el sistema. Otra mejora es que este algoritmo en cualquier momento de la comprobación si un perfil no es planificable, descarta ya toda la combinación, sin necesidad de evaluar el resto de los perfiles de esa misma combinación.

Asimismo dos de las propiedades más importantes que se podían desprender del algoritmo desarrollado en la tesis se siguen manteniendo: en primer lugar cabe destacar que en caso que la red o alguno de los nodos físicos no fuese planificable, la combinación que se está evaluando no será planificable. En segundo lugar, 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, es decir el deadline de la aplicación.

En contra, como ya se describió anteriormente en este capítulo, al no dotar de flexibilidad al sistema no es necesario verificar que el resto de las aplicaciones en el sistema no se vean afectadas por la recién introducida.

Como ya se estudio en la tesis este algoritmo tiene el inconveniente del coste que implica trabajar con aplicaciones con un gran número de combinaciones posibles. Eso nos llevó a la implementación de diferentes algoritmos que procedemos a explicar a continuación.

4.2.3. Sistema Heurístico

Este sistema es una adaptación del sistema mejorado utilizado en la tesis de Iria Estévez [1], para su uso en nuestro proyecto fin de carrera. Como se explicó en 3.4.2, el algoritmo exhaustivo tiene la desventaja de la complejidad que adquiere cuando aumenta el número de servicios de una aplicación. Basándose en lo anterior se presentó el algoritmo mejorado como un algoritmo basado en la disminución del número de ramas a evaluar mediante el uso de heurísticos de poda. Como este algoritmo hace uso de heurísticos, nosotros hemos decidido aplicar un nombre más correcto al mismo denominándole algoritmo heurístico. Por definición [82] "*un algoritmo heurístico es un procedimiento de búsqueda de soluciones casi óptimas a un coste computacional razonable, sin ser capaz de garantizar la factibilidad de las soluciones empleadas ni determinar a qué distancia de la solución óptima nos en-*

contramos". Por lo tanto como lo que nuestro algoritmo pretende es devolver en un tiempo de computación razonable, una solución planificable, pero sin asegurarse de que esa sea la óptima, ya que no se evalúan todas, creemos que el nombre de algoritmo y sistema heurístico es más correcto.

4.2.3.1. Algoritmo Heurístico

Como se explicó anteriormente el algoritmo heurístico 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 bloques, de tal manera que sólo evaluará un número determinado de ramas del grafo total, al evaluar solo determinados bloques. El tamaño de los bloques ha de ser elegido dependiendo del heurístico que se quiera aplicar. Por otra parte, hay que destacar que la figura de mérito relativa empleada ha de estar fuertemente vinculada a la figura de mérito global. Los pasos básicos del algoritmo desarrollado se pueden seguir en el diagrama de bloques 4.3 y cuya explicación se realiza a continuación:

1. Una vez cargadas todos las características previas tanto de los nodos, como de los servicios y perfiles, se procede a comprobar si algún nodo tiene el tiempo de computación mayor que el deadline del nodo, al mismo tiempo que se comprueba si la utilidad del nodo supera el umbral establecido, en este caso el máximo, la unidad, cuando se introduzca la nueva tarea con su tiempo de computación.

2. Se ordenan las tareas (perfiles) de cada servicio por su figura de merito relativa, es decir un criterio parcial.

3. 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, esto es consecuencia del paso 2

4. Se seleccionan los primeros bloques para cada servicio.

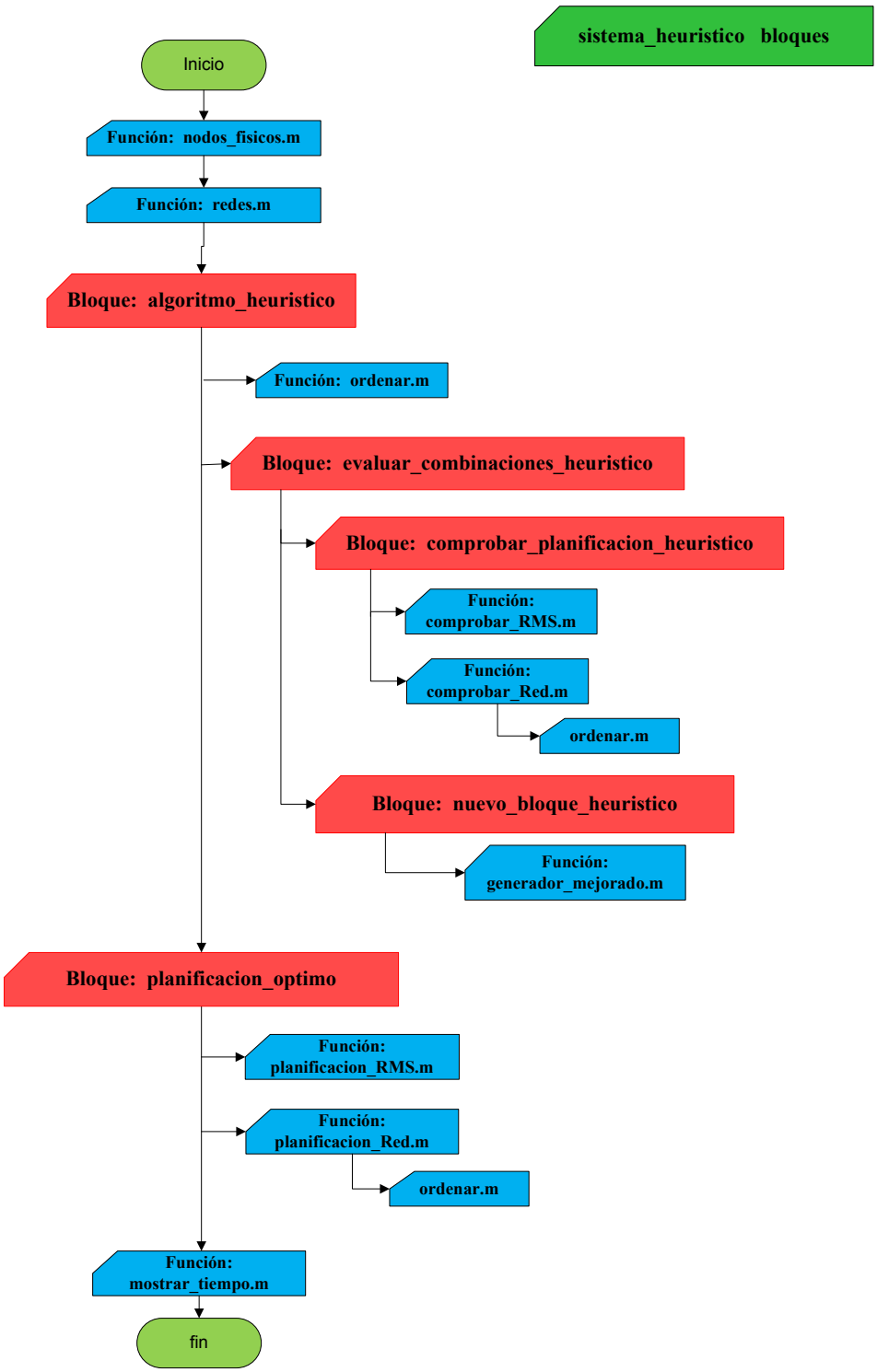

Figura 4.3: Diagrama explicativo de bloques del sistema heurístico

5. Se procede a generar todas las combinaciones posibles de los bloques seleccionados. Estas se generarán teniendo en cuenta todos servicios de la aplicación y sus posibles perfiles incluidos en los bloques seleccionados.

6. En caso que el número de combinaciones evaluadas sea menor que el umbral máximo de combinaciones a evaluar, se van seleccionando, mediante un bucle, todas las combinaciones antes generadas, que representarán cada uno de los posibles caminos dentro del grafo de la aplicación. En caso de superar el umbral citado se accede al paso 11

7. Trabajando con bucles se irá evaluando la planificabilidad y tiempos de respuesta de los distintos servicios:

   a) Se evalúa un servicio en serie. Si no existen servicios en paralelo se volvería de nuevo a este paso (Paso 7*a*) hasta haber evaluado todos los servicios de la combinación elegida, tras lo cual se pasará al paso 8. En caso de que no fuese planificable ese servicio se descarta la combinación y se va al paso 6.

   b) Se evalúan todos los servicios en paralelo. En caso de no ser alguno planificable se pasa al paso 6.

   c) En caso de que algún mensaje haya sido no planificable significará que la red está saturada y no es planificable por lo que será imposible encontrar una combinación planificable ya que no se van a poder enviar los mensajes a través de la red, por lo tanto se accede al paso 11 para la finalización del sistema.

   d) Si han existido paralelos y tienen el punto de sincronización en el próximo nivel, se calculará el camino máximo del paralelo. Si existen más nodos en serie por evaluar, pasamos a los nodos del siguiente nivel (paso 7*a*)

8. Una vez terminada la evaluación, se puede confirmar que la combinación es planificable en el sistema. En caso de tener la mínima figura de merito para la aplicación, se almacenará.

9. Si no se han terminado de evaluar todas las combinaciones se vuelve al paso 6.

10. Cuando se termine de iterar sobre todos las posibles combinaciones, ya se habrá obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura

de mérito dada y la nueva planificación del sistema (se avanza al paso 11). En caso de no existir ninguna combinación planificable se procede con el paso 10*a*:

a) Conociendo el primer servicio en el que no ha sido planificable ninguno de los perfiles del bloque, se procede a elegir el siguiente bloque de ese servicio. Se retrocederá al paso de generación de las nuevas combinaciones (paso 5). En caso de haber agotado todos los bloques se procederá a informar al sistema de la imposibilidad de planificación de la aplicación y a finalizar el mismo.

11. Si se ha conseguido encontrar una combinación planificable, se puede proceder a introducirla en el sistema real. En caso contrario se procederá a informar al sistema de la imposibilidad de planificación de la aplicación y a finalizar el mismo.

Como se puede apreciar existen una serie de diferencias con el algoritmo mejorado presentado en la tesis 3.4.2. Una de las principales novedades es que no hemos creído necesario el cálculo para cada nivel de la media (μ_s) y la desviación típica (σ_s) de las figuras de merito relativas de los perfiles disponibles. Con ello se obtenía una dispersión de los valores que servía para decidir si hacer o no la poda. Nosotros hemos decidido que en caso de que sea elegido este algoritmo se realice la poda bajo cualquier dispersión, por lo que no será necesario el uso de la media y la desviación. Otra diferencia se produce en el orden en que se realizan las comprobaciones de los bloques. Es decir, el algoritmo de la tesis va nivel a nivel evaluando los bloques, y en caso de que alguno no sea planificable está programado para elegir el siguiente bloque de ese nivel. Sin embargo en nuestro caso en un principio se elegirán los bloques, los cuales contendrán los perfiles a evaluar. Una vez seleccionados se procederá a generar todas las combinaciones posibles entre los perfiles de esos bloques. En caso de no planificabilidad en algún nodo, si no se encuentra ninguna combinación planificable, se procede a seleccionar otro nuevo bloque de ese servicio y a generar las nuevas combinaciones. Esta última diferencia nos lleva a que la similitud que muestran ambos marcando un umbral máximo en la evaluación para evitar que el algoritmo llegue a evaluaciones exhaustivas de la aplicación, difiera a la hora de asignarlo. Mientras en la tesis se tomara como umbral máximo un número de bloques a evaluar, en el proyecto hemos tomado un número de combinaciones. Esta decisión es consecuencia del orden en que se realizan las comprobaciones de los bloques que fue expresado anteriormente. El

nuevo umbral es calculado como un cuarto del número máximo de combinaciones posibles, donde el número de combinaciones posible se calcula como:

$$num_{comb} = \prod_{i=1}^{N} P_i \tag{4.7}$$

donde i va recorriendo los N servicios/operaciones que tiene la aplicación, y P_i se refiere al número de perfiles que posee cada servicio.

4.2.4. Sistema Mejorado

Este sistema será de nueva creación en el presente proyecto fin de carrera. Con la experiencia previa adquirida en el desarrollo de la tesis, se creyó necesaria la creación de un algoritmo más competitivo para aplicaciones con gran número de combinaciones a evaluar, en las cuales el exhaustivo se muestra su gran complejidad y su gran tiempo de respuesta que le hacen inutilizables para sistemas de tiempo real, así como la dificultad en determinados casos del algoritmo heurístico para poder seleccionar la tarea más óptima entre todas las posibles a la hora de elegirla para planificarla en el sistema. Lo que se persigue con este algoritmo es acotar el tiempo de respuesta, pero sin renunciar a conseguir la combinación óptima de entre todas las existentes. Por lo tanto será una mezcla entre los dos algoritmos antes introducidos, seleccionando de cada uno de ellos sus principales virtudes, el exhaustivo (combinación óptima) y el heurístico (menor tiempo de respuesta).

4.2.4.1. Algoritmo Mejorado

Como se ha introducido anteriormente, este algoritmo es parte de las aportaciones de este proyecto fin de carrera, por lo tanto se va a explicar su funcionamiento de manera general, para posteriormente poder evaluarlo. Al igual que los dos anteriores, se sigue conservando la asignación de prioridades de forma estática en tiempo de ejecución y se sigue trabajando con *Rate Monotonic Scheduling*. Su funcionamiento es el siguiente:

1. En primer lugar se cargan todas las características previas tanto de los nodos, como de los servicios y perfiles. Posteriormente se procede a comprobar si algún nodo tiene el tiempo de computación mayor que el deadline del nodo, al mismo tiempo que se

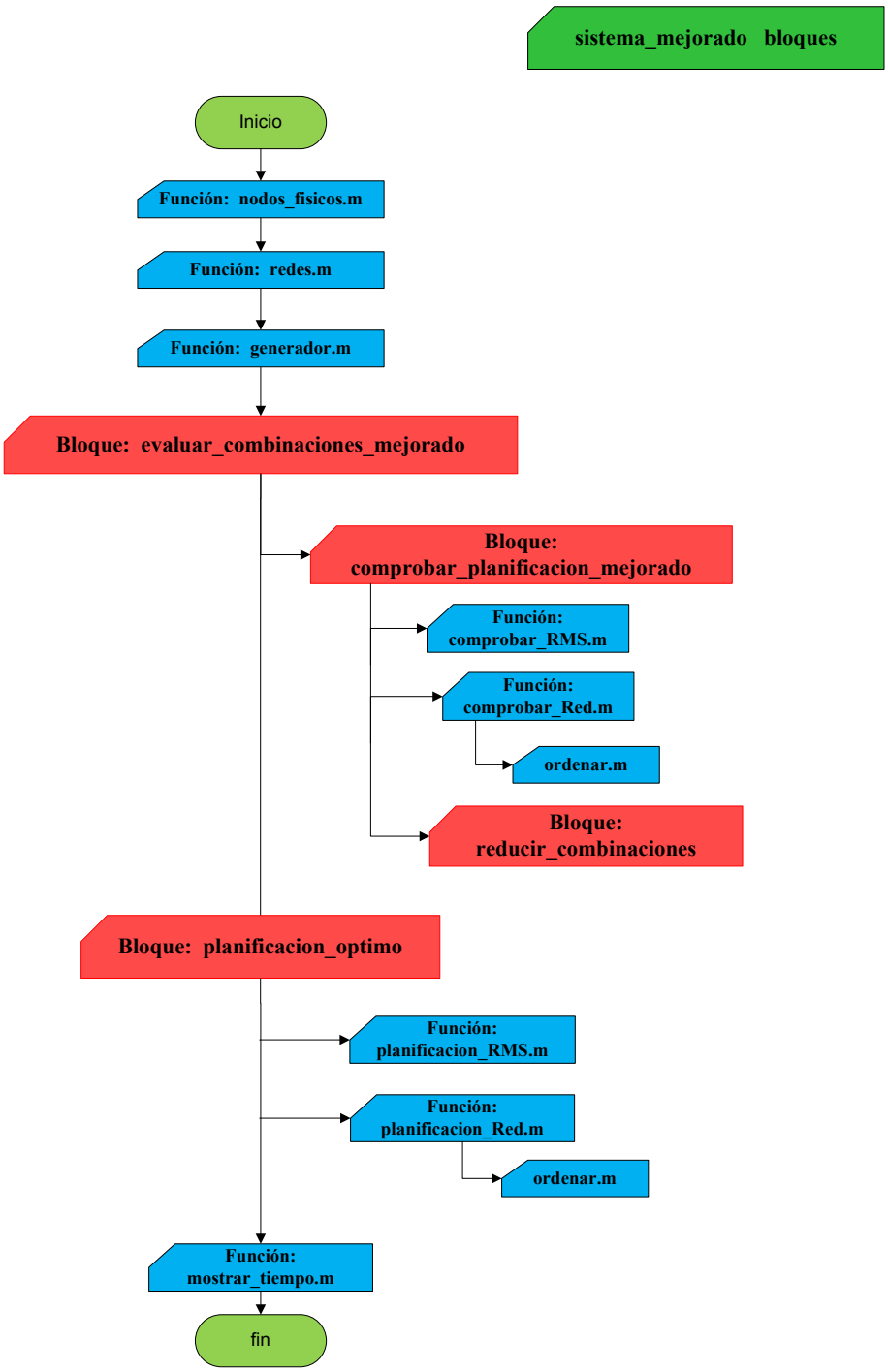

Figura 4.4: Diagrama explicativo de bloques del sistema mejorado

comprueba si la utilidad del nodo supera el umbral establecido, en este caso el máximo, la unidad, cuando se introduzca la nueva tarea con su tiempo de computación.

2. Apoyándose en cálculos previos, estamos en disposición de generar todas las combinaciones posibles. Esta generación será idéntica a la del algoritmo exhaustivo

3. Mediante un bucle se van seleccionando todas las combinaciones antes generadas, que representarán cada uno de los posibles caminos dentro del grafo de la aplicación:

4. Trabajando con bucles se irán evaluando la planificabilidad y tiempos de respuesta de los distintos servicios tanto en serie como en paralelo:

   a) Se evalúa un servicio en serie. Si no existen servicios en paralelo se volvería de nuevo a este paso (Paso 4*a*) hasta haber evaluado todos los servicios de la combinación elegida, tras lo cual se pasará al paso 5. En caso de que no fuese planificable ese servicio se descarta la combinación y se va al paso 4*d*.

   b) Se evalúan todos los servicios en paralelo. En caso de no ser alguno planificable se pasa al paso 4*d*.

   c) En caso de que algún mensaje haya sido no planificable significará que la red está saturada y no es planificable por lo que será imposible encontrar una combinación planificable ya que no se van a poder enviar los mensajes a través de la red, por lo tanto se accede al paso 7 para la finalización del sistema.

   d) En caso que algún nodo no haya podido ser planificable, el sistema mejorado irá eliminando las posibles combinaciones que no van a ser planificables, antes de evaluarlas. Para ello eliminará todas las combinaciones cuya combinación de perfiles hasta el servicio no planificable, sea igual a la combinación ya descartada como no planificable, así como todas las combinaciones que sobre el nodo físico no planificable, desarrollen los mismos perfiles que la combinación ya detectada. Como la combinación que estamos evaluando ya no puede ser planificable, procedemos a evaluar otra combinación (paso 3)

   e) Si todos los nodos han sido planificables y han existido paralelos y tienen el punto de sincronización en el próximo nivel, se calculará el camino máximo del paralelo. Si existiesen próximos niveles se retrocede al paso 4*a*.

5. Una vez completada la evaluación de todas las posibles combinaciones, se puede confirmar que la combinación es planificable en el sistema. En caso de tener la mínima figura de mérito para la aplicación se almacenará como la combinación óptima.

6. Si no se han terminado de evaluar todas las combinaciones se vuelve al paso 3.

7. Cuando se termine de iterar sobre todas las combinaciones que vayan quedando, ya se habrá obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura de mérito dada y la nueva planificación del sistema. Por lo tanto se puede proceder a introducirla en el sistema real.

Del algoritmo anterior se pueden sacar las siguientes conclusiones:

- Este algoritmo será idéntico al algoritmo exhaustivo para el caso de estar trabajando sobre nodos físicos poco cargados y con tareas que tengan tiempos de computación no excesivamente altos. Esto es debido a que en caso de tener descargados los nodos físicos, todas las posibles combinaciones podrán ser planificables y por lo tanto el algoritmo evaluará todas ellas, al igual que realiza el exhaustico.

- En el caso de nodos físicos cargados, el algoritmo funcionará mejor para aplicaciones con un gran número de servicios, ya que estas tendrán mayor número de repeticiones en la creación de las combinaciones y por lo tanto en caso de detectar algún servicio no planificable, el número de combinaciones que se podrán eliminar y no evaluar será mayor.

- Comparándolo con el algoritmo exhaustivo, en principio solo podemos garantizar que se asegura con este algoritmo la obtención de la combinación óptima en cada una de sus ejecuciones, en contraposición del algoritmo heurístico, que no en todos los casos seleccionaba la mejor combinación para cada una de sus ejecuciones

4.2.5. Sistema Heurístico Mejorado

Para finalizar se decidió aplicar las ventajas observadas en el sistema heurístico y en el sistema mejorado para elaborar un nuevo sistema que sea mezcla de ambos. De esa colaboración entre ambos sistemas nace el sistema heurístico mejorado, cuyo algoritmo

perseguirá una reducción del tiempo de ejecución del sistema trabajando con bloques, pero teniendo asimismo en cuenta la posibilidad de reducir en caso de no planificabilidad, el número de combinaciones generadas a la hora de elegir los bloques.

4.2.5.1. Algoritmo Heurístico Mejorado

Como se ha introducido anteriormente, el algoritmo heurístico mejorado es una mezcla de las principales aportaciones de los dos algoritmos anteriores (heurístico y mejorado). Lo que se ha perseguido con la creación de este algoritmo es la maximización de resultados cuando el sistema se encuentre en fase de saturación, es decir que los nodos físicos se encuentren bajo una utilización alta. Para ello el algoritmo a desarrollar tiene su esquema básico en el algoritmo mejorado y su sistema por bloques, pero añadiéndole la funcionalidad que en caso de existir alguna combinación no planificable, las combinaciones futuras que sigan el mismo patrón de la actual sean eliminadas sin necesidad de evaluación. Siguiendo lo anterior los pasos básicos a seguir por el algoritmo son los siguientes:

1. Una vez cargadas todos las características previas tanto de los nodos, como de los servicios y perfiles, se procede a comprobar si algún nodo tiene el tiempo de computación mayor que el deadline del nodo, al mismo tiempo que se comprueba si la utilidad del nodo supera el umbral establecido, en este caso el máximo, la unidad, cuando se introduzca la nueva tarea con su tiempo de computación.

2. Se ordenan las tareas (perfiles) de cada servicio por su figura de merito relativa, es decir un criterio parcial.

3. 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, esto es consecuencia del paso 2

4. Se seleccionan los primeros bloques para cada servicio.

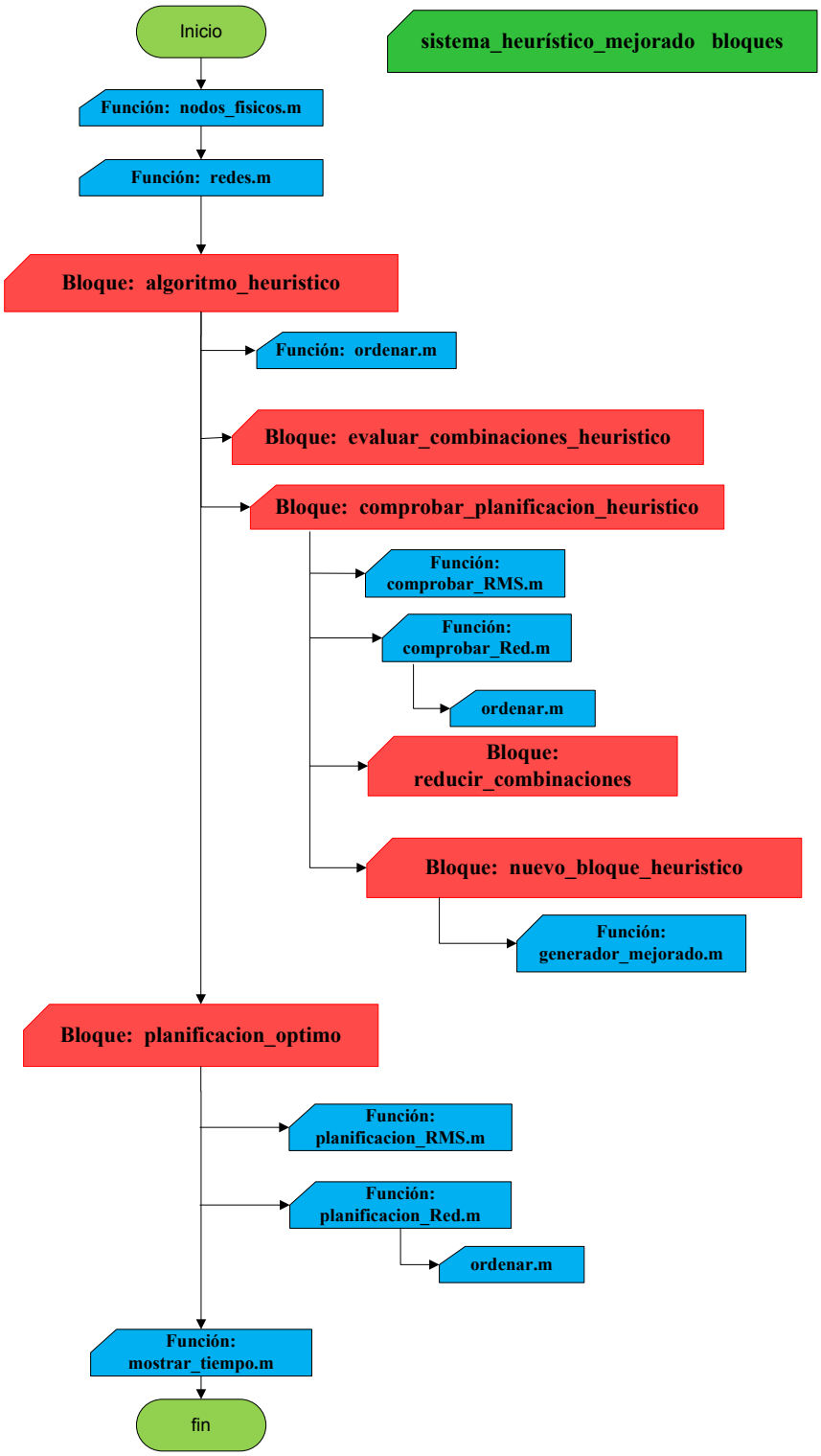

Figura 4.5: Diagrama explicativo de bloques del sistema heurístico mejorado

5. Se procede a generar todas las combinaciones posibles de los bloques seleccionados. Estas se generarán teniendo en cuenta todos servicios de la aplicación y sus posibles perfiles incluidos en los bloques seleccionados.

6. En caso que el número de combinaciones evaluadas sea menor que el umbral máximo de combinaciones a evaluar, se van seleccionando, mediante un bucle, todas las combinaciones antes generadas, que representarán cada uno de los posibles caminos dentro del grafo de la aplicación. En caso de superar el umbral citado ya no se podrá evaluar ninguna otra combinación y por lo tanto se accede al paso 11.

7. Trabajando con bucles se irán evaluando la planificabilidad y tiempos de respuesta de los distintos servicios:

   a) Se evalúa un servicio en serie. Si no existen servicios en paralelo se volvería de nuevo a este paso (Paso 7*a*) hasta haber evaluado todos los servicios de la combinación elegida, tras lo cual se pasará al paso 8. En caso de que no fuese planificable ese servicio se descarta la combinación y se va al paso 7*c*.

   b) Se evalúan todos los servicios en paralelo. En caso de no ser alguno planificable se pasa al paso 7*c*.

   c) En caso que algún nodo no haya podido ser planificable, el sistema mejorado irá eliminando las posibles combinaciones que no van a ser planificables, antes de evaluarlas. Para ello eliminará todas las combinaciones cuya combinación de perfiles hasta el servicio no planificable, sea igual a la combinación ya descartada como no planificable, así como todas las combinaciones que sobre el nodo físico no planificable, desarrollen los mismos perfiles que la combinación ya detectada. Como la combinación que estamos evaluando ya no puede ser planificable, procedemos a evaluar otra combinación (paso 6)

   d) En caso de que algún mensaje haya sido no planificable significará que la red está saturada y no es planificable por lo que será imposible encontrar una combinación planificable ya que no se van a poder enviar los mensajes a través de la red, por lo tanto se accede al paso 11 para la finalización del sistema.

e) Si todos los nodos han sido planificables y han existido paralelos y tienen el punto de sincronización en el próximo nivel, se calculará el camino máximo del paralelo. Si existiesen próximos niveles se retrocede al paso 7*a*.

8. Una vez terminada la evaluación, se puede confirmar que la combinación es planificable en el sistema. En caso de tener la mínima figura de merito para la aplicación, se almacenará la combinación ye la figura de merito.

9. Si no se han terminado de evaluar todas las combinaciones se vuelve al paso 6.

10. Cuando se termine de iterar sobre todos las posibles combinaciones, ya se habrá obtenido, si existe, la mejor combinación de perfiles de servicio respecto a la figura de mérito dada y la nueva planificación del sistema (se avanza al paso 11). En caso de no existir ninguna combinación planificable se procede con el paso 10*a*:

    a) Conociendo el primer servicio en el que no ha sido planificable ninguno de los perfiles del bloque, se procede a elegir el siguiente bloque de ese servicio. Se retrocederá al paso de generación de las nuevas combinaciones (paso 5). En caso de haber agotado todos los bloques se procederá a informar al sistema de la imposibilidad de planificación de la aplicación y a finalizar el mismo.

11. Si se ha conseguido encontrar una combinación planificable, se puede proceder a introducirla en el sistema real. En caso contrario se procederá a informar al sistema de la imposibilidad de planificación de la aplicación y a finalizar el mismo.

Finalmente para concluir se pueden resaltar las siguientes características:

- Como se puede comprobar en estos pasos, el algoritmo será heurístico ya que la solución elegida no asegura que sea la óptima, sino la mejor entre los primeros bloques que sean planificables.
- Cabe destacar asimismo, que en caso de que alguna combinación de algún bloque no sea planificable, existirán gran número de combinaciones en esa generación que sigan el mismo patrón, ya que a la hora de generar las combinaciones se toman solo los perfiles de los bloques seleccionados.

4.3. Conclusiones

En este capítulo se han explicado los pasos seguidos para la implementación de los algoritmos elegidos para la composición de aplicaciones distribuidas de tiempo real. En un primer momento se procedió a la implementación en lenguaje de programación *Matlab*, de los dos algoritmos estudiados en la tesis de Estévez Ayres [1], los cuales se han denominado en este estudio como Algoritmos Exhaustivo y Heurístico. A la hora de la realización se consideró fuera del alcance del presente proyecto el dotar de flexibilidad al sistema, lo cual llevó a que se decidiese que el sistema fuese gobernado exclusivamente por tiempo. Otra de las decisiones que se tomaron de cara a la implementación de las aplicaciones fue que el intercambio de mensajes se realizase en todos los casos a través de una red. Como últimas decisión de diseño, se decidió la manera de tratar los perfiles de cada servicio y la existencia de tareas en estos perfiles de tipo consumidor, productor y consumidor/productor. Asimismo se decidió la creación de tres *deadlines*: el *deadline* de nodo para los servicios, el *deadline* de red para los mensajes y el *deadline* de aplicación.

Para la realización de los algoritmos se decidió establecer prioridades a las tareas para una mejor planificación de las mismas. La asignación de prioridades se realiza de forma estática en tiempo de ejecución y usando el método de asignación RMS (*Rate Monotonic Scheduling*).

Durante la implementación del Algoritmo Exhaustivo, se decidió la introducción de mejoras en el diseño que llevaron a que este realice las comprobaciones de la planificabilidad simulando el sistema, sin necesidad de estar modificándolo en cada evaluación. Una vez ya elegida la combinación óptima, ya se puede ejecutar en el sistema. Otra de las mejoras es que el algoritmo en cualquier momento de la comprobación si un perfil no es planificable, descarta ya toda la combinación, sin necesidad de evaluar el resto de los perfiles de esa misma combinación. Asimismo se siguió manteniendo debido a su importancia, el hecho que la no planificación de un mensaje haga que la combinación que se esté evaluando sea descartada, así como la obligación que el tiempo de respuesta de la combinación sea menor que el plazo deseado por el usuario para la aplicación, el *deadline* de la aplicación.

Respecto al diseño del Algoritmo Heurístico, se decidió modificar la selección de los bloques y la evaluación de las combinaciones. Para ello al inicio del mismo se seleccionan

por cada servicio un bloque con un número finito de perfiles, con los cuales se procede a generar todas las combinaciones posibles. Posteriormente se comprueba su planificabilidad, y en caso de no encontrar combinación alguna, se selecciona otro nuevo bloque del servicio que haya resultado no planificable y se generan las nuevas combinaciones a evaluar. Para delimitar el tiempo de ejecución del algoritmo se decidió establecer un límite de combinaciones máximas a evaluar, el cual se estableció en un cuarto. La principal característica de este algoritmo es que no asegura la obtención de la combinación óptima.

Una vez implementados estos algoritmos se decidió realizar una serie de mejoras sobre ellos que nos permitiesen una optimización de los mismos cuando los sistemas estuvieran más cargados. A partir de esta premisa surge el Algoritmo Mejorado. Este algoritmo se basó en el diseño del exhaustivo pero añade la nueva característica de eliminar los patrones que se sabe de antemano que van a ser no planificables. Al igual que el exhaustivo seguirá asegurando la obtención de la combinación óptima.

Finalmente se decidió implementar la característica de eliminación de patrones sobre el Algoritmo Heurístico, obteniéndose así el Algoritmo Heurístico Mejorado.

Capítulo 5

Programa desarrollado

En este capítulo se va a explicar función a función la implementación que se ha realizado para cada una de ellas. Esta explicación será exhaustiva, ya que se persigue que sirva como complemento y manual de usuario ante una persona que se quiera enfrentar a la modificación y evaluación del código desarrollado. Se van a ir explicando en primer lugar las funciones de los cuatro sistemas desarrollados para en una segunda parte explicar el resto de las funciones necesarias para el funcionamiento correcto de los sistemas. Cada una de las functions va a estar acompañada de sus respectivos diagramas de bloques para facilitar la comprensión de las mismas.

5.1. Sistemas

En esta primera parte se va a proceder a explicar cada uno de los sistemas que se han desarrollado de forma minimalista. Asimismo se le da especial atención al desarrollo de los algoritmos, detallándolos paso por paso para una mayor comprensión de los mismos. Estos sistemas ya fueron introducidos y tratados de una forma mucho más básica en el capítulo 4 (véase apartado 4.2).

5.1.1. Sistema genérico

5.1.1.1. Introducción

Función principal del sistema. Desde esta función se coordinan y dirigen todas las operaciones que se han de realizar para poder comprobar si el sistema que el cliente desea implementar es planificable dentro de los nodos y redes existentes. Esta función evaluará la arquitectura pasada por el cliente, generando y evaluando las posibles combinaciones de las operaciones que se quieren realizar y estudiando su planificabilidad así como su mínima figura de mérito global para su realización. En principio se ha diseñado para que la figura de mérito global sea el tiempo mínimo de respuesta, aunque como se explicó en el capítulo anterior, ésta podría ser modificada.

Como se introdujo en el apartado 4.2.2 esta función genérica, ejecutará el algoritmo exhaustivo, el cual evalúa todas las posibles combinaciones de las distintas operaciones en los nodos, obteniendo para cada una de ellas unos valores de planificabilidad y de tiempo de respuesta.

5.1.1.2. Parámetros de entrada y de salida

Para poder ejecutar el sistema será necesario que la función reciba por parámetro un fichero adaptado a las especificaciones impuestas y que será la arquitectura de operaciones, nodos y redes que se quiere evaluar (las características de este fichero se pueden encontrar en el apartado 5.3). Este fichero ha de ser introducido entre comillas simples. Un ejemplo de ejecución sería el siguiente:

```
>> sistema_generico ('Aplicacion_simple')
```

En relación a los parámetros de salida, será el propio usuario el que decida que parámetros recibir. Parece lógico tener como parámetro de salida el mejor tiempo de respuesta. A continuación se muestran los parámetros que se han utilizado como salida a la hora de realizar el diseño y las comprobaciones del sistema:

- **t_resp_min:** En la comprobación de planificabilidad, tiempo de respuesta mínimo del sistema.

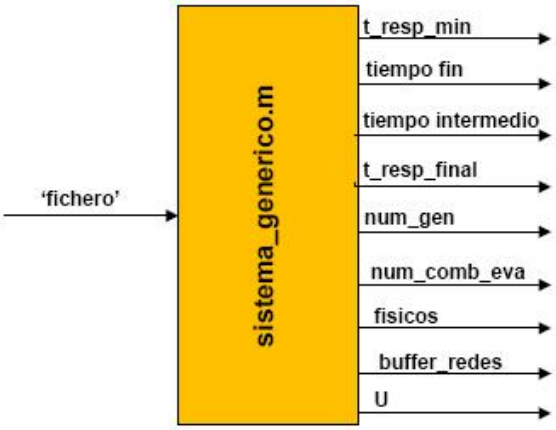

Figura 5.1: Parámetros de entrada/salida de la función *sistema_genrico.m*

- **tiempo_fin:** Devuelve el tiempo en segundos que ha tardado el sistema en ejecuatarse.

- **tiempo_intermedio:** Devuelve el tiempo en segundos que ha tardado el sistema en comprobar la planificación.

- **t_resp_final:** Una vez elegida la combinación a introducir en los nodos y las redes, tiempo de respuesta del sistema. Ha de coincidir con el tiempo de respuesta mínimo de la comprobación.

- **num_gen:** Número de combinaciones que tiene la tabla del generador una vez comprobadas todas las necesarias.

- **num_comb_eva:** Número de combinaciones evaluadas por el sistema en cada ejecución para poder definir su planificabilidad.

- **físicos:** Matrices que contienen los nodos físicos y las tareas en ellos incluidos.

- **buffer_redes:** Matrices que contienen los mensajes transmitidos por cada red física.

- **U:** Utilidades de los nodos físicos.

5.1.1.3. Explicación detallada

Basándonos en los diagramas de flujo, vamos a explicar detalladamente cómo está programado el sistema genérico. En primer lugar, nos apoyaremos en la figura 5.2:

1. La primera acción a realizar una vez ejecutado el sistema es cargar las definiciones de las distintas variables globales. Posteriormente se inicializarán las variables del sistema.

2. Se procede a hallar el número de niveles, nodos en serie, que tiene la aplicación, así como calcular el número de nodos en paralelo para cada nivel. Asimismo teniendo en cuenta ambos se calcula el número total de nodos que tendrá la aplicación.

3. Se hallan el número de posibles perfiles que habrá para cada operación/servicio.

4. Se creara un vector *long_gen* que contendrá las longitudes de las distintas operaciones del sistema.

5. Unido al paso anterior, y usando su vector, se procede a generar todas las combinaciones posibles de la aplicación. Para ello se llama a la función *generador.m* explicada en 5.2.4.

6. Una vez obtenidas todas las posibles combinaciones procederemos a evaluarlas. Este proceso queda reflejado en la figura 5.3.

7. Bucle while irá comprobando combinación a combinación, de las obtenidas en la generación, cuáles de ellas son planificables y en caso de serlo, cual es la que nos da menor tiempo de respuesta. En caso no de existir más combinaciones se va al paso 22.

8. Al evaluar una nueva combinación procedemos a aumentar el contador de combinaciones evaluadas. Asimismo se inicializan las distintas variables de tiempos y de planificación.

9. Se resetean los búferes auxiliares en los cuales se simula el sistema.

10. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en serie.

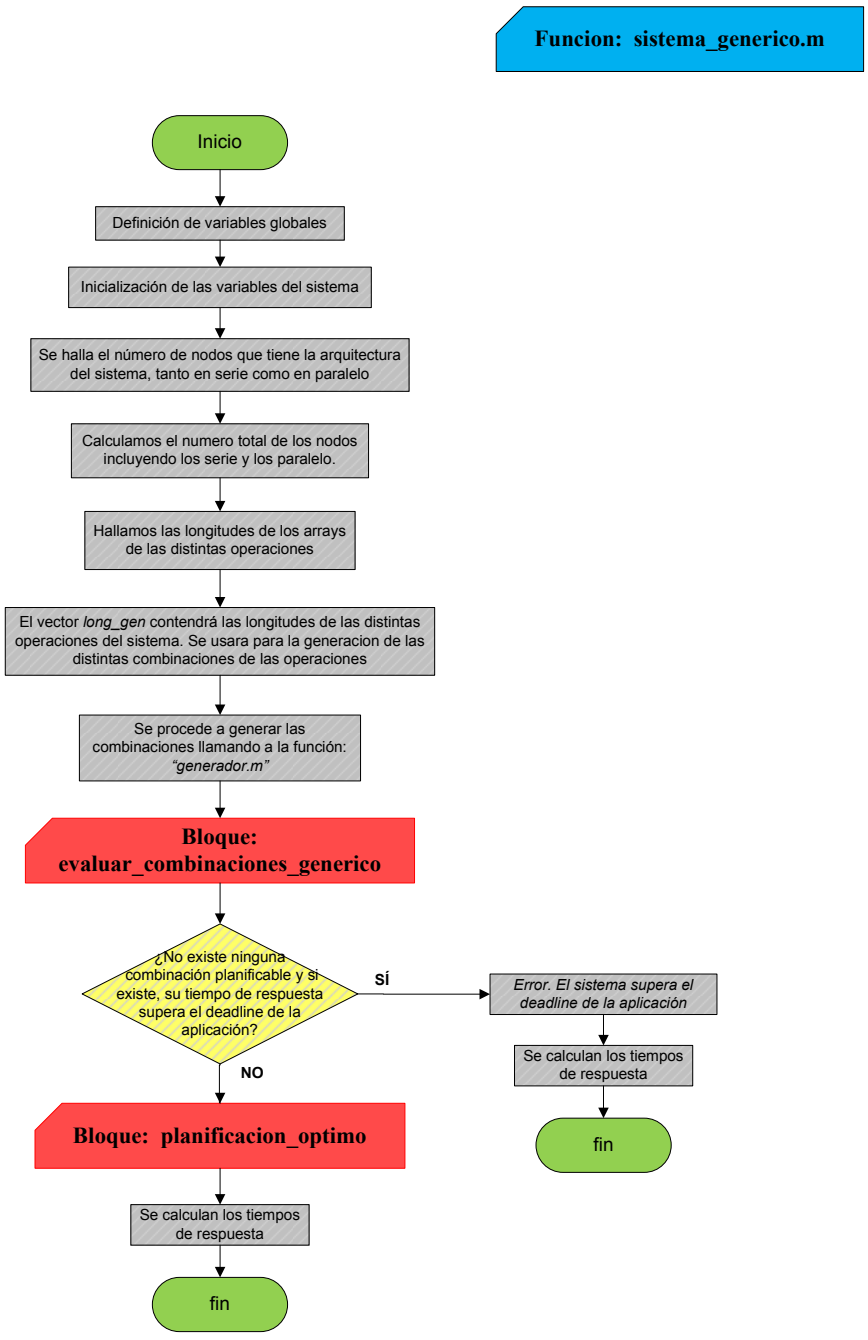

Figura 5.2: Diagrama bloques de la *function sistema_genrico.m*

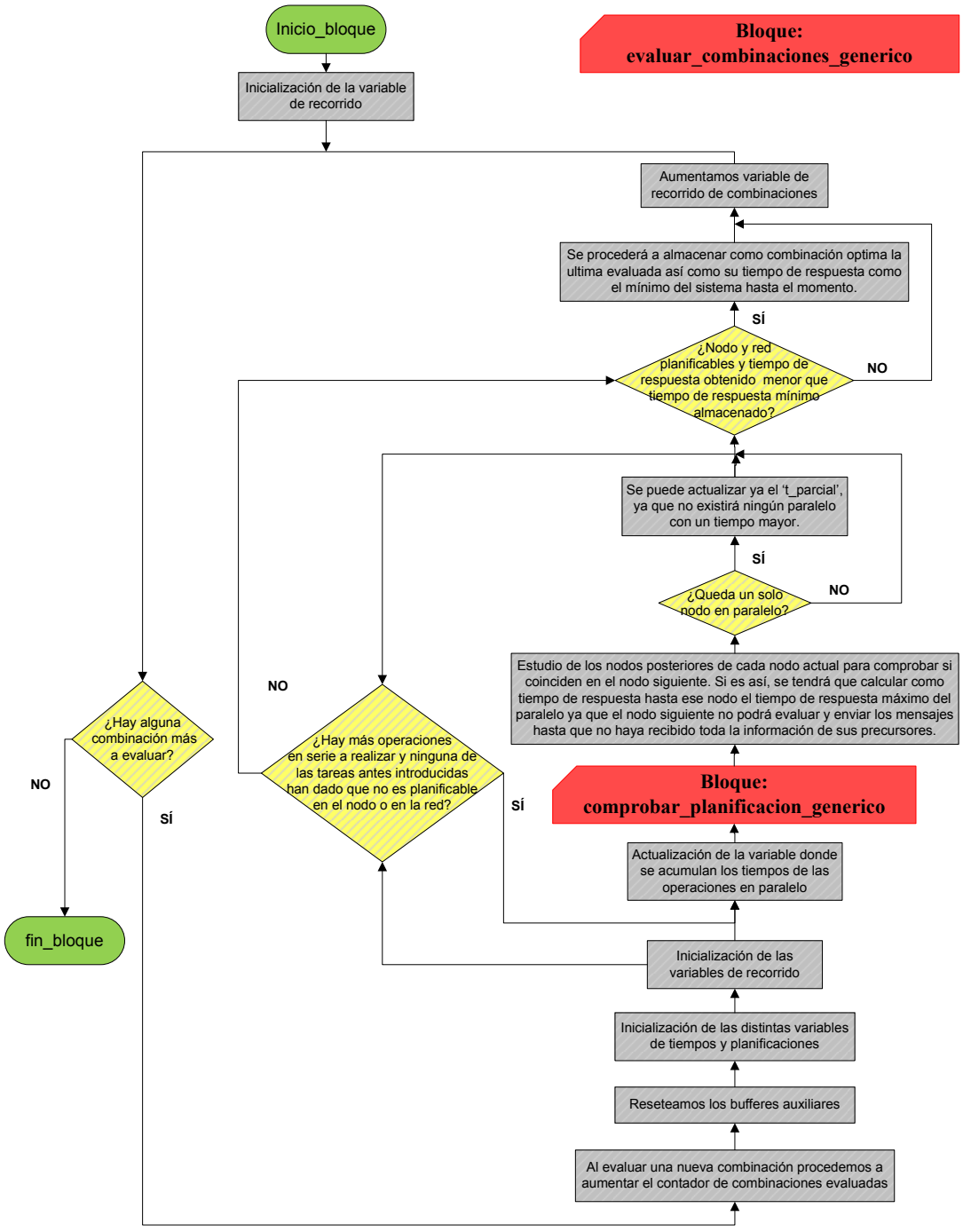

Figura 5.3: Diagrama bloques del bloque *evaluar_combinaciones_generico*

11. Bucle *while* que mientras haya operaciones en serie a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable en el nodo o en la red, seguirá evaluando la planificabilidad de la combinación. Si fallasen las comprobaciones se va al paso 20.

12. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en paralelo. Nos encontraremos ahora en los diagramas de la figura 5.4.

13. Bucle *while* que mientras haya operaciones en paralelo a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable, en el nodo o en la red, seguirá evaluando la planificabilidad del paralelo. Si fallasen las comprobaciones, paso 19. Nota: Hay que resaltar que un solo nodo en un nivel corresponde a una operación en paralelo. En caso de existir dos operaciones en un nivel, habrá dos operaciones en paralelo, y así sucesivamente.

14. En la combinación que se está evaluando se guarda la posición del nodo que estamos comprobando.

15. Teniendo en cuenta la operación que desarrolle el nodo a evaluar, se procede a comprobar la planificabilidad y el tiempo de respuesta de esa tarea sobre el sistema. Para ello se llama a la función comprobar_RMS (véase 5.2.6).

16. Se evalúa si estamos trabajando con el último nodo de la aplicación, en cuyo caso el tiempo de respuesta total de la operación no hace falta adaptarlo al tiempo LSW. Se pasa al paso 18.

17. Teniendo en cuenta la red sobre la que ha de mandar el mensaje el nodo evaluado, se procede a comprobar la planificabilidad y el tiempo de respuesta de ese mensaje. Para ello se llama a la función comprobar_Red (véase 5.2.7). Se calcula posteriormente el tiempo total de la operación como el tiempo de respuesta de la tarea, adaptado a tiempos LSW, más el tiempo de respuesta del mensaje, que ya está adaptado a tiempos LSW.

18. Actualizamos los tiempos totales del paralelo. Se vuelve al paso 13.

19. Se realiza un estudio de los nodos posteriores de cada nodo actual para comprobar si coinciden en el nodo siguiente. Si es así, se tendrá que calcular como tiempo de

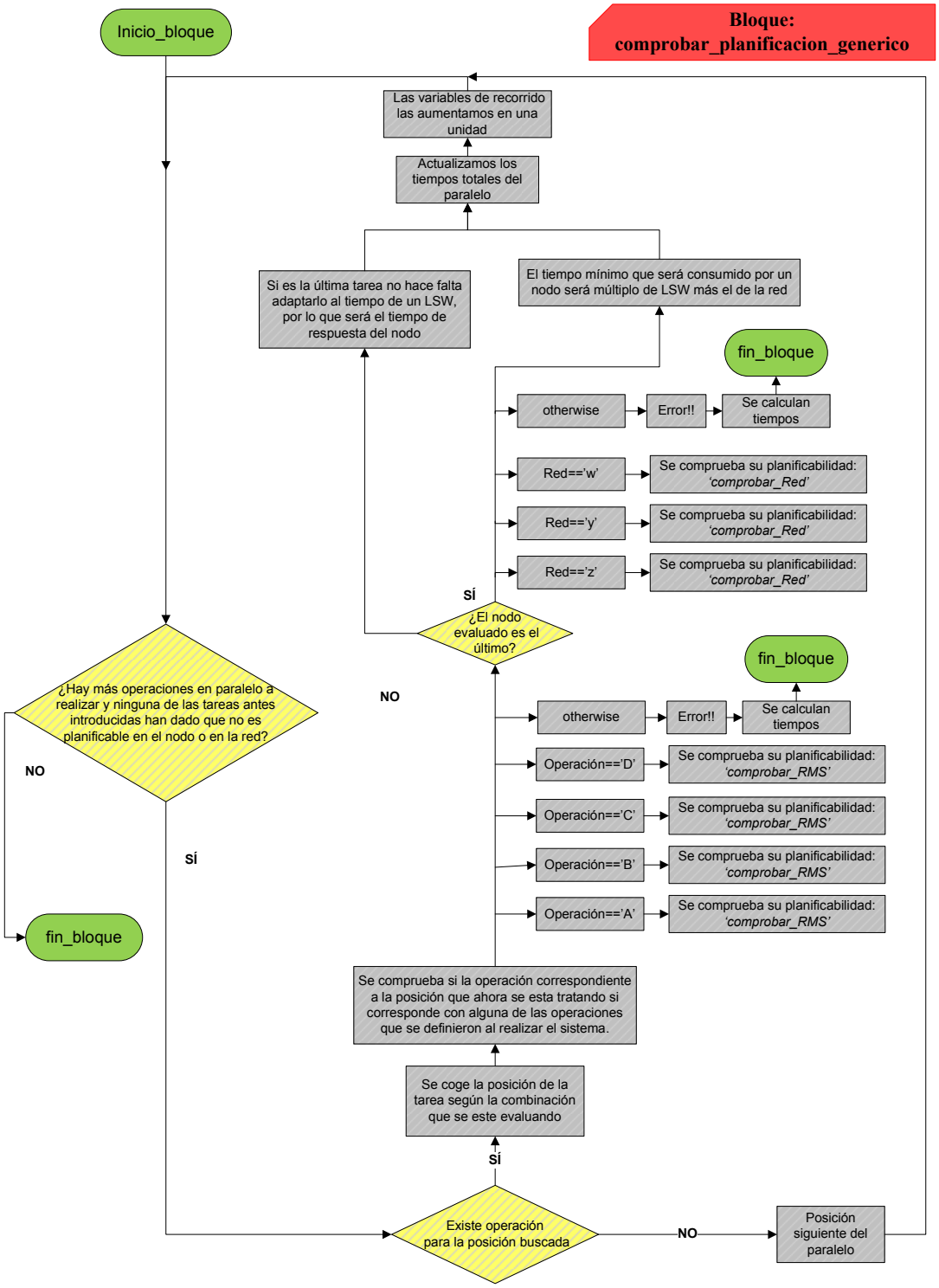

Figura 5.4: Diagrama bloques del bloque *comprobar_planificacion_generico*

respuesta hasta ese nodo el tiempo de respuesta máximo del paralelo, ya que el nodo siguiente no podrá evaluar y enviar los mensajes hasta que no haya recibido toda la información de sus precursores. Se vuelve al paso 11.

20. Se habrá terminado de evaluar la combinación y si todo ha sido planificable se procederá a comprobar si el tiempo de respuesta obtenido es menor que tiempo de respuesta mínimo ya almacenado. En caso negativo se pasara al paso 7.

21. Se procederá a almacenar como combinación óptima la última evaluada así como su tiempo de respuesta como el mínimo del sistema hasta el momento, y seguimos en el paso 7.

22. Ya no existirá ninguna combinación más a evaluar por lo que se procede a comprobar si existe alguna combinación planificable entre las anteriores, y en caso afirmativo si el tiempo de respuesta obtenido en la evaluación no supera el *deadline* de la aplicación. En caso contrario se reportará un error y se finalizará el sistema.

    Nos encontraremos ahora en los diagrama de la figura 5.5.

23. Se calcula en segundos el tiempo que ha tardado el sistema en comprobar todas las combinaciones (tiempo intermedio).

24. Bucle *while* que mientras haya operaciones en serie seguirá evaluando el tiempo de respuesta de la combinación óptima. Si no hubiese más servicios a evaluar va al paso 32.

25. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en paralelo.

26. Bucle *while* que mientras haya operaciones en paralelo seguirá evaluando los nodos en paralelo. Si no existiese más nodos en paralelo, paso 31.

27. Teniendo en cuenta la operación que desarrolle el nodo a evaluar, se procede a insertar el perfil en el nodo físico correspondiente y calcular el tiempo de respuesta de esa tarea sobre el sistema. Para ello se llama a la función planificacion_RMS (véase 5.2.8).

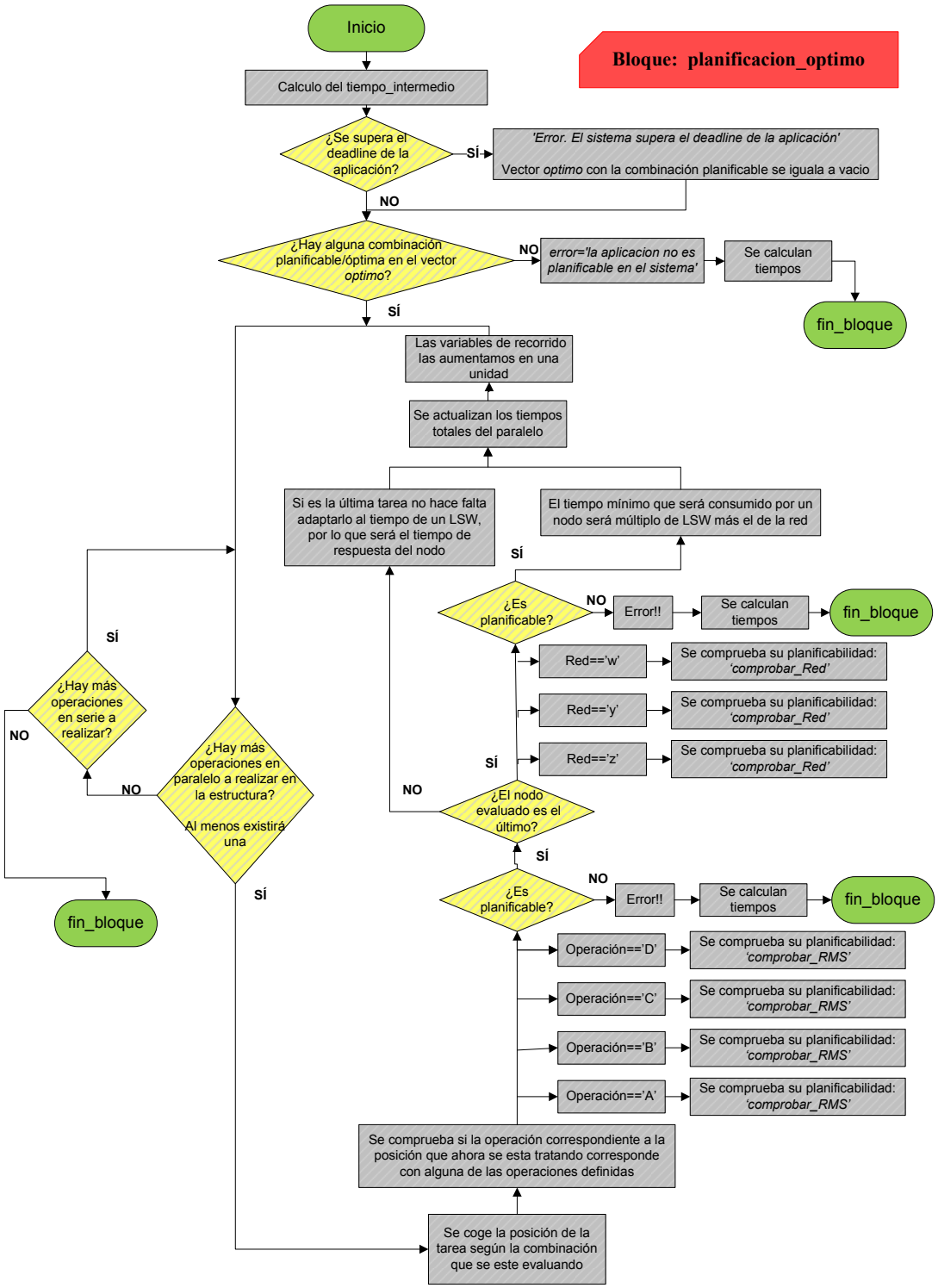

Figura 5.5: Diagrama bloques del bloque *planificacion_optimo*

28. Se evalúa si estamos trabajando con el último nodo de la aplicación, en cuyo caso el tiempo de respuesta total de la operación no hace falta adaptarlo al tiempo LSW. Se pasa al paso 30.

29. Teniendo en cuenta la red sobre la que ha de mandar el mensaje el nodo evaluado, se procede a insertar el mensaje sobre el nodo físico correspondiente y a calcular el tiempo de respuesta de ese mensaje. Para ello se llama a la función planificacion_Red (véase 5.2.9). Se calcula posteriormente el tiempo total de la operación como el tiempo de respuesta de la tarea, adaptado a tiempos LSW, más el tiempo de respuesta del mensaje, que ya está adaptado a tiempos LSW.

30. Actualizamos los tiempos totales del paralelo. Se vuelve al paso 26.

31. Se realiza un estudio de los nodos posteriores de cada nodo actual para comprobar si coinciden en el nodo siguiente. Si es así, se tendrá que calcular como tiempo de respuesta hasta ese nodo el tiempo de respuesta máximo del paralelo, ya que el nodo siguiente no podrá evaluar y enviar los mensajes hasta que no haya recibido toda la información de sus precursores. Se vuelve al paso 24.

32. Se habrá terminado de evaluar la combinación por lo que ya tenemos el tiempo de respuesta final.

33. Calculamos el tiempo que ha tardado en ejecutar todo el sistema.

5.1.2. Sistema heurístico

5.1.2.1. Introducción

Función creada a partir del sistema genérico. Esta función coordinará y dirigirá todas las operaciones que se han de realizar para poder comprobar si el sistema que el cliente desea implementar es planificable dentro de los nodos y redes existentes. A partir de la arquitectura pasada por el cliente, estudiará la planificabilidad de las combinaciones necesarias, y hallará la mínima figura de mérito global, para su realización. Se ha diseñado para que esta figura de mérito global sea el tiempo mínimo de respuesta. La diferencia principal con respecto al sistema genérico, explicado en el apartado 5.1.1, es que este sistema

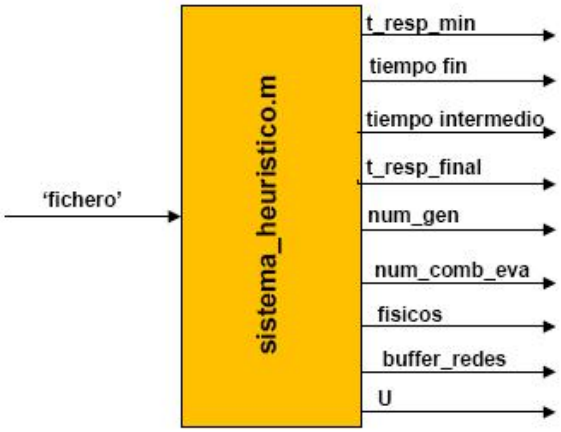

Figura 5.6: Parámetros de entrada/salida de la función *sistema_heuristico.m*

hace uso del algoritmo heurístico explicado en el apartado 4.2.3. Como se explicó en ese apartado, un sistema heurístico por definición, [82], "*es un procedimiento de búsqueda de soluciones casi óptimas a un coste computacional razonable, sin ser capaz de garantizar la factibilidad de las soluciones empleadas ni determinar a qué distancia de la solución óptima nos encontramos*". Por lo tanto la diferencia principal con respecto al sistema genérico radicará en la creación y evaluación de las combinaciones a evaluar.

5.1.2.2. Parámetros de entrada y de salida

Para poder ejecutar el sistema será necesario que la función reciba por parámetro un fichero adaptado a las especificaciones impuestas y que será la arquitectura de operaciones, nodos y redes que se quiere evaluar (las características de este fichero se pueden encontrar en el apartado 5.3). Este fichero ha de ser introducido entre comillas simples. Un ejemplo de ejecución sería el siguiente:

```
>> sistema_heuristico ('Aplicacion')
```

En relación a los parámetros de salida, será el propio usuario el que decida que parámetros recibir. A continuación se muestran los parámetros que se han utilizado como salida a la hora de realizar el diseño y las comprobaciones del sistema:

- **t_resp_min:** En la comprobación de planificabilidad, tiempo de respuesta mínimo del sistema.
- **tiempo_fin:** Devuelve el tiempo en segundos que ha tardado el sistema en ejecutarse.
- **tiempo_intermedio:** Devuelve el tiempo en segundos que ha tardado el sistema en comprobar la planificación.
- **t_resp_final:** Una vez elegida la combinación a introducir en los nodos y las redes, tiempo de respuesta del sistema. Ha de coincidir con el tiempo de respuesta mínimo de la comprobación.
- **num_gen:** Número de combinaciones que tiene la tabla del generador una vez comprobadas todas las necesarias.
- **num_comb_eva:** Número de combinaciones evaluadas por el sistema en cada ejecución para poder definir su planificabilidad.
- **físicos:** Matrices que contienen los nodos físicos y las tareas en ellos incluidos.
- **buffer_redes:** Matrices que contienen los mensajes transmitidos por cada red física.
- **U:** Utilidades de los nodos físicos.

5.1.2.3. Explicación detallada

Basándonos en los diagramas de flujo, vamos a explicar detalladamente cómo está programado el sistema heurístico. En primer lugar, nos apoyaremos en la figura 5.7:

Nota: Aunque gran cantidad de los pasos seguidos por el sistema son idénticos a los realizados por el sistema genérico, vamos a volver a reproducirlos, ya que se persigue que este capítulo funcione como un manual de usuario a la hora de enfrentarse al código en Matlab que se ha programado.

1. La primera acción a realizar una vez ejecutado el sistema es cargar las definiciones de las distintas variables globales. Posteriormente se inicializarán las variables del sistema.

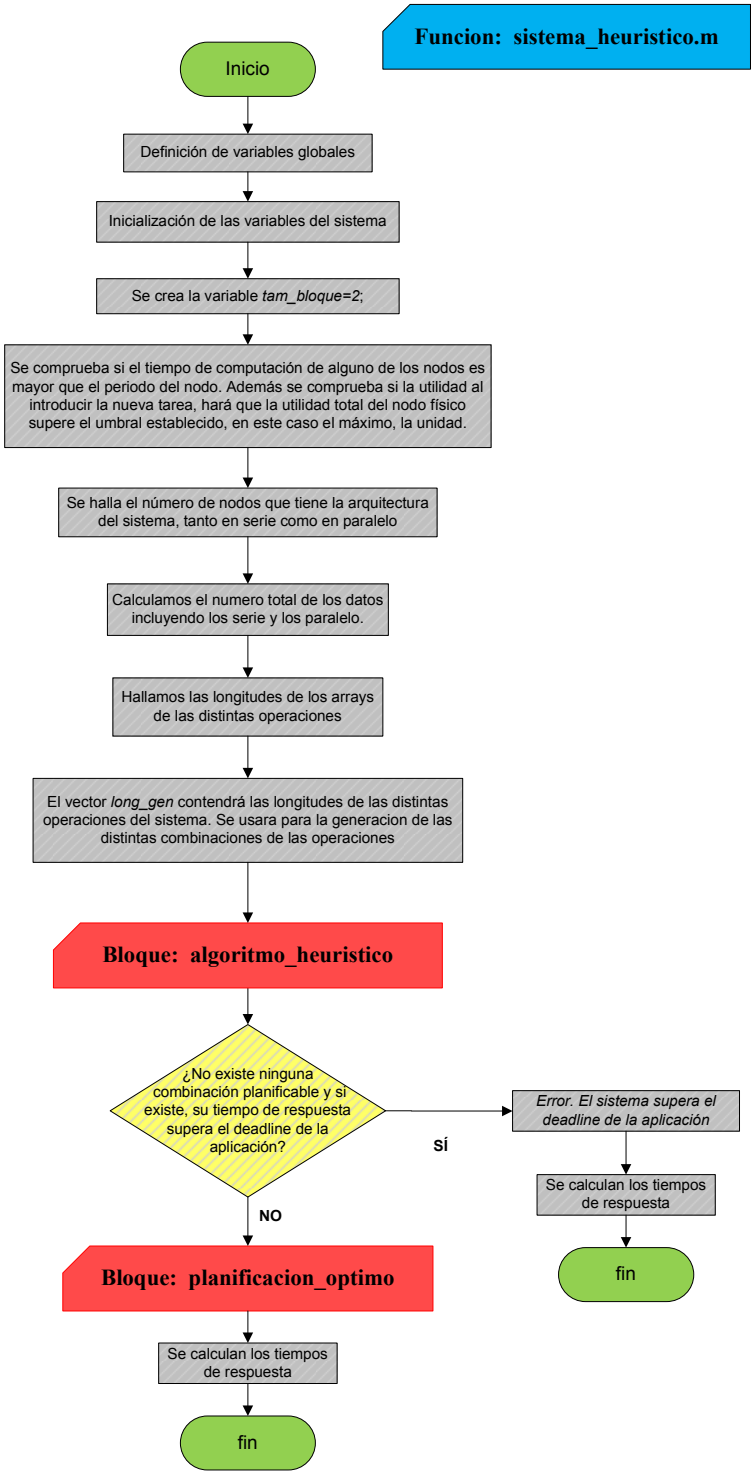

Figura 5.7: Diagrama bloques de la *function sistema_heuristico.m*

2. Se define el tamaño de bloque a utilizar en el algoritmo heurístico. Este tamaño de bloque limitará las combinaciones que se evalúen en el sistema. En nuestro caso se decidió que `tam_bloque=2`.

3. Se procede a hallar el número de niveles, nodos en serie, que tiene la aplicación, así como calcular el número de nodos en paralelo para cada nivel. Asimismo teniendo en cuenta ambos se calculan el número total de nodos que tendrá la aplicación.

4. Se hallan el número de posibles perfiles que habrá para cada operación/servicio.

5. Se creará un vector *long_gen* que contendrá las longitudes de las distintas operaciones del sistema.

6. Antes de realizar la generación de las posibles combinaciones procedemos a ordenar las tareas por su figura de mérito relativa, es decir un criterio parcial que en nuestro caso será el tiempo de computación de las mismas (C_i). Esta parte corresponderá al algoritmo heurístico propiamente, por lo que queda reflejada sobre el diagrama de bloques de la figura 5.8.

7. Se almacena en una variable el tamaño máximo de las tareas de cada nodo ($long\_gen\_max$), que nos será necesario para poder conocer el número máximo de tamaños de bloques que vamos a poder evaluar.

8. Se modifica la variable *long_gen* adaptándola a las longitudes máximas de las tareas de cada uno de los nodos para la creación de combinaciones, es decir, se adapta al tamaño que se ha seleccionado para cada bloque.

9. Se comprueba que no sobrepase el tamaño máximo de algún nodo, comparándolo con la variable antes establecida, $long\_gen\_max$.

10. Se establece el umbral máximo de combinaciones a evaluar como un cuarto del número máximo de combinaciones posibles. Este umbral se decidió, después de realizar prueba con diferentes cotas máximas. Lo que se pretende es que en aplicaciones con gran número de servicios no se evalúen un número desorbitado de casos, que harán que el sistema sea ineficiente para trabajar en tiempo real debido a su enorme tiempo de ejecución. Asimismo hay q permitir que en aplicaciones con pocos servicios, el

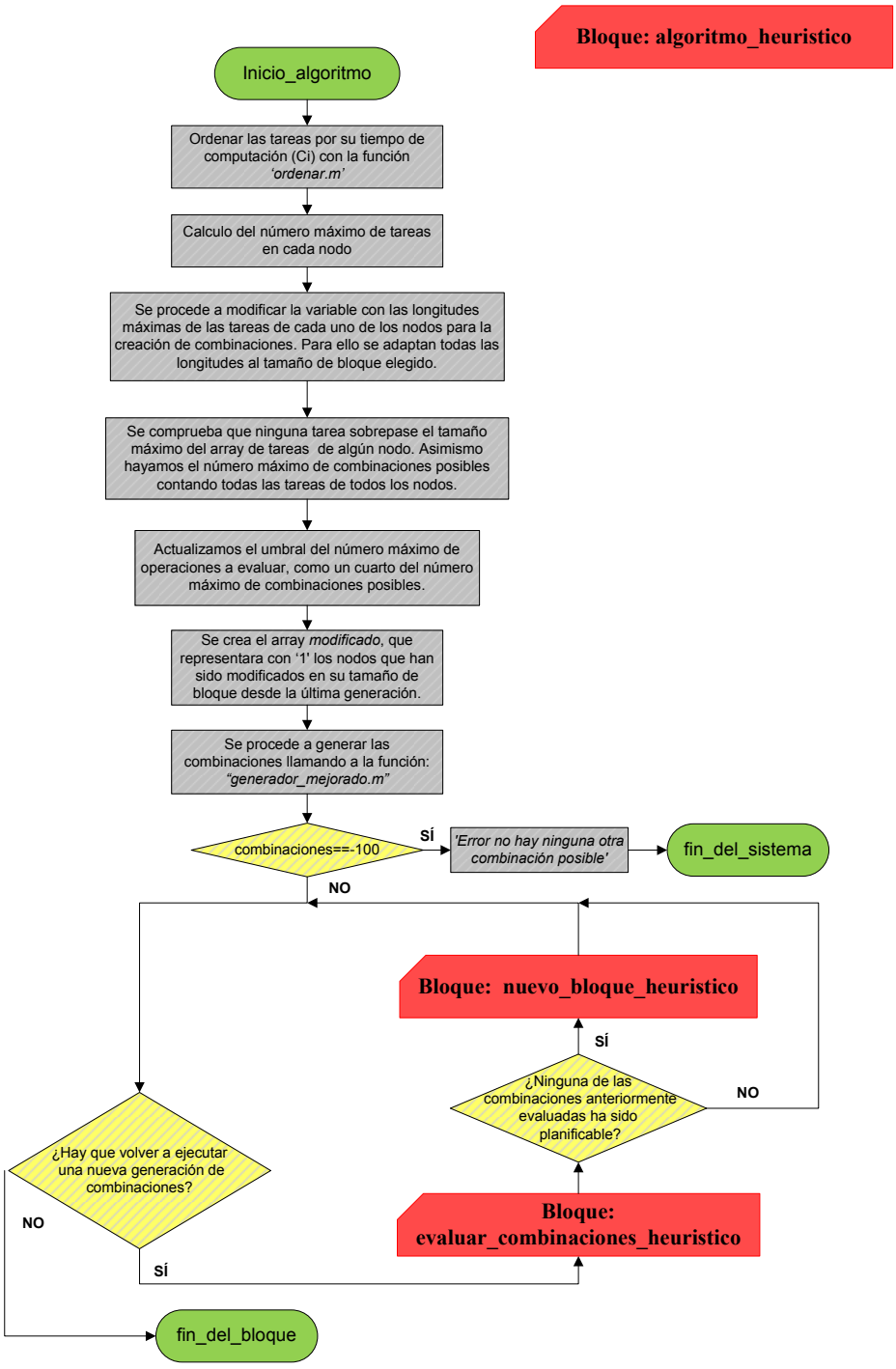

Figura 5.8: Diagrama bloques del bloque *algoritmo_heuristico*

número de evaluaciones sea adecuado. Con cotas menores, el número de evaluaciones era insuficiente.

11. Se crea la variable *ultimo_long* para almacenar los tamaños de los nodos comprobados anteriormente.

12. Se crea el *array* modificado, que incluirá los nodos que han sido modificados en su tamaño de bloque desde la última generación (representados por '1') y los que no han sufrido modificación alguna (representados por '0').

13. Una vez realizados todos los pasos anteriores, se procede a generar todas las combinaciones posibles teniendo en cuenta el tamaño de los bloques. Para ello se llama a la función *generador_mejorado.m* explicada en 5.2.5.

14. Obtenidas las posibles combinaciones procederemos a evaluarlas. Para ello será necesario la creación de un bucle *while* que comprobará si hay que volver a ejecutar una nueva generación de combinaciones. Este bucle facilitará nuevas entradas a la ejecución del algoritmo en caso de que con la evaluación de los primeros bloques, el algoritmo no haya sido capaz de encontrar una combinación que sea planificable. En caso de no ser necesario nuevas ejecuciones (en caso que el flag *begin_again* este desactivado) se procederá a introducir la combinación más favorable, si existiese: Paso 38.

    A partir de este paso nos encontramos en la figura 5.9.

15. Bucle *while* irá comprobando combinación a combinación, de las obtenidas en la generación, cuáles de ellas son planificables y en caso de serlo, cual es la que nos da menor tiempo de respuesta. Asimismo comprueba que el número de combinaciones evaluadas no supera el umbral máximo elegido. En caso no de existir más combinaciones a evaluar o superar ese umbral, se va al paso 30

16. Al evaluar una nueva combinación procedemos a aumentar el contador de combinaciones evaluadas. Asimismo se inicializan las distintas variables de tiempos y de planificación.

17. Se resetean los búferes auxiliares en los cuales se simula el sistema.

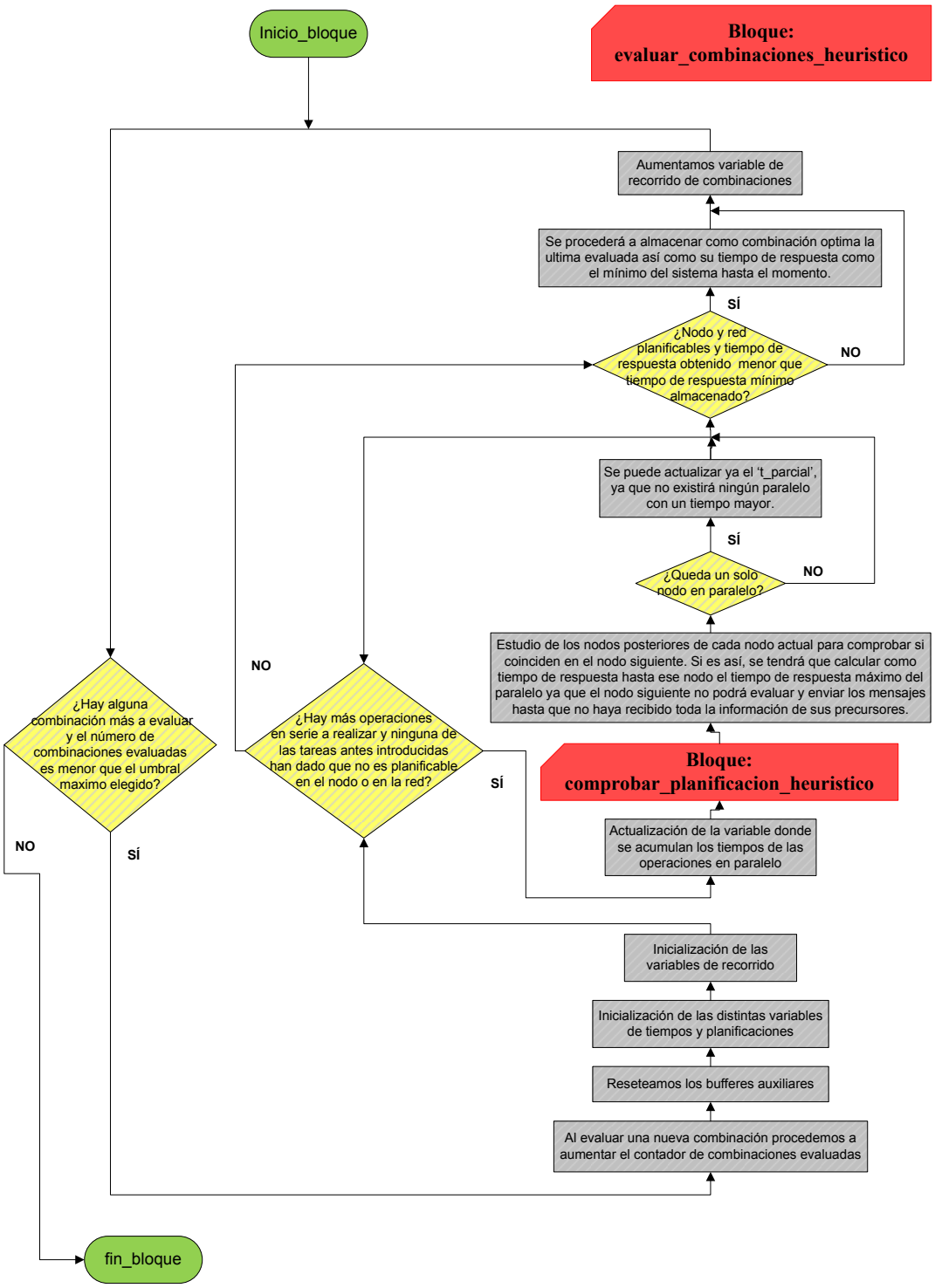

Figura 5.9: Diagrama bloques del bloque *evaluar_combinaciones_heuristico*

18. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en serie.

19. Bucle *while* que mientras haya operaciones en serie a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable en el nodo o en la red, seguirá evaluando la planificabilidad de la combinación. Si fallasen las comprobaciones se va al paso 28.

20. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en paralelo. Nos encontraremos ahora en los diagramas de la figura 5.10.

21. Bucle *while* que mientras haya operaciones en paralelo a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable, en el nodo o en la red, seguirá evaluando la planificabilidad del paralelo. Si fallasen las comprobaciones, se accede al paso 27.

    Nota: Hay que resaltar que un solo nodo en un nivel corresponde a una operación en paralelo. En caso de existir dos operaciones en un nivel, habrá dos operaciones en paralelo, y así sucesivamente.

22. En la combinación que se está evaluando se guarda la posición del nodo que estamos comprobando.

23. Teniendo en cuenta la operación que desarrolle el nodo a evaluar, se procede a comprobar la planificabilidad y el tiempo de respuesta de esa tarea sobre el sistema. Para ello se llama a la función comprobar_RMS (véase 5.2.6).

24. Se evalúa si estamos trabajando con el último nodo de la aplicación, en cuyo caso el tiempo de respuesta total de la operación no hace falta adaptarlo al tiempo LSW. Se pasa al paso 26.

25. Teniendo en cuenta la red sobre la que ha de mandar el mensaje el nodo evaluado, se procede a comprobar la planificabilidad y el tiempo de respuesta de ese mensaje. Para ello se llama a la función comprobar_Red (véase 5.2.7). Se calcula posteriormente el tiempo total de la operación como el tiempo de respuesta de la tarea, adaptado a tiempos LSW, más el tiempo de respuesta del mensaje, que ya está adaptado a tiempos LSW.

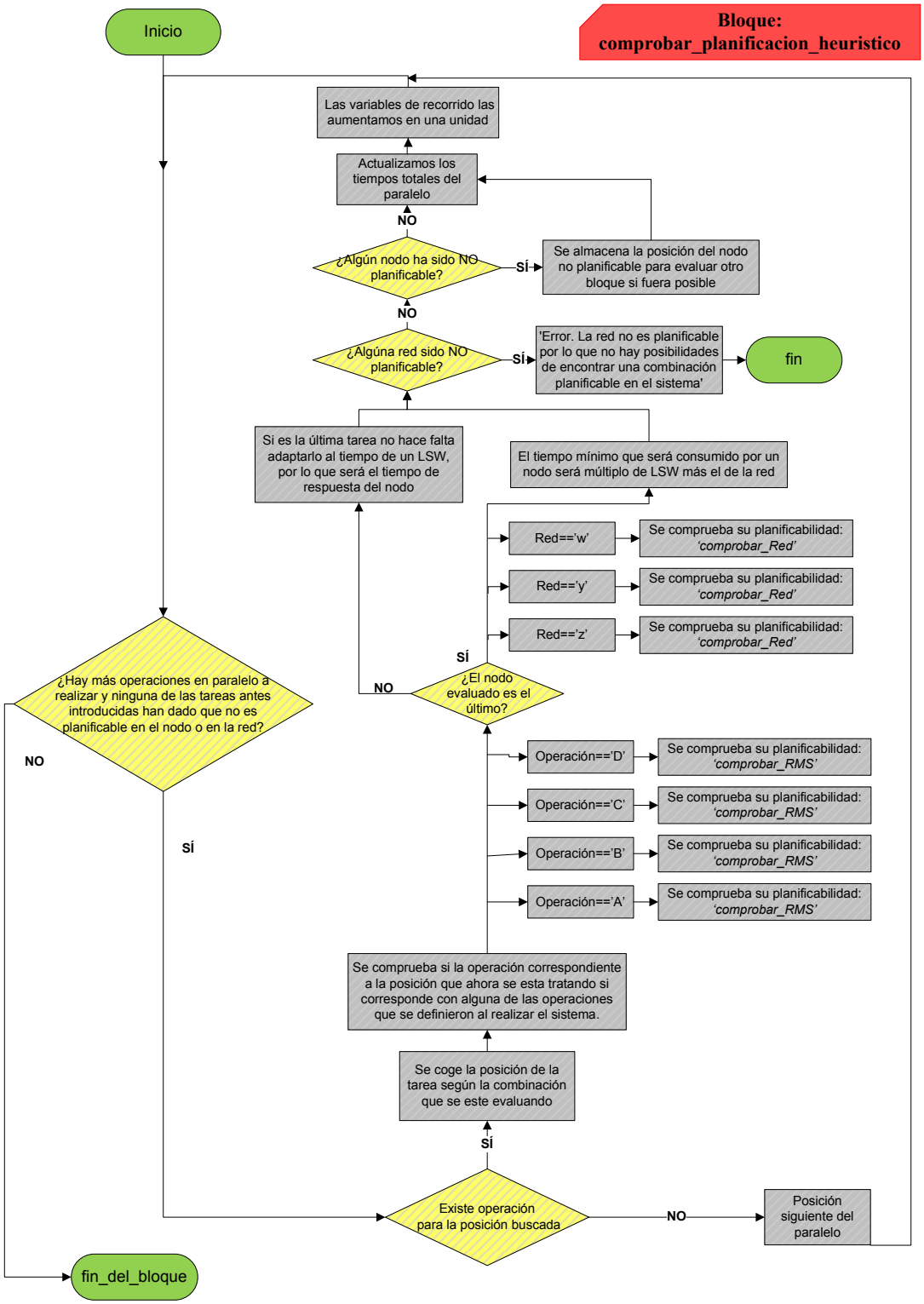

Figura 5.10: Diagrama bloques del bloque *comprobar_planificacion_heuristico*

26. Actualizamos los tiempos totales del paralelo. Se vuelve al paso 21.

27. Se realiza un estudio de los nodos posteriores de cada nodo actual para comprobar si coinciden en el nodo siguiente. Si es así, se tendrá que calcular como tiempo de respuesta hasta ese nodo el tiempo de respuesta máximo del paralelo, ya que el nodo siguiente no podrá evaluar y enviar los mensajes hasta que no haya recibido toda la información de sus precursores. Se vuelve al paso 19.

28. Se habrá terminado de evaluar la combinación y si todo ha sido planificable se procederá a comprobar si el tiempo de respuesta obtenido es menor que el tiempo de respuesta mínimo ya almacenado. En caso negativo se pasara al paso 15.

29. Se procederá a almacenar como combinación óptima la ultima evaluada así como su tiempo de respuesta como el mínimo del sistema hasta el momento, y seguimos en el paso 15.

30. Ya no existirá ninguna combinación más a evaluar, por lo que se comprueba si hay alguna que haya sido planificable. Si hay al menos una planificable se accederá a introducirla en los nodos y redes físicas (paso 38). En caso de no existir ninguna se accede al paso 31.

31. Esta parte del algoritmo está reflejada en la figura 5.11. Una vez terminado de evaluar todas las combinaciones obtenidas con la generación para los tamaños de bloques seleccionados, al no haberse encontrado ninguna combinación planificable, será necesario modificar los perfiles a evaluar del servicio donde primero haya fallado la evaluación.

32. En primer lugar se comprueba si ya se ha superado el umbral de combinaciones máximas a evaluar. En caso de ser así ya no se podrán evaluar más combinaciones, por lo que no será necesario volver a generar más combinaciones con nuevos perfiles. Debido a esto se fuerza al bucle a terminar y abandonar la evaluación, accediendo al sistema de planificación de la combinación optima. Para que esto suceda se desactiva el flag ($begin\_again$) y se vuelve al paso 14.

33. Si se pueden evaluar nuevas combinaciones, procedemos a modificar la variable $long\_gen$ del servicio donde ha fallado en primer lugar la planificación, para así adaptarla a

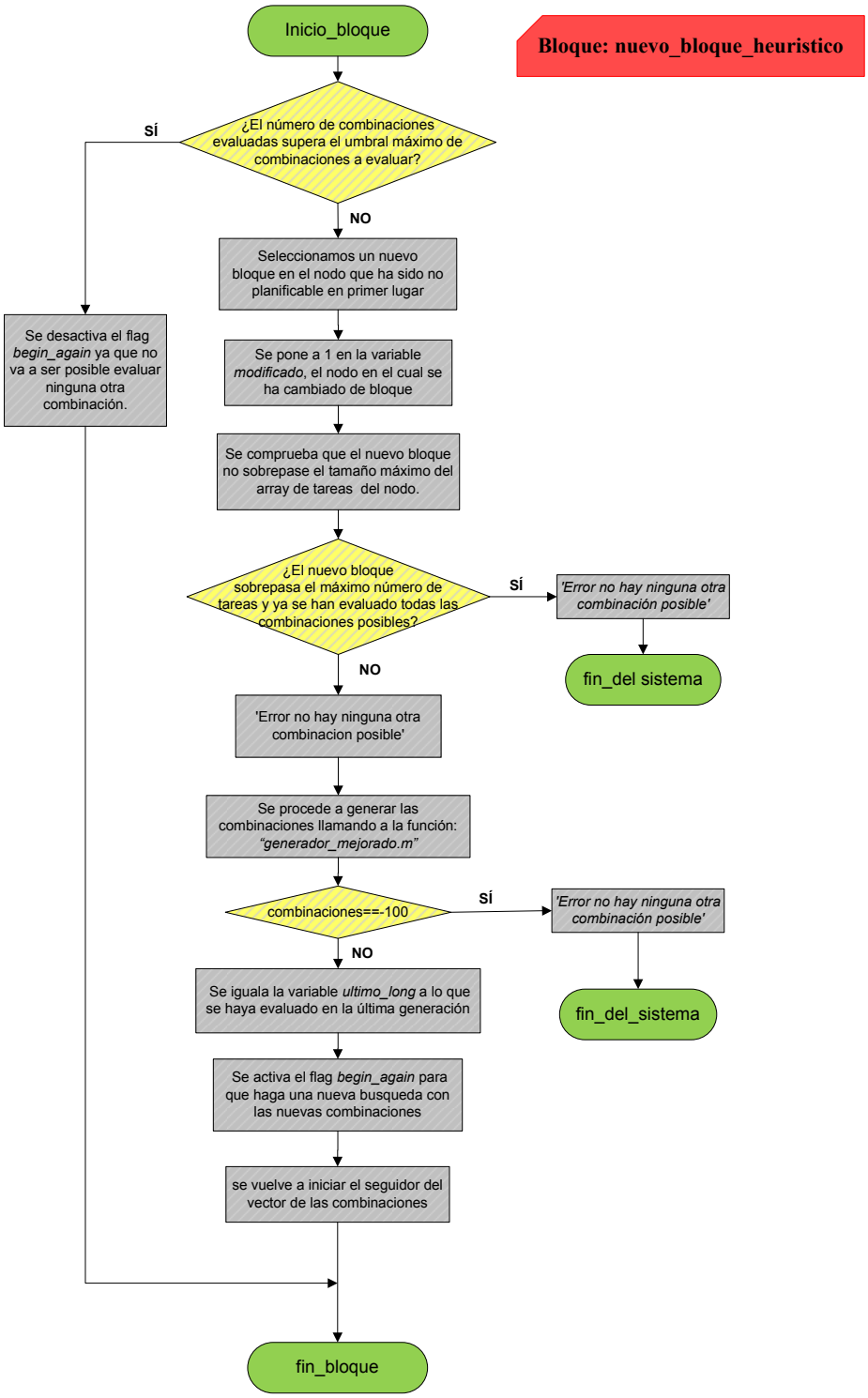

Figura 5.11: Diagrama bloques del bloque *nuevo_bloque_heuristico*

un nuevo bloque de perfiles. Asimismo pondremos a '1' la posición del servicio que hemos modificado en la variable $modificado$.

34. Se comprueba que los nuevos perfiles que queremos evaluar, existan y no se sobrepase el tamaño máximo del número de perfiles del servicio. En caso de sobrepasarlo, significará que ya se puede asegurar que no existirá ninguna combinación que sea planificable, por lo que se puede finalizar el sistema.

35. Si existen nuevos perfiles a evaluar se procede a generar las nuevas combinaciones, llamando a la función $generador\_mejorado$.

36. En caso de que esta función nos devuelva que no existe ninguna otra combinación posible, se mostrará un error y se finalizará el sistema.

37. Si existiesen nuevas combinaciones, se actualizan las variables usadas y se activa el $flag$ para evaluarlas de nuevo ($begin\_again = 1$) y se vuelve al paso 14 para buscar de nuevo una combinación que sea planificable.

38. Nos encontraremos ahora en los diagrama de la figura 5.5. Este último paso es idéntico para cualquiera de los sistemas, ya que se trata de introducir en los nodos físicos y en las redes físicas los servicios seleccionados como mejores por el algoritmo. Por lo tanto no se procederá a su reproducción, pudiéndose seguir su desarrollo en el paso 22 del sistema genérico 5.1.1.

5.1.3. Sistema mejorado

5.1.3.1. Introducción

Función creada a partir del sistema genérico. Esta función coordinará y dirigirá todas las operaciones que se han de realizar para poder comprobar si el sistema que el cliente desea implementar es planificable dentro de los nodos y redes existentes. A partir de la arquitectura pasada por el cliente, estudiará la planificabilidad de las combinaciones necesarias, y hallará la mínima figura de mérito global, para su realización. Esta función es una versión avanzada de la utilizada en el sistema genérico. Debe su nombre, a raíz de ser una versión mejorada del primer sistema implementado en este proyecto fin de carrera. La diferencia principal con respecto al sistema genérico, es la eliminación de combinaciones que

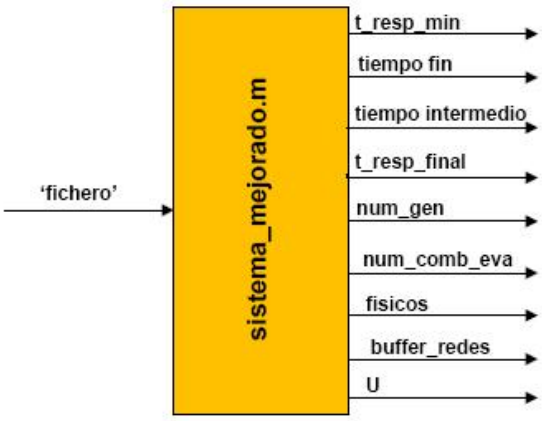

Figura 5.12: Parámetros de entrada/salida de la función *sistema_mejorado.m*

ya sabemos de antemano que no van a ser planificables. La explicación de este algoritmo se puede encontrar en el apartado 4.2.4

5.1.3.2. Parámetros de entrada y de salida

Para poder ejecutar el sistema será necesario que la función reciba por parámetro un fichero adaptado a las especificaciones impuestas y que será la arquitectura de operaciones, nodos y redes que se quiere evaluar (las características de este fichero se pueden encontrar en el apartado 5.3). Este fichero ha de ser introducido entre comillas simples. Un ejemplo de ejecución sería el siguiente:

```
>> sistema_mejorado ('Aplicación')
```

En relación a los parámetros de salida, será el propio usuario el que decida que parámetros recibir. Parece lógico tener como parámetro de salida el mejor tiempo de respuesta. A continuación se muestran los parámetros que se han utilizado como salida a la hora de realizar el diseño y las comprobaciones del sistema:

- **t_resp_min:** En la comprobación de planificabilidad, tiempo de respuesta mínimo del sistema.
- **tiempo_fin:** Devuelve el tiempo en segundos que ha tardado el sistema en ejecutarse.

- **tiempo_intermedio:** Devuelve el tiempo en segundos que ha tardado el sistema en comprobar la planificación.
- **t_resp_final:** Una vez elegida la combinación a introducir en los nodos y las redes, tiempo de respuesta del sistema. Ha de coincidir con el tiempo de respuesta mínimo de la comprobación.
- **num_gen:** Número de combinaciones que tiene la tabla del generador una vez comprobadas todas las necesarias, es decir, se comprobará que si ha habido alguna combinación no planificable, al menos esa combinación habrá sido eliminada de la tabla.
- **num_comb_eva:** Número de combinaciones evaluadas por el sistema en cada ejecución para poder definir su planificabilidad.
- **físicos:** Matrices que contienen los nodos físicos y las tareas en ellos incluidos.
- **buffer_redes:** Matrices que contienen los mensajes transmitidos por cada red física.
- **U:** Utilidades de los nodos físicos.

5.1.3.3. Explicación detallada

Basándonos en los diagramas de flujo, vamos a explicar detalladamente cómo está programado el sistema mejorado. En primer lugar, nos apoyaremos en la figura 5.13:

1. La primera acción a realizar una vez ejecutado el sistema es cargar las definiciones de las distintas variables globales. Posteriormente se inicializarán las variables del sistema.
2. Se procede a hallar el número de niveles, nodos en serie, que tiene la aplicación, así como calcular el número de nodos en paralelo para cada nivel. Asimismo teniendo en cuenta ambos se calcula el número total de nodos que tendrá la aplicación.
3. Se hallan el número de posibles perfiles que habrá para cada operación/servicio.
4. Se creará un vector *long_gen* que contendrá las longitudes de las distintas operaciones del sistema.

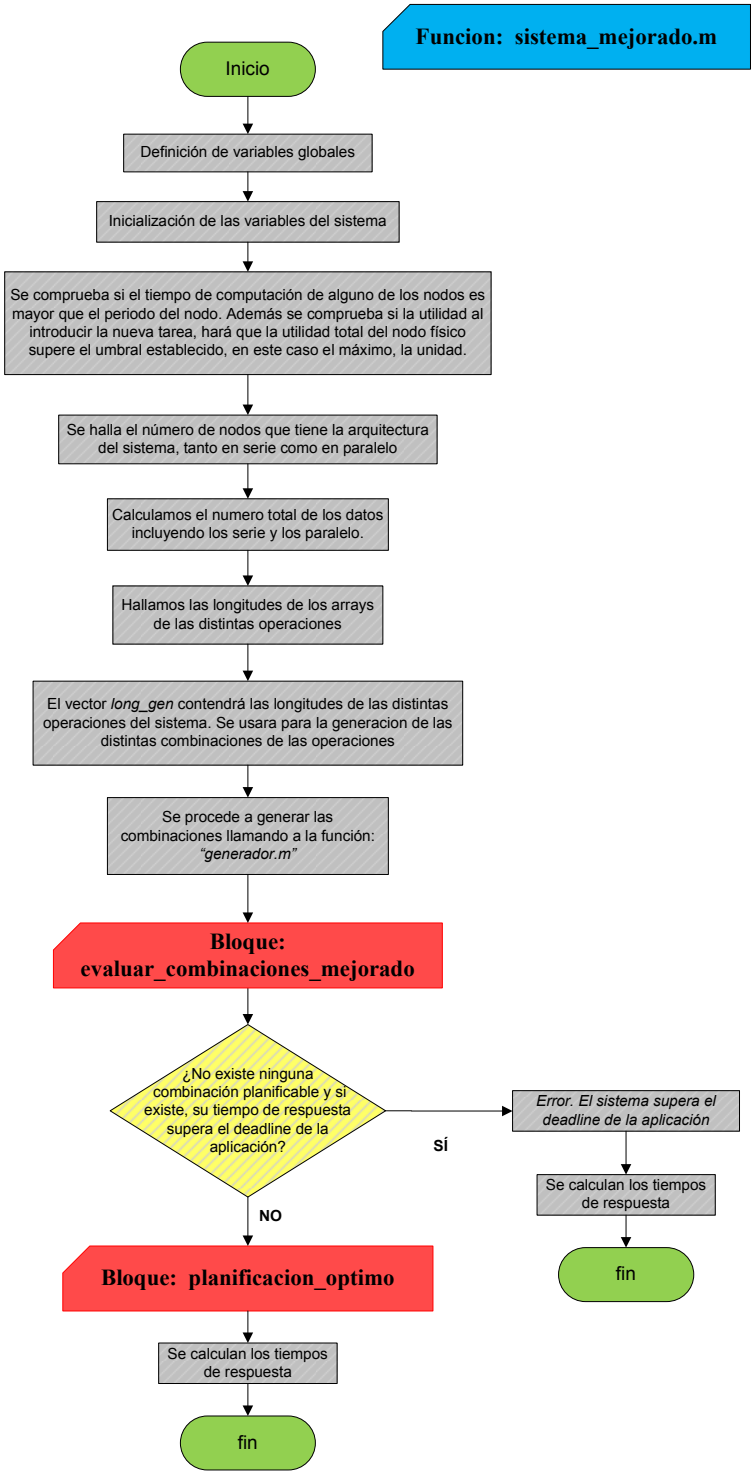

Figura 5.13: Diagrama bloques de la *function sistema_mejorado.m*

5. Unido al paso anterior, y usando su vector, se procede a generar todas las combinaciones posibles de la aplicación. Para ello se llama a la función *generador.m* explicada en 5.2.4.

6. Una vez obtenidas todas las posibles combinaciones procederemos a evaluarlas. Este proceso queda reflejado en la figura 5.14.

7. Bucle *while* irá comprobando combinación a combinación, de las obtenidas en la generación, cuáles de ellas son planificables y en caso de serlo, cual es la que nos da menor tiempo de respuesta. En caso no de existir más combinaciones se va al paso 24.

8. Al evaluar una nueva combinación procedemos a aumentar el contador de combinaciones evaluadas. Asimismo se inicializan las distintas variables de tiempos y de planificación.

9. Se resetean los búferes auxiliares en los cuales se simula el sistema.

10. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en serie.

11. Bucle *while* que mientras haya operaciones en serie a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable en el nodo o en la red, seguirá evaluando la planificabilidad de la combinación. Si fallasen las comprobaciones se va al paso 22.

12. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en paralelo. Nos encontraremos ahora en los diagramas de la figura 5.15.

13. Bucle *while* que mientras haya operaciones en paralelo a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable, en el nodo o en la red, seguirá evaluando la planificabilidad del paralelo. Si fallasen las comprobaciones, paso 21.

    Nota: Hay que resaltar que un solo nodo en un nivel corresponde a una operación en paralelo. En caso de existir dos operaciones en un nivel, habrá dos operaciones en paralelo, y así sucesivamente.

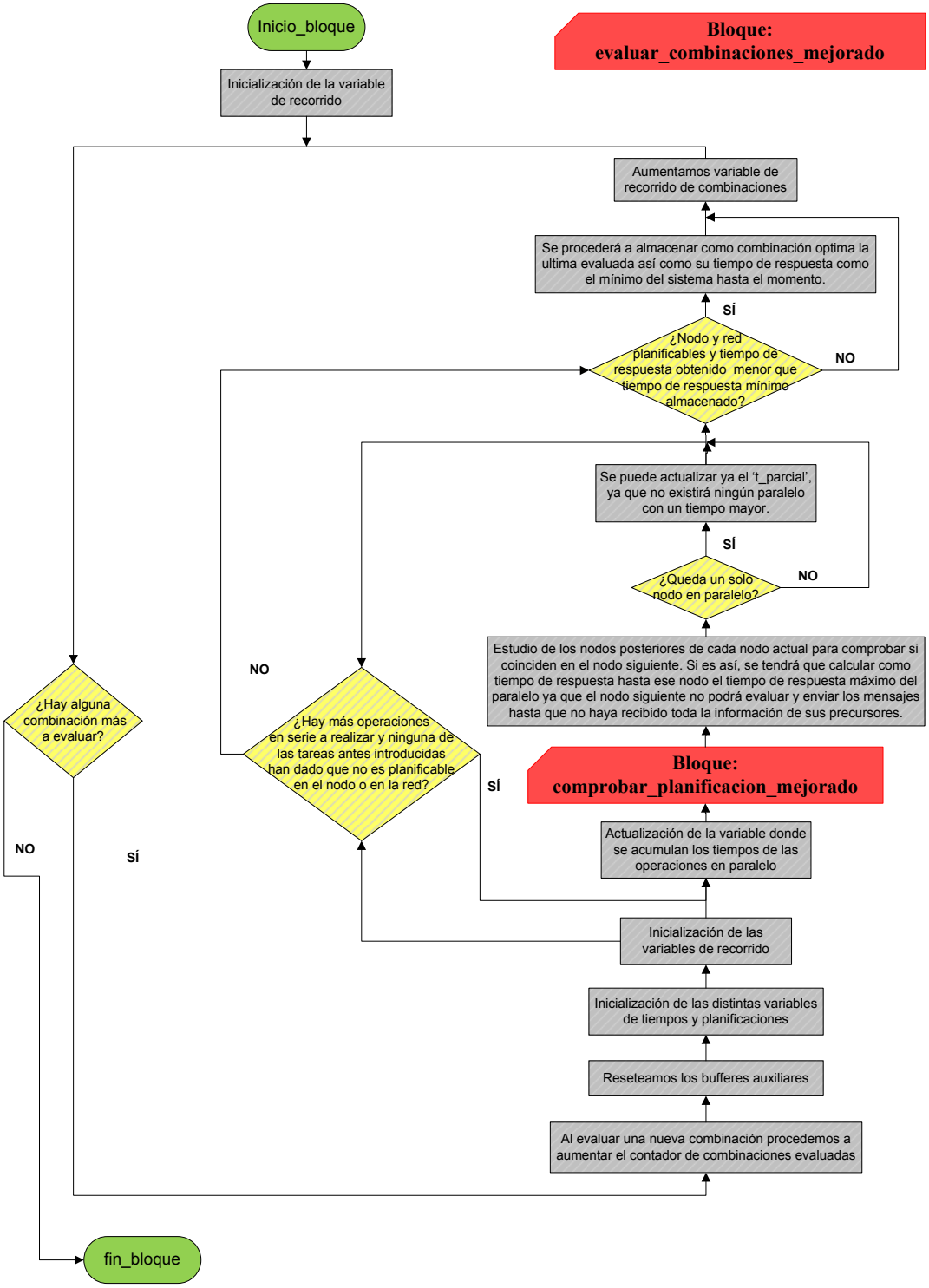

Figura 5.14: Diagrama bloques del bloque *evaluar_combinaciones_mejorado*

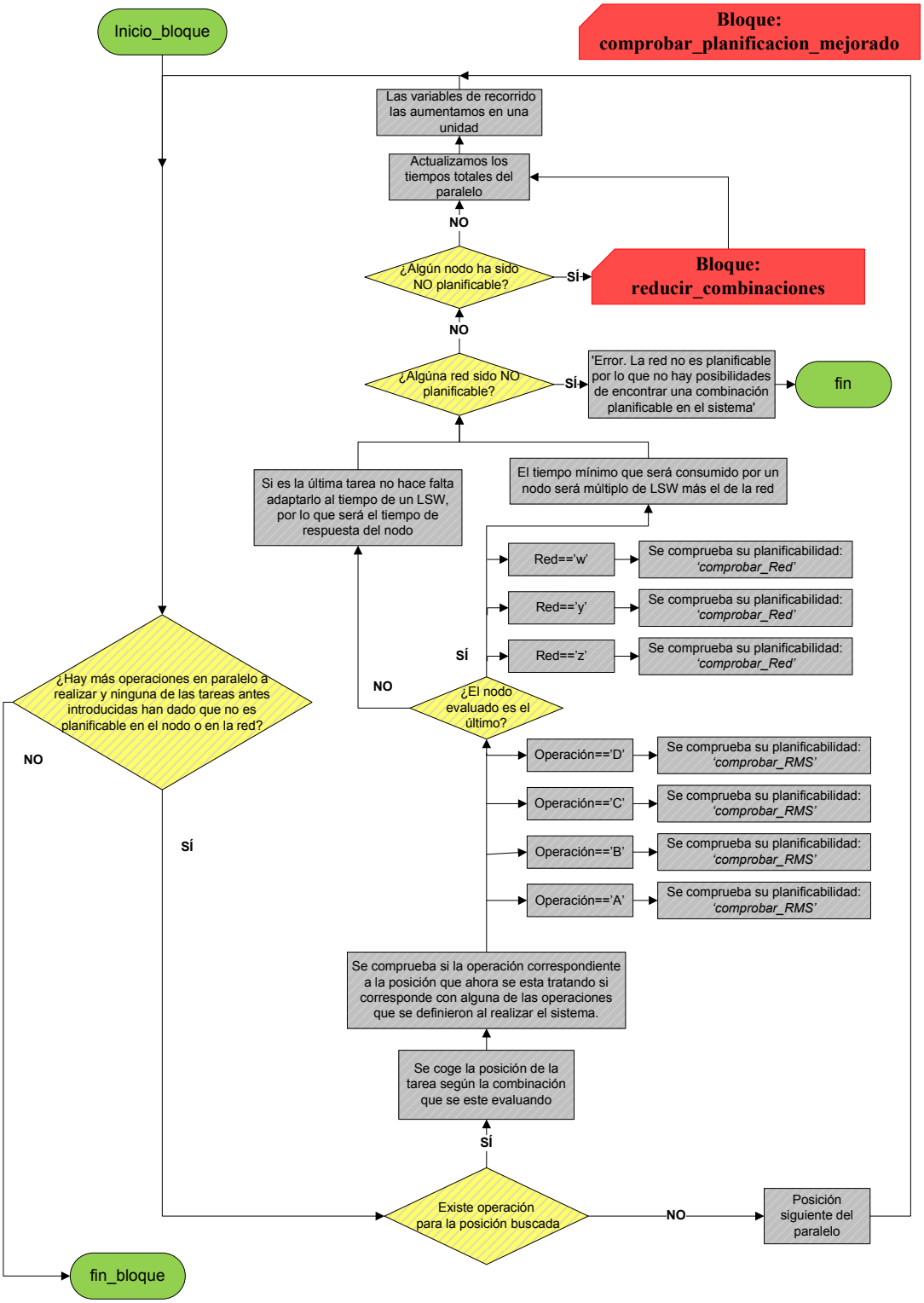

Figura 5.15: Diagrama bloques del bloque *comprobar_planificacion_mejorado*

14. En la combinación que se está evaluando se guarda la posición del nodo que estamos comprobando.

15. Teniendo en cuenta la operación que desarrolle el nodo a evaluar, se procede a comprobar la planificabilidad y el tiempo de respuesta de esa tarea sobre el sistema. Para ello se llama a la función comprobar_RMS (véase 5.2.6).

16. Se evalúa si estamos trabajando con el último nodo de la aplicación, en cuyo caso el tiempo de respuesta total de la operación no hace falta adaptarlo al tiempo LSW. Se pasa al paso 20.

17. Teniendo en cuenta la red sobre la que ha de mandar el mensaje el nodo evaluado, se procede a comprobar la planificabilidad y el tiempo de respuesta de ese mensaje. Para ello se llama a la función comprobar_Red (véase 5.2.7). Se calcula posteriormente el tiempo total de la operación como el tiempo de respuesta de la tarea, adaptado a tiempos LSW, más el tiempo de respuesta del mensaje, que ya está adaptado a tiempos LSW.

18. Se comprueba que el mensaje evaluado haya sido planificable en la red. En caso de que haya dado la no planificabilidad del mensaje, entonces significará que la red no es planificable por lo que no habrá posibilidades de encontrar una combinación planificable en el sistema, con lo que se informa con un mensaje de error que se ha de terminar la evaluación del sistema.

19. Se comprueba que al evaluar la planificabilidad de la tarea, esta haya dado planificable para poder así continuar. En caso contrario, que haya fallado la planificabilidad, se entrará en la siguiente subrutina. Esta subrutina se puede contemplan en la figura 5.16.

    a) Se almacena la combinación de operaciones que ha resultado no planificable. Lo que se pretende es conocer las características de la combinación para ir eliminando las combinaciones posteriores que de antemano ya sabemos que no van a ser planificables.

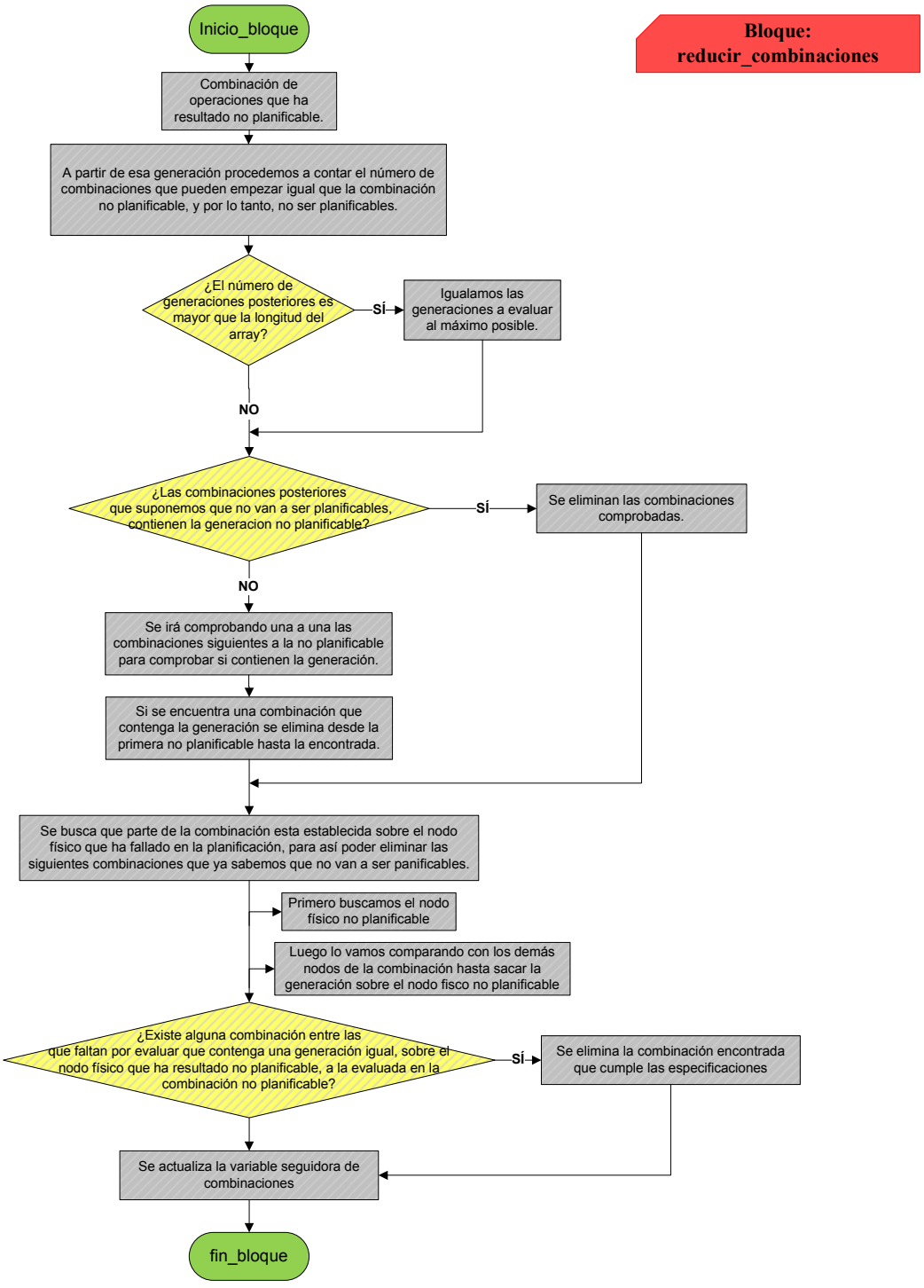

Figura 5.16: Diagrama bloques del bloque *reducir_combinaciones*

b) A partir de esa generación procedemos a contar el número de combinaciones que pueden empezar igual que la combinación no planificable, y por lo tanto, no ser planificables.

c) En el caso que el número hallado en el apartado anterior sea mayor que la longitud del *array*, se procederá a igualarlo a la longitud del *array*.

d) En caso que una combinación nos haya dado no planificable, todas las combinaciones que contengan esa generación no serán planificables. Debido al modo en que las combinaciones son generadas, sabemos que si la combinación correspondiente al número calculado en 19*b* sigue teniendo el mismo patrón que el almacenado en 19*a*. En caso de que se halla hallado en ese lugar se procede con el paso 19*f* .Asimismo se usará este if en el caso que solo esa combinación no sea planificable, es decir que sea una combinación de todos los nodos evaluados y por lo tanto no existan más combinaciones con esa generación.

e) En el caso que alguna combinación se hubiese eliminado anteriormente, la comprobación anterior no nos permitirá encontrar en un principio el fin de las combinaciones que corresponden con el patrón que conocemos que no es planificable, por lo que procederemos a ir buscándolo desde la combinación máxima hallada en 19*b*, hacia arriba, hasta dar con ella.

f) Una vez encontrada la última combinación que contiene el patrón de no planificabilidad se procede a eliminar las combinaciones entre la actual y la última encontrada, ya que estamos seguros que ninguna de ellas será planificable, ahorrándonos su evaluación.

g) Se busca que parte de la combinación está establecida sobre el nodo físico que ha fallado en la planificación, para así poder eliminar las siguientes combinaciones que ya sabemos que no van a ser panificables. Primero buscamos el nodo físico no planificable. Luego vamos comparando con los siguientes nodos para ver cuales trabajan sobre el mismo. Con esta búsqueda se crea un nuevo patrón que contendrá la combinación de servicios que sabemos que no es planificable y que además conocemos que se intenta ejecutar sobre el mismo nodo fisico.

h) Se comprueba desde la combinación actual hasta la última combinación de la tabla de generaciones aún por evaluar, cuales desarrollan el patrón hallado en

el paso anterior. En caso de desarrollar ese patrón se procederá a eliminar esa combinación por conocer previamente que su evaluación va a ser no planificable.

20. Actualizamos los tiempos totales del paralelo. Se vuelve al paso 13.

21. Se realiza un estudio de los nodos posteriores de cada nodo actual para comprobar si coinciden en el nodo siguiente. Si es así, se tendrá que calcular como tiempo de respuesta hasta ese nodo el tiempo de respuesta máximo del paralelo, ya que el nodo siguiente no podrá evaluar y enviar los mensajes hasta que no haya recibido toda la información de sus precursores. Se vuelve al paso 11.

22. Se habrá terminado de evaluar la combinación y si todo ha sido planificable se procederá a comprobar si el tiempo de respuesta obtenido es menor que el tiempo de respuesta mínimo ya almacenado. En caso negativo se pasara al paso 7.

23. Se procederá a almacenar como combinación óptima la última evaluada, así como su tiempo de respuesta como el mínimo del sistema hasta el momento.

24. Ya no existirá ninguna combinación más a evaluar por lo que se procede a comprobar si existe alguna combinación planificable entre las anteriores, y en caso afirmativo si el tiempo de respuesta obtenido en la evaluación no supera el *deadline* de la aplicación. En caso contrario se reportará un error y se finalizará el sistema.

25. Nos encontraremos ahora en los diagrama de la figura 5.5. Este último paso es idéntico para cualquiera de los sistemas, ya que se trata de introducir en los nodos físicos y en las redes físicas los servicios seleccionados como mejores por el algoritmo. Por lo tanto no se procederá a su reproducción, pudiéndose seguir su desarrollo en el paso 22 del sistema genérico 5.1.1.

5.1.4. Sistema heurístico mejorado

5.1.4.1. Introducción

Función que como su propio nombre indica fue creada a partir de los dos sistemas anteriores (sistema heurístico y sistema mejorado). Esta función coordinará y dirigirá todas las operaciones que se han de realizar para poder comprobar si el sistema que el cliente

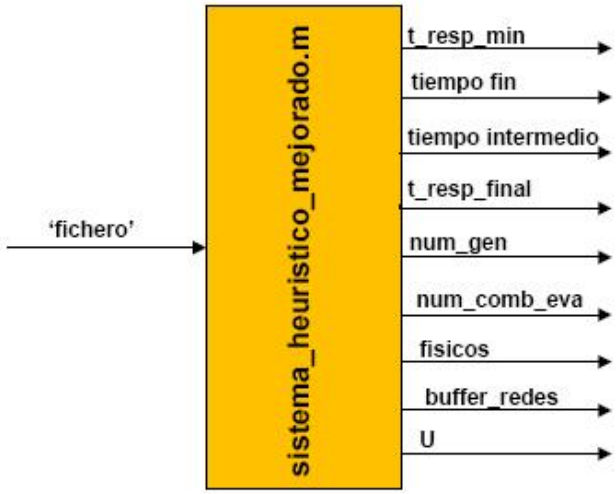

Figura 5.17: Parámetros de entrada/salida de la función *sistema_heuristico_mejorado.m*

desea implementar es planificable dentro de los nodos y redes existentes. Este sistema será un sistema completo ya que aúna las mejoras en el tiempo de ejecución, introducidas por el sistema heurístico, así como la funcionalidad de eliminar combinaciones que ya se sabe de antemano que no van a resultar planificables, que introducía el sistema mejorado.

5.1.4.2. Parámetros de entrada y de salida

Como el resto de los demás sistemas será necesario que la función reciba por parámetro un fichero adaptado a las especificaciones impuestas y que será la arquitectura de operaciones, nodos y redes de la aplicación (las características de este fichero se pueden encontrar en el apartado 5.3). Este fichero ha de ser introducido entre comillas simples. Un ejemplo de ejecución sería el siguiente:

```
>> sistema_heuristico_mejorado ('Aplicacion')
```

En relación a los parámetros de salida, será el propio usuario el que decida que parámetros recibir. A continuación se muestran los parámetros que se han utilizado como salida a la hora de realizar el diseño y las comprobaciones del sistema:

- **t_resp_min:** En la comprobación de planificabilidad, tiempo de respuesta mínimo del sistema.

- **tiempo_fin:** Devuelve el tiempo en segundos que ha tardado el sistema en ejecutarse.
- **tiempo_intermedio:** Devuelve el tiempo en segundos que ha tardado el sistema en comprobar la planificación.
- **t_resp_final:** Una vez elegida la combinación a introducir en los nodos y las redes, tiempo de respuesta del sistema. Ha de coincidir con el tiempo de respuesta mínimo de la comprobación.
- **num_gen:** Número de combinaciones que tiene la tabla del generador una vez comprobadas todas las necesarias, es decir se comprobará que si ha habido alguna combinacion no planificable, al menos esa combinación habrá sido eliminada de la tabla.
- **num_comb_eva:** Número de combinaciones evaluadas por el sistema en cada ejecución para poder definir su planificabilidad.
- **físicos:** Matrices que contienen los nodos físicos y las tareas en ellos incluidos.
- **buffer_redes:** Matrices que contienen los mensajes transmitidos por cada red física.
- **U:** Utilidades de los nodos físicos.

5.1.4.3. Explicación detallada

Basándonos en los diagramas de flujo, vamos a explicar detalladamente cómo está programado el sistema heurístico. En primer lugar, nos apoyaremos en la figura 5.7, que aunque corresponde al sistema heurístico, es similar a los pasos que se seguirán en este nuevo sistema.

Nota: Aunque la mayoría de los pasos del sistema son idénticos a los realizados por los sistemas anteriores, vamos a volver a reproducirlos, ya que se persigue que este capítulo funcione como un manual de usuario a la hora de enfrentarse al código en *Matlab* que se ha programado.

1. La primera acción a realizar una vez ejecutado el sistema es cargar las definiciones de las distintas variables globales. Posteriormente se inicializarán las variables del sistema.

2. Se definirá el tamaño de bloque a utilizar en el algoritmo heurístico. Este tamaño de bloque limitará las combinaciones que se evalúen en el sistema. En nuestro caso se decidió que `tam_bloque=2`.

3. Se procede a hallar el número de niveles, nodos en serie, que tiene la aplicación, así como calcular el número de nodos en paralelo para cada nivel. Asimismo teniendo en cuenta ambos se calcula el número total de nodos que tendrá la aplicación.

4. Se hallan el número de posibles perfiles que habrá para cada operación/servicio.

5. Se creará un vector *long_gen* que contendrá las longitudes de las distintas operaciones del sistema.

6. Antes de realizar la generación de las posibles combinaciones procedemos a ordenar las tareas por su figura de mérito relativa, es decir, un criterio parcial que en nuestro caso será el tiempo de computación de las mismas (C_i). Esta parte corresponderá al algoritmo heurístico mejorado propiamente, que al ser similar a la figura que representa el diagrama de bloques del algoritmo heurístico, podrá éste utilizarse como guía (figura 5.8).

7. Se almacena en una variable el tamaño máximo de las tareas de cada nodo ($long\_gen\_max$), que nos será necesario para poder conocer el número máximo de tamaños de bloques que vamos a poder evaluar.

8. Se modifica la variable *long_gen* adaptándola a las longitudes máximas de las tareas de cada uno de los nodos para la creación de combinaciones, es decir, se adapta al tamaño que se ha seleccionado para cada bloque.

9. Se comprueba que no sobrepase el tamaño máximo de algún nodo, comparándolo con la variable antes establecida, $long\_gen\_max$.

10. Se establece el umbral máximo de combinaciones a evaluar como un cuarto del número máximo de combinaciones posibles. Este umbral se decidió, después de realizar pruebas con diferentes cotas máximas. Lo que se pretende es que en aplicaciones con gran número de servicios no se evalúen un número desorbitado de casos, que harán que el sistema sea ineficiente para trabajar en tiempo real debido a su enorme tiempo de ejecución. Asimismo hay q permitir que en aplicaciones con pocos servicios, el

número de evaluaciones sea adecuado. Con cotas menores, el número de evaluaciones era insuficiente.

11. Se crea la variable *ultimo_long* para almacenar los tamaños de los nodos comprobados anteriormente.

12. Se crea el *array* modificado, que incluirá los nodos que han sido modificados en su tamaño de bloque desde la última generación (representados por '1') y los que no han sufrido modificación alguna (representados por '0').

13. Una vez realizados todos los pasos anteriores, se procede a generar todas las combinaciones posibles teniendo en cuenta el tamaño de los bloques. Para ello se llama a la función *generador_mejorado.m* explicada en 5.2.5.

14. Obtenidas las posibles combinaciones procederemos a evaluarlas. Para ello será necesario la creación de un bucle *while* que comprobará si hay que volver a ejecutar una nueva generación de combinaciones. Este bucle facilitará nuevas entradas a la ejecución del algoritmo en caso de que con la evaluación de los primeros bloques, el algoritmo no haya sido capaz de encontrar una combinación que sea planificable. En caso de no ser necesario nuevas ejecuciones (en caso que el flag *begin_again* este desactivado) se procederá a introducir la combinación más favorable, si existiese: Paso 40.

    A partir de este paso se corresponderán con el diagrama de bloques de la figura 5.9 del sistema heurístico ya que no difiere del utilizado por este nuevo sistema.

15. Bucle *while* irá comprobando combinación a combinación, de las obtenidas en la generación, cuáles de ellas son planificables y en caso de serlo, cual es la que nos da menor tiempo de respuesta. Asimismo comprueba que el número de combinaciones evaluadas no supera el umbral máximo elegido. En caso no de existir más combinaciones a evaluar o superar ese umbral, se va al paso 32.

16. Al evaluar una nueva combinación procedemos a aumentar el contador de combinaciones evaluadas. Asimismo se inicializan las distintas variables de tiempos y de planificación.

17. Se resetean los búferes auxiliares en los cuales se simula el sistema.

18. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en serie.

19. Bucle *while* que mientras haya operaciones en serie a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable en el nodo o en la red, seguirá evaluando la planificabilidad de la combinación. Si fallasen las comprobaciones se va al paso 30.

20. Se inicializa la variable de recorrido del próximo bucle, que servirá para evaluar los nodos en paralelo. Nos encontraremos ahora en los diagramas de la figura 5.18 similar al bloque correspondiente al sistema heurístico.

21. Bucle *while* que mientras haya operaciones en paralelo a realizar y ninguna de las tareas antes introducidas haya dado que no es planificable, en el nodo o en la red, seguirá evaluando la planificabilidad del paralelo. Si fallasen las comprobaciones, se accede al paso 29.

    Nota: Hay que resaltar que un solo nodo en un nivel corresponde a una operación en paralelo. En caso de existir dos operaciones en un nivel, habrá dos operaciones en paralelo, y así sucesivamente.

22. En la combinación que se está evaluando se guarda la posición del nodo que estamos comprobando.

23. Teniendo en cuenta la operación que desarrolle el nodo a evaluar, se procede a comprobar la planificabilidad y el tiempo de respuesta de esa tarea sobre el sistema. Para ello se llama a la función comprobar_RMS (véase 5.2.6).

24. Se evalúa si estamos trabajando con el último nodo de la aplicación, en cuyo caso el tiempo de respuesta total de la operación no hace falta adaptarlo al tiempo LSW. Se pasa al paso 28.

25. Teniendo en cuenta la red sobre la que ha de mandar el mensaje el nodo evaluado, se procede a comprobar la planificabilidad y el tiempo de respuesta de ese mensaje. Para ello se llama a la función comprobar_Red (véase 5.2.7). Se calcula posteriormente el tiempo total de la operación como el tiempo de respuesta de la tarea, adaptado

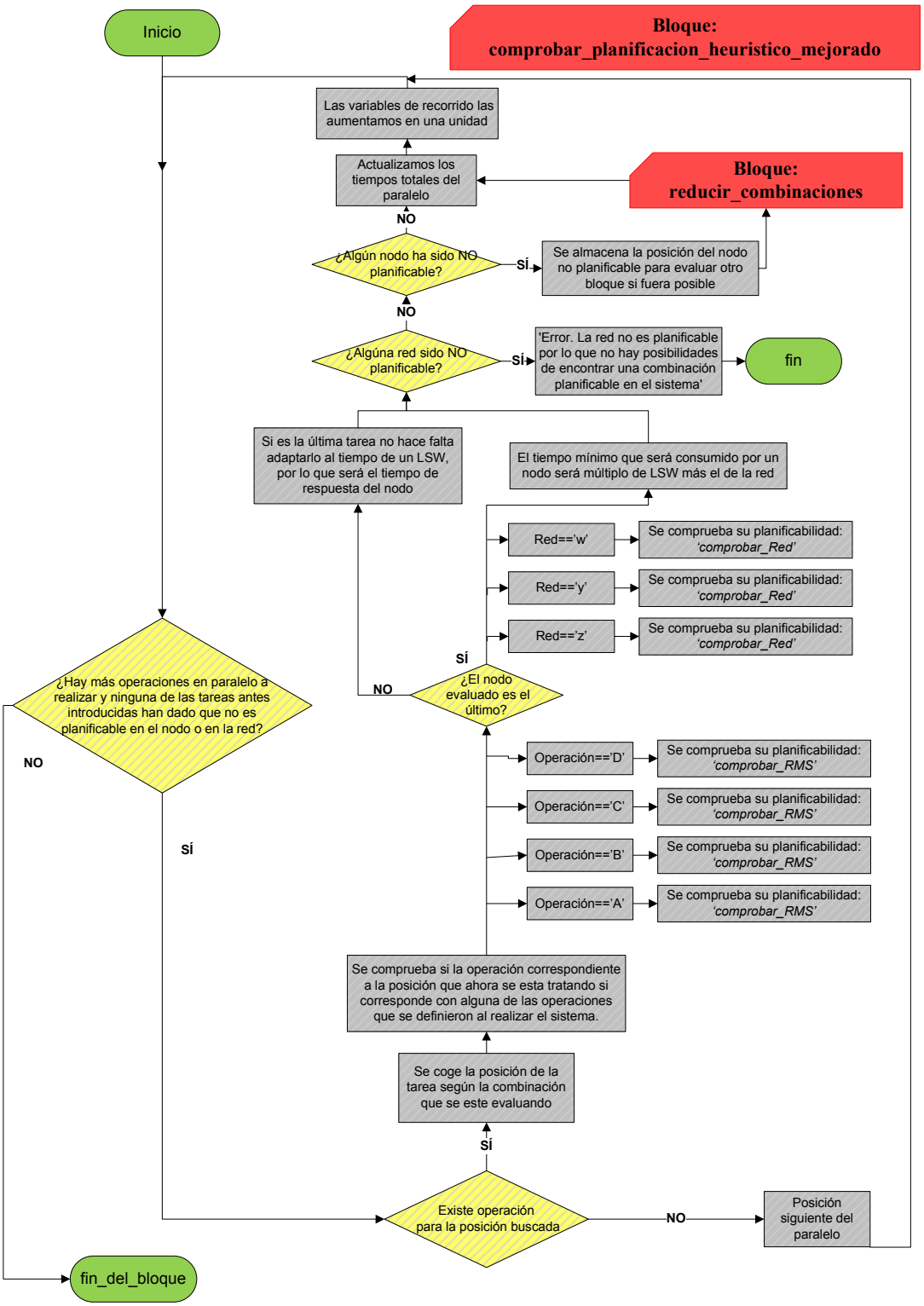

Figura 5.18: Diagrama bloques del bloque *comprobar_planificacion_heuristico_mejorado*

a tiempos LSW, más el tiempo de respuesta del mensaje, que ya está adaptado a tiempos LSW.

26. Se comprueba que el mensaje evaluado haya sido planificable en la red. En caso de que haya dado la no planificabilidad del mensaje, entonces significará que la red no es planificable por lo que no habrá posibilidades de encontrar una combinación planificable en el sistema, con lo que se informa con un mensaje de error que se ha de terminar la evaluación del sistema.

27. Se comprueba que al evaluar la planificabilidad de la tarea, ésta haya dado planificable para poder así continuar. En caso contrario, que haya fallado la planificabilidad, se entrará en la siguiente subrutina. Esta subrutina se puede contemplar en la figura 5.16 correspondiente al sistema mejorado, pero que reproduce los pasos que aquí se van a seguir.

    a) Se almacena la combinación de operaciones que ha resultado no planificable. Lo que se pretende es conocer las características de la combinación para ir eliminando las combinaciones posteriores que de antemano ya sabemos que no van a ser planificables.

    b) A partir de esa generación procedemos a contar el número de combinaciones que pueden empezar igual que la combinación no planificable, y por lo tanto, no ser planificables.

    c) En el caso que el número hallado en el apartado anterior sea mayor que la longitud del *array*, se procederá a igualarlo a la longitud del *array*.

    d) En caso que una combinación nos haya dado no planificable, todas las combinaciones que contengan esa generación no serán planificables. Debido al modo en que las combinaciones son generadas, sabemos que si la combinación correspondiente al número calculado en 27*b* sigue teniendo el mismo patrón que el almacenado en 27*a*. En caso de que se halla hallado en ese lugar se procede con el paso 27*f*.Asimismo se usará este *if* en el caso que solo esa combinación no sea planificable, es decir que sea una combinación de todos los nodos evaluados y por lo tanto no existan más combinaciones con esa generación.

e) En el caso que alguna combinación se hubiese eliminado anteriormente, la comprobación anterior no nos permitirá encontrar en un principio el fin de las combinaciones que corresponden con el patrón que conocemos que no es planificable, por lo que procederemos a ir buscándolo desde la combinación máxima hallada en 27*b*, hacia arriba, hasta dar con ella.

f) Una vez encontrada la última combinación que contiene el patrón de no planificabilidad se procede a eliminar las combinaciones entre la actual y la última encontrada, ya que estamos seguros que ninguna de ellas será planificable, ahorrándonos su evaluación.

g) Se busca que parte de la combinación esté establecida sobre el nodo físico que ha fallado en la planificación, para así poder eliminar las siguientes combinaciones que ya sabemos que no van a ser panificables. Primero buscamos el nodo físico no planificable. Luego vamos comparando con los siguientes nodos para ver cuales trabajan sobre el mismo. Con esta búsqueda se crea un nuevo patrón que contendrá la combinación de servicios que sabemos que no es planificable y que además conocemos que se intenta ejecutar sobre el mismo nodo físico.

h) Se comprueba desde la combinación actual hasta la última combinación de la tabla de generaciones aún por evaluar, cuales desarrollan el patrón hallado en el paso anterior. En caso de desarrollar ese patrón se procederá a eliminar esa combinación por conocer previamente que su evaluación va a ser no planificable.

28. Actualizamos los tiempos totales del paralelo. Se vuelve al paso 21.

29. Se realiza un estudio de los nodos posteriores de cada nodo actual para comprobar si coinciden en el nodo siguiente. Si es así, se tendrá que calcular como tiempo de respuesta hasta ese nodo el tiempo de respuesta máximo del paralelo, ya que el nodo siguiente no podrá evaluar y enviar los mensajes hasta que no haya recibido toda la información de sus precursores. Se vuelve al paso 19.

30. Se habrá terminado de evaluar la combinación y si todo ha sido planificable se procederá a comprobar si el tiempo de respuesta obtenido es menor que el tiempo de respuesta mínimo ya almacenado. En caso negativo se pasará al paso 15.

31. Se procederá a almacenar como combinación óptima la última evaluada así como su tiempo de respuesta como el mínimo del sistema hasta el momento, y seguimos en el paso 15.

32. Ya no existirá ninguna combinación más a evaluar, por lo que se comprueba si hay alguna que haya sido planificable. Si hay al menos una planificable se accederá a introducirla en los nodos y redes físicas (paso 38). En caso de no existir ninguna se accede al paso 33.

33. Esta parte se puede ver reflejada en el bloque 5.11 del sistema heurístico. Una vez terminado de evaluar todas las combinaciones obtenidas con la generación para los tamaños de bloques seleccionados, al no haberse encontrado ninguna combinación planificable, será necesario modificar los perfiles a evaluar del servicio donde primero haya fallado la evaluación.

34. En primer lugar se comprueba si ya se ha superado el umbral de combinaciones máximas a evaluar. En caso de ser así ya no se podrán evaluar más combinaciones, por lo que no será necesario volver a generar más combinaciones con nuevos perfiles. Debido a esto se fuerza al bucle a terminar y abandonar la evaluación, accediendo al sistema de planificación de la combinación óptima. Para que esto suceda se desactiva el flag ($begin\_again$) y se vuelve al paso 14.

35. Si se pueden evaluar nuevas combinaciones, procedemos a modificar la variable $long\_gen$ del servicio donde ha fallado en primer lugar la planificación, para así adaptarla a un nuevo bloque de perfiles. Asimismo pondremos a '1' la posición del servicio que hemos modificado en la variable $modificado$.

36. Se comprueba que los nuevos perfiles que queremos evaluar, existan y no se sobrepase el tamaño máximo del número de perfiles del servicio. En caso de sobrepasarlo, significará que ya se puede asegurar que no existirá ninguna combinación que sea planificable, por lo que se puede finalizar el sistema.

37. Si existen nuevos perfiles a evaluar se procede a generar las nuevas combinaciones, llamando a la función $generador\_mejorado$.

38. En caso de que esta función nos devuelva que no existe ninguna otra combinación posible, se mostrará un error y se finalizará el sistema.

39. Si existiesen nuevas combinaciones, se actualizan las variables usadas y se activa el flag para evaluarlas de nuevo ($begin\_again = 1$) y se vuelve al paso 14 para buscar de nuevo una combinación que sea planificable.

40. Nos encontraremos ahora en los diagrama de la figura 5.5. Este último paso es idéntico para cualquiera de los sistemas, ya que se trata de introducir en los nodos físicos y en las redes físicas los servicios seleccionados como mejores por el algoritmo. Por lo tanto no se procederá a su reproducción, pudiéndose seguir su desarrollo en el paso 22 del sistema genérico 5.1.1.

5.2. Funciones

En esta segunda parte se va a proceder a explicar cada una de las funciones en las cuales se apoya el desarrollo de los sistemas anteriormente explicados. Asimismo en este apartado se va a realizar una explicación exhaustiva para un mejor entendimiento del código desarrollado.

5.2.1. Main

5.2.1.1. Introducción

Esta función se encarga de configurar las variables finales y globales con las que se va a utilizar la aplicación. Asimismo y más importante es la encargada de resetear los búferes tanto de la redes como de los nodos físicos. Esta función será la primera en ser llamada al inicio de cualquier ejecución que se desee realizar ya que es la encargada de llamar a nodos_fisicos.m (véase 5.2.2) y redes_fisicas.m (véase 5.2.3), las cuales crearán los nodos y las redes físicas, que son imprescindibles para poder realizar cualquier ejecución. Una vez creadas, si se desean resetear (por ejemplo, debido a que se han saturado) o que se desea empezar una nueva ejecución con los nodos.

5.2.1.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> Main ()
```

donde podemos observar que no tiene ningún parámetro de entrada ni de salida.

5.2.1.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *Main.m* cuyo diagrama de bloques se puede encontrar en la figura 5.19.

- Variables modificables por el usuario.
  - `EC`: Variable global que nos indicará el tiempo que se establece para EC (ciclo elemental).
  - `LSW`: Variable global que nos indicará el tiempo que se establece para LSW (Longitud de ventana de sincronización).
  - `num_fisicos`: el número de nodos físicos que va a tener la aplicación a desarrollar. Por defecto se ha establecido 3 nodos.
  - `tam_buffer_nodos`: el tamaño de los nodos que van a almacenar las tareas que se intercambiará la aplicación, por defecto se ha decidido que los nodos sean de tamaño 50.
  - `num_redes`: el número de redes que va a tener la aplicación a desarrollar. Por defecto se ha establecido 3 redes.
  - `tam_buffer_redes`: el tamaño de los bufferes que van a almacenar los mensajes que se intercambiará la aplicación, por defecto se ha decidido que los bufferes sean de tamaño 50.
- Finalmente se resetean los nodos físicos así como las redes físicas. Para ello se llamará a las funciones correspondientes con parámetro '-1000'. Es decir:
  - `planificacion_RMS (-1000) ;`
  - `planificacion_Red (-1000) ;`

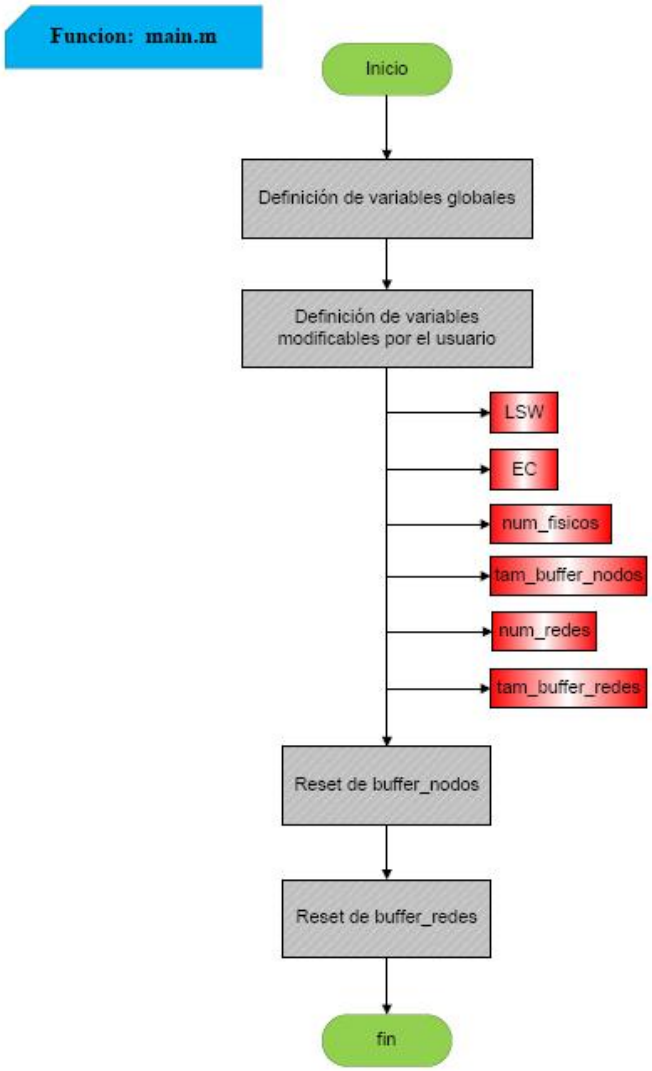

Figura 5.19: Diagrama bloques de la función *Main.m*

5.2.2. Nodos Físicos

5.2.2.1. Introducción

Esta función se encarga de configurar los nodos físicos que va a usar la aplicación. Configurará tanto los nodos físicos como sus respectivas variables y matrices auxiliares que serán necesarias para realizar las comprobaciones de planificabilidad. Esta función será llamada por la función *Main.m* al iniciar un sistema y cuya explicación se encuentra en el apartado 5.2.1.

5.2.2.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> nodos_fisicos ()
```

donde podemos observar que no tiene ningún parámetro de entrada ni de salida.

5.2.2.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *nodos_fsicos.m* cuyo diagrama de bloques se puede encontrar en la figura 5.20.

- Se establecerán las variables locales que se van a utilizar a lo largo de la ejecución del sistema. Estas serán los *arrays* que contendrán los nodos físicos, las utilidades, las tareas que se están ejecutando y los tamaños de los búferes que contendrán esas tareas. Asimismo se establecen los búferes auxiliares que servirán como apoyo a la hora de las comprobaciones de planificabilidad.
- Se crean los búferes globales que almacenarán las tareas en cada uno de los nodos físicos. Las tareas que se almacenarán tendrán los siguientes cinco parámetros:

  [C_i T_i D_i `Pr Op`] es decir [T.computacion Periodo Deadline Prioridad Operacion]
- Se generan las variables globales que almacenarán el número de tareas que ya se están ejecutando en cada nodo.
- Se crean las variables globales que almacenarán la utilidad de cada nodo físico.

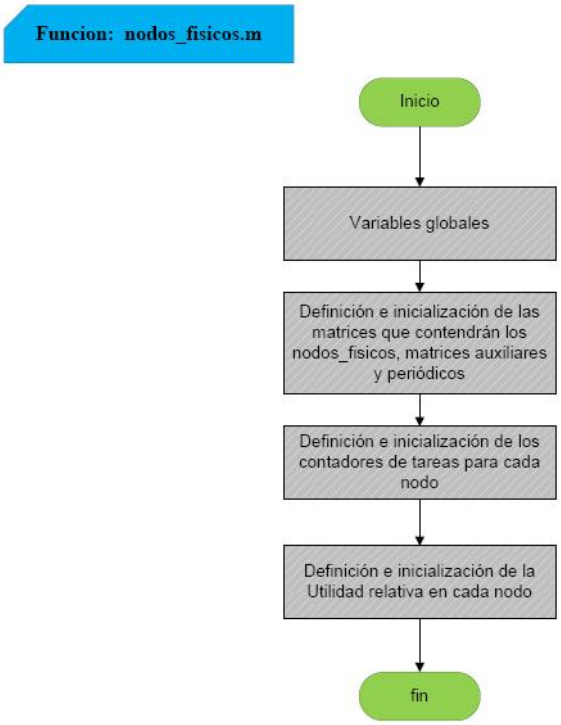

Figura 5.20: Diagrama bloques de la función *nodos_fisicos.m*

5.2.3. Redes Físicas

5.2.3.1. Introducción

Esta función se encarga de configurar las redes que va a usar la aplicación. Configura tanto las redes físicas como sus respectivas variables y matrices auxiliares necesarias para realizar las comprobaciones de planificabilidad en la red. Esta función será llamada por la función *Main.m*, al inicio de cualquier inicialización de un sistema y cuya explicación se encuentra en el apartado 5.2.1.

5.2.3.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> redes_fisicas ()
```

donde podemos observar que no tiene ningún parámetro de entrada ni de salida.

5.2.3.3. Explicación detallada

En este apartado se explica detalladamente el código de la función *redes_fisicas.m* cuyo diagrama de bloques se puede encontrar en la figura 5.21.

- Se establecerán las variables locales que se van a utilizar a lo largo de la ejecución del sistema. Estas serán los *arrays* que contendrán las redes físicas, los tamaños de los búferes que contendrán los mensajes y los punteros indicando donde introducir los mensajes. Asimismo se establecen los búferes auxiliares que servirán como apoyo a la hora de las comprobaciones de planificabilidad en la red.

- Se crean los búferes globales que almacenarán los mensajes en cada uno de las redes físicas. Los mensajes que se almacenarán tendrán los siguientes cuatro parámetros:

  [C_i D_i `num_mens t_resp_mens`] es decir [t_computacion Deadline posicion_en_red t_resp_mens]

- Asimismo se crearán los búferes auxiliares idénticos a los búferes globales creados en el punto anterior.

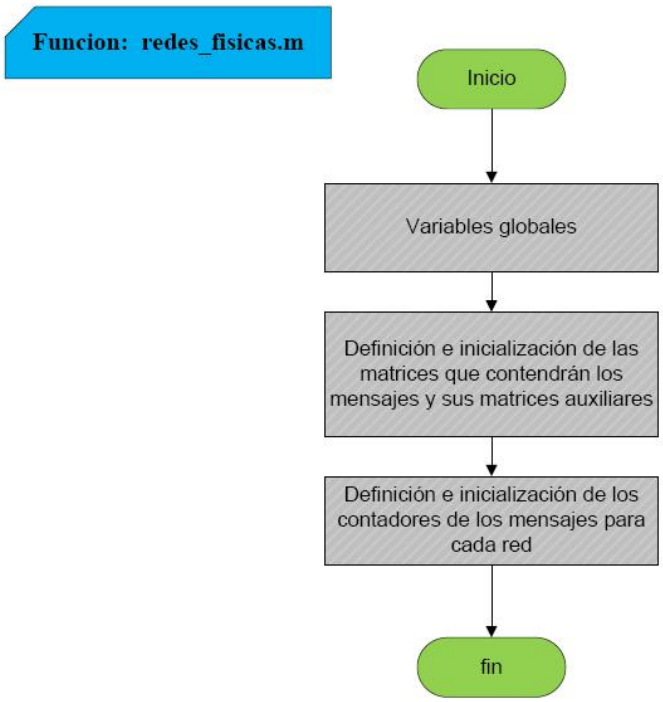

Figura 5.21: Diagrama bloques de la función *redes_fisicas.m*

- Se generan las variables globales que almacenarán el número de mensajes que ya se están ejecutando en cada nodo. Estas variables se denominarán punteros.

5.2.4. Generador

5.2.4.1. Introducción

Esta función es la encargada de generar las diferentes combinaciones para así poder evaluar y elegir cuál es la óptima. Estas combinaciones serán unas combinaciones de números; cada dígito de la combinación representará el perfil de una operación. Por lo tanto, tendremos tantas combinaciones posibles como diferentes posibilidades entre los distintos perfiles de las diferentes operaciones de la aplicación.

5.2.4.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

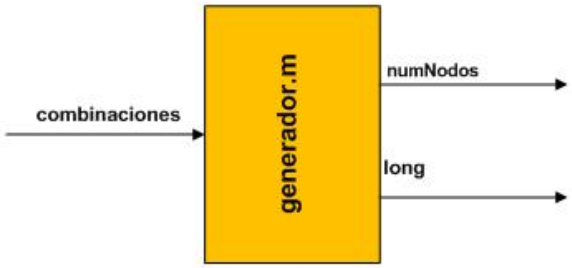

Figura 5.22: Parámetros de entrada/salida de la función *generador.m*

```
>> [combinaciones]=generador (numNodos, long)
```

donde los parámetros de entrada son:

- **numNodos:** variable que indicará el número de nodos totales (serie más paralelo) que existirán en la aplicación.
- **long:** será una matriz de una sola fila y tantas columnas como el número de operaciones que tenga la aplicación. Lo que indicará cada dígito de la matriz *long* es el número de perfiles posibles que tendrá cada servicio. Estos perfiles serán previamente especificados a la hora de llamar al sistema.

y el parámetro de salida:

- **combinaciones:** matriz con número de columnas igual al número de nodos y número de filas que depende del número de combinaciones posibles. Nos devolverá todas las combinaciones posibles entre las distintas operaciones.

5.2.4.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *generador.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.23.

- En primer lugar habrá un *switch* que dependiendo del número de nodos nos llevará a realizar unas operaciones u otras, aunque básicamente el funcionamiento del gene-

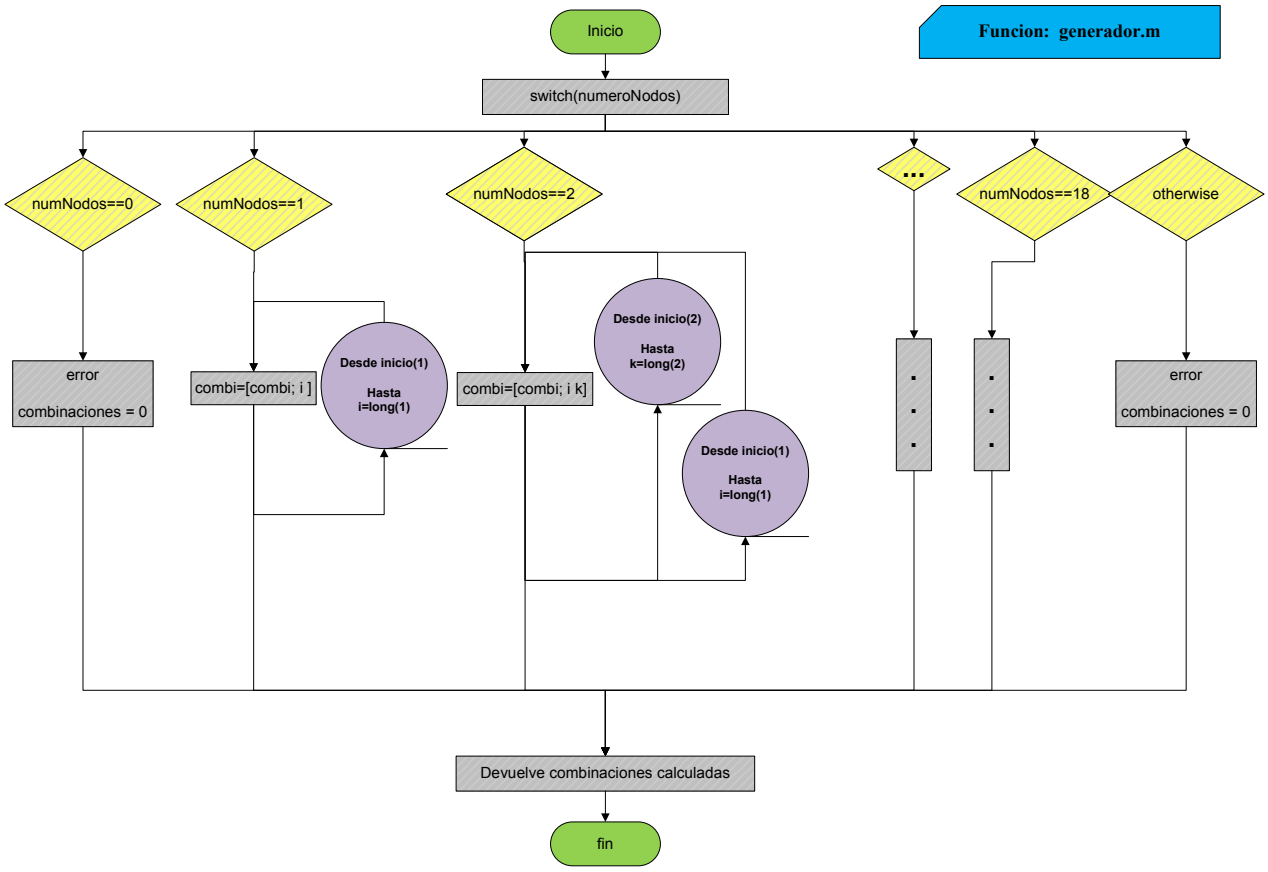

Figura 5.23: Diagrama bloques de la función *generador.m*

rador es idéntico para cualquier número de nodos, por lo tanto vamos a proceder a explicarlos para tres nodos, por ejemplo.

- Una vez seleccionado el número de nodos pasados por parámetro, en nuestro caso tres, se procede a comprobar que la longitud del vector *long* recibido por parámetros es idéntico al número de nodos. En caso contrario se mostrará un mensaje de error y se devolverá que las combinaciones son cero.

- Se crearán tres búferes *for* anidados. Todos ellos empezarán en uno y terminarán en el valor pasado en el vector *long*. En el caso del primer *for*, irá hasta el valor de la primera posición del vector *long*, y así sucesivamente.

- Al estar los bucles anidados haciendo combinación de todos ellos iremos obteniendo las distintas combinaciones que se irán almacenando en filas en el vector de salida *combinaciones*.

- Una vez construido todo este vector de combinaciones se devuelve como salida, terminando la función.

5.2.5. Generador Mejorado

5.2.5.1. Introducción

Esta función es la encargada de generar las diferentes combinaciones para así poder evaluar y elegir cuál es la óptima. La diferencia con la función 'generador', radica en que esta será usada por los algoritmos heurísticos, ya que realiza generaciones de combinaciones dependiendo del tamaño de bloque seleccionado. Estas combinaciones serán unas combinaciones de números; cada dígito de la combinación representará el perfil de una operación. Por lo tanto tendremos tantas combinaciones posibles como diferentes posibilidades entre los distintos perfiles de las diferentes operaciones de la aplicación.

5.2.5.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

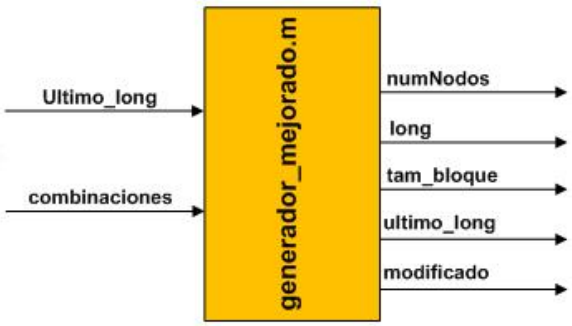

Figura 5.24: Parámetros de entrada/salida de la función *generador_mejorado.m*

>> `[ultimo_long,combinaciones]= generador_mejorado (numNodos,long,tam_bloque, ultimo_long, modificado)`

donde los parámetros de entrada son:

- **numNodos:** variable que indicará el número de nodos totales (serie más paralelo) que existirán en la aplicación.
- **long:** será una matriz de una sola fila y tantas columnas como el número de operaciones que tenga la aplicación. Lo que indicará cada dígito de la matriz *long* es el número de perfiles posibles que tendrá cada servicio. Estos perfiles serán previamente especificados a la hora de llamar al sistema.
- **tam_bloque:** variable que indicará el tamaño del bloque en el cual se van a evaluar las distintas combinaciones de los nodos. Con ello se evita generar todas las posibles combinaciones, generándose las correspondientes al tamaño de bloque definido.
- **ultimo_long:** variable para almacenar los tamaños de los nodos comprobados anteriormente.
- **modificado:** *array* que incluirá los nodos que han sido modificados en su tamaño de bloque desde la última generación (representados por un '1') y los que no han sufrido modificación alguna (representados por un '0').

y los parámetros de salida:

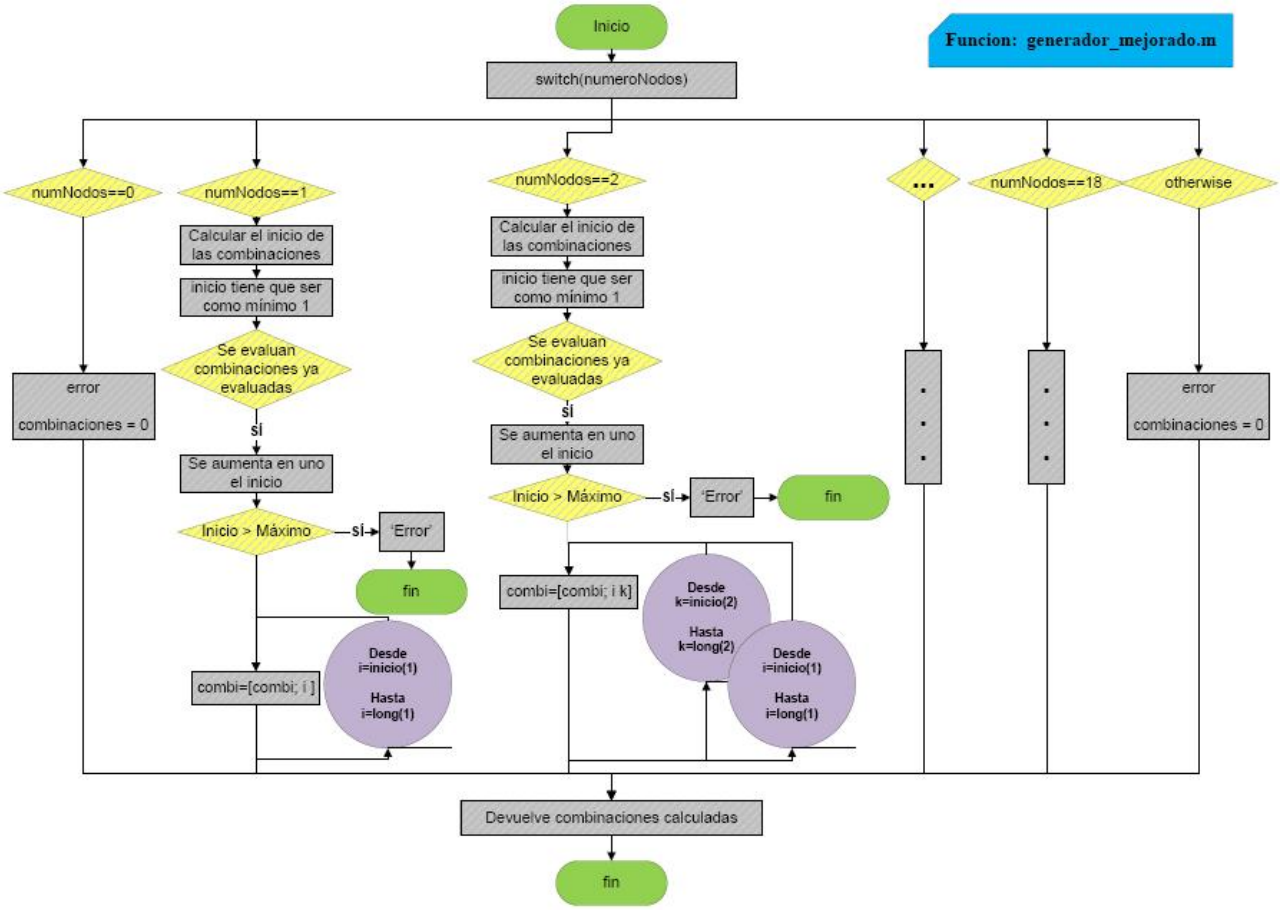

Figura 5.25: Diagrama bloques de la función *generador_mejorado.m*

- **combinaciones:** matriz con igual número de columnas que el número de nodos y número de filas que depende del número de combinaciones posibles. Nos devolverá todas las combinaciones posibles entre los distintos nodos teniendo en cuenta el tamaño de bloque introducido.

- **ultimo_long:** variable para almacenar los tamaños de los nodos comprobados en la última ejecución.

5.2.5.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *generador_mejorado.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.25.

- En primer lugar habrá un *switch* que dependiendo del número de nodos nos llevará a realizar unas operaciones u otras, aunque básicamente el funcionamiento del generador es idéntico para cualquier número de nodos, por lo tanto vamos a proceder a explicarlos para tres nodos, por ejemplo.

- Una vez seleccionado el número de nodos pasados por parámetro, en nuestro caso tres, se procede a comprobar que la longitud del vector *long* recibido por parámetros es idéntico al número de nodos. En caso contrario se mostrará un mensaje de error y se devolverá que las combinaciones son cero.

- Se crea una variable auxiliar donde iremos almacenando el inicio de las combinaciones para cada nodo.

- Se busca el inicio desde donde se empiezan a crear combinaciones para cada nodo. Para ello se almacena en la variable auxiliar antes creada el resultado del vector long (en cada nodo) menos el tamaño de bloque decidido más una unidad. Esto nos indicará que a partir de ese perfil deberemos empezar a generar las combinaciones.

- Posteriormente se realizan una serie de comprobaciones sobre el inicio obtenido anteriormente. En caso que el número inicial sea menor que 1, se aumenta hasta 1 ya que todos los vectores empiezan por 1.

- En caso que quieran evaluar combinaciones que ya se evaluaron en anteriores ocasiones, procederemos a aumentar el inicio para que esto no ocurra. Esto se comprueba observando los vectores *ultimo_long* y $modificado$.

- En el supuesto caso que después de haber realizado las comprobaciones anteriores se observase que hubiese más combinaciones posibles (la variable de inicio esté por encima del máximo de perfiles de ese servicio), nos indicará que ya se han evaluado todas las combinaciones de ese servicio, y procederemos a advertirlo con un error.

- Se crearán tres búferes for anidados. Todos ellos empezarán en uno y terminarán en el valor pasado en el vector *long*. En el caso del primer for, irá hasta el valor de la primera posición del vector *long*, y así sucesivamente.

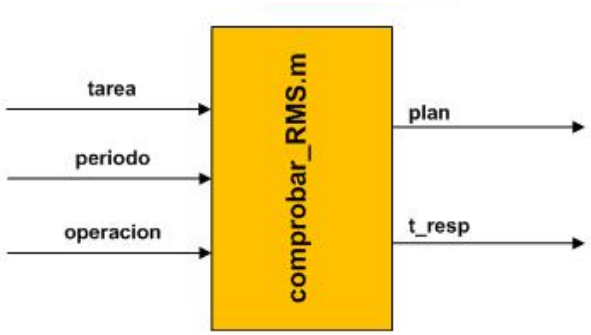

Figura 5.26: Parámetros de entrada/salida de la función *comprobar_RMS.m*

- Al estar los bucles anidados haciendo combinación de todos ellos iremos obteniendo las distintas combinaciones que se irán almacenando en filas en el vector de salida *combinaciones*.
- Una vez construido todo este vector de combinaciones se devuelve como salida, terminando la función.

5.2.6. Comprobar RMS

5.2.6.1. Introducción

Esta función se encarga de realizar una comprobación sobre los nodos físicos existentes para hallar si existe la posibilidad de añadir una nueva tarea de un determinado tiempo de computación sobre el nodo físico señalado. En ningún caso, la función va a introducir las tareas recibidas en los nodos físicos, sino que va a introducirlos en búferes auxiliares que simulan los nodos físicos y así nos darán su planificabilidad. Esta función surge de la decisión tomada que una vez introducida una tarea en una red física, esta no podría ser eliminada.

5.2.6.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

>> `[plan,`$t_r esp$`]=`*comprobar_RMS* `(tarea,periodo,operacion)`

donde los parámetros de entrada son los siguientes:

- **tarea:** será una matriz de 1 sola fila y 2 columnas, que incluirán el tiempo de computación del mensaje, así como el nodo físico en el que se debe incluir el mensaje. Cada 'tarea' estará compuesta por [C_i Nodofísico].

- **periodo:** Al tratarse de una tarea periódica y trabajar sobre RMS, el periodo y el *deadline* coincidirán, por lo que es el tiempo máximo de respuesta del nodo y que será tomado como el tiempo máximo de respuesta de la nueva tarea que quiere ser introducida en la red.

- **operación:** Parámetro que se refiere al código de la operación que se quiere realizar sobre ese nodo. Lo que se pretende con este parámetro es que al final cuando se muestran al usuario las configuraciones, el usuario tenga más fácil distinguir las operaciones que se han realizado sobre los diferentes nodos físicos. En este caso los códigos de operación se distribuyen de la siguiente manera:

  - **'1'** se corresponderá con una operación de red 'A'
  - **'2'** se corresponderá con una operación de red 'B'
  - **'3'** se corresponderá con una operación de red 'C'
  - **'4'** se corresponderá con una operación de red 'D'

y los parámetros de salida serán:

- **plan:** Booleano que nos indica si el mensaje introducido ha sido planificable sobre el nodo auxiliar físico que se pedía. En caso afirmativo se devolverá un '1', en caso negativo un '0'.

- t_resp**:** tiempo que necesita la tarea para ser ejecutada en el nodo, este tiempo dependerá de lo cargado que esté el nodo con las tareas previas.

5.2.6.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *comprobar_RMS.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.27.

1. Se añaden las variables globales que vamos a necesitar.

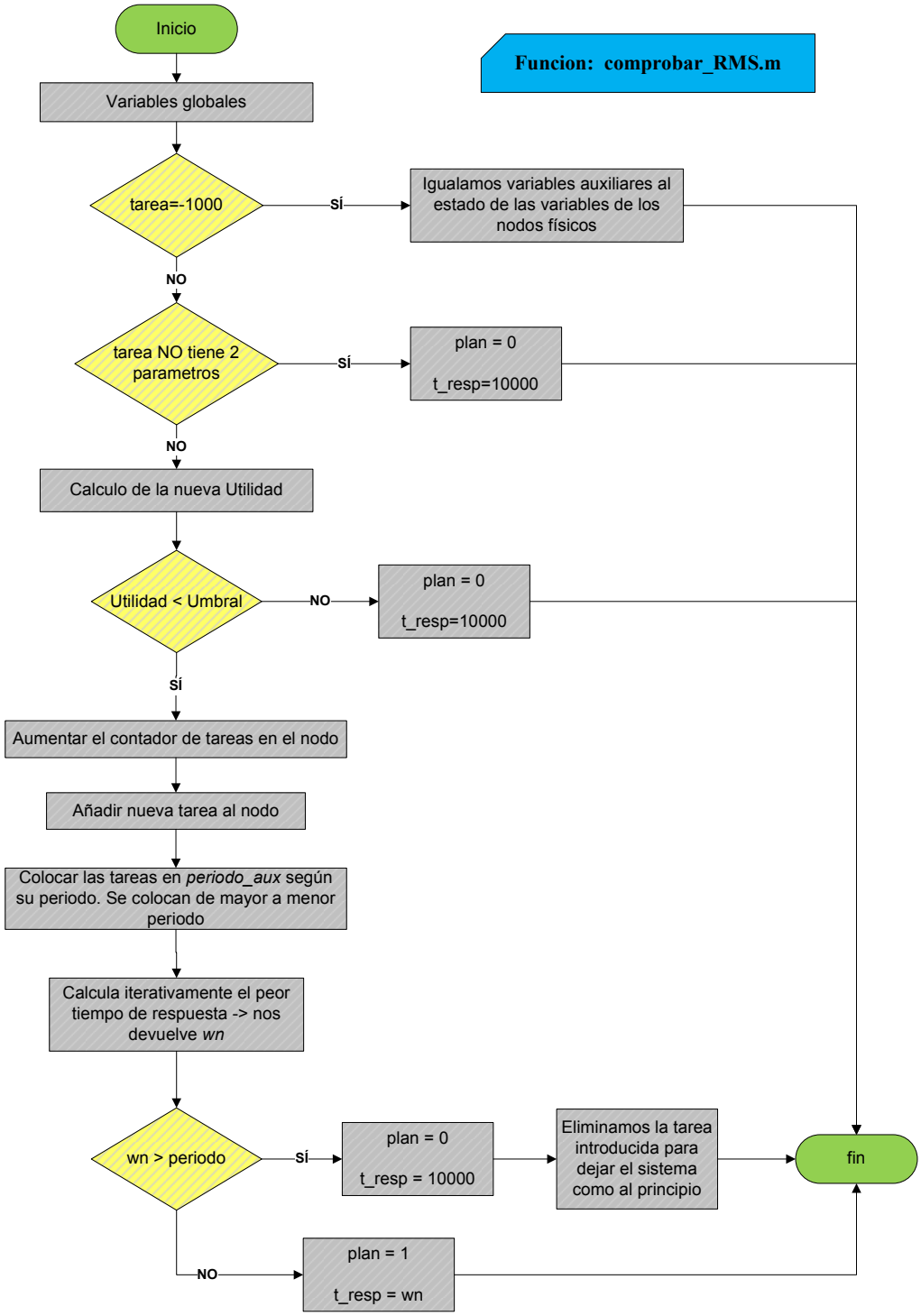

Figura 5.27: Diagrama bloques de la función *comprobar_RMS.m*

2. Se inicializan las variables de salida

3. Si el mensaje introducido es '-1000', entonces se resetean los búferes auxiliares que estamos usando. Para ello, igualamos variables auxiliares al estado de las variables de los nodos físicos, y se finaliza la función. Si no se quieren resetear los búferes se pasa al siguiente paso.

4. Si no se pasan al menos dos parámetros por tarea, ésta será incorrecta y se devolverá un mensaje de error.

5. Calcularemos la utilización del nodo físico en cuestión. Para ello usamos la variable auxiliar en la que tenemos almacenada las diferentes utilidades de los nodos físicos con las tareas ya incluidas en ellos. U_{aux} será la variable auxiliar que contenga las utilidades de las tareas ya incluidas anteriormente en los búferes auxiliares. El cálculo de la utilidad del nodo viene definida como el sumatorio de los tiempos de computación de cada tarea partido los periodos de las mismas. Es decir, la utilidad acumulada en la variable auxiliar más el tiempo de computación de la nueva tarea partido por su periodo.

6. Se comprueba si la Utilidad del nodo no supera el Umbral máximo establecido. En caso de que se supere se devolverá un error advirtiendo que la tarea no es planificable.

7. Al tener una nueva tarea aumentaremos el número de tareas en la variable auxiliar del buffer del nodo que deseamos usar.

8. En primer lugar añadiremos la tarea al buffer. Consideraremos una prioridad común ('1') para todas ellas, que posteriormente modificaremos. Al estar trabajando sobre RMS tanto el periodo como el *deadline* son iguales. Cabe recordar que una tarea en el buffer viene definida por:

   - `[Ci Ti Di Pr Op]` es decir [T.computación Periodo Deadline Prioridad Operación]

9. Se procede a colocar las tareas dependiendo de su periodo para seleccionar su prioridad. Para ello se calculan las prioridades de cada tarea. Las ya existentes se actualizan debido a la llegada de la nueva tarea.

- En el caso que sólo existiese una tarea se le colocaría la prioridad '1' y se pasaría al paso 10.
- Si existen más tareas las iremos ordenando por su periodo, ya que a menor periodo, mayor será su prioridad, como se explicó en la ecuación 2.4.

10. Como ya tenemos las tareas ordenadas por su periodo en el buffer auxiliar, ahora procederemos a añadirle su periodo en el nodo dependiendo de su posición en el buffer auxiliar de periodos. Como sabemos a menor periodo mayor prioridad y en caso de igualdad, tendrá mayor prioridad el que haya entrado antes en el sistema.

11. Finalmente sólo queda calcular el tiempo de respuesta, que se hará de manera iterativa.

12. Iterativamente iremos calculando los tiempos de respuesta, hasta que se cumpla que los tiempos de respuesta de dos iteraciones consecutivas coincidan entre sí. Para el cálculo del tiempo de respuesta se usará la ecuación presentada en 2.19.

13. En caso de haber evaluado 50 iteraciones y no haber encontrado dos iteraciones consecutivas con el mismo tiempo de respuesta, se puede afirmar que el tiempo de respuesta será infinito, y por lo tanto la tarea no será planificable en el sistema.

14. Una vez obtenido el tiempo de respuesta, se comprobará que éste no supera el *deadline* del nodo, ya que de ser así la tarea tampoco sería planificable.

15. Llegado a este punto, se podrá afirmar que la tarea es planificable (plan=1) y que el tiempo de respuesta será el obtenido en la iteración.

5.2.7. Comprobar Red

5.2.7.1. Introducción

Esta función se encarga de realizar una comprobación sobre las redes existentes para hallar si sería posible añadir un nuevo mensaje de un determinado tiempo de computación sobre la red señalada. En ningún caso, esta función va a introducir los mensajes recibidos en la redes físicas, sino que va a introducirlos en búferes auxiliares que simulan las redes

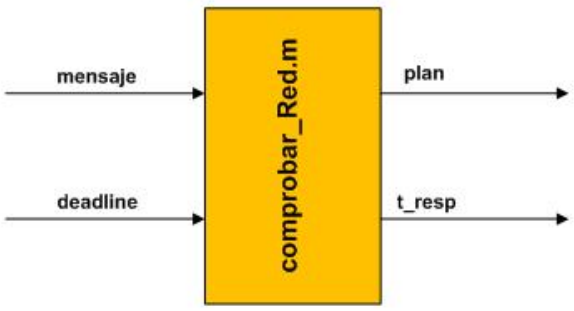

Figura 5.28: Parámetros de entrada/salida de la función *comprobar_Red.m*

físicas y así nos darán su planificabilidad. Esto es debido a que se tomó la decisión de que una vez introducido un mensaje en una red física, este no podría ser eliminado.

Asimismo se ha implementado en la función, que un mensaje añadido anteriormente en tiempo no se pueda ver modificado por un mensaje añadido posteriormente en tiempo, es decir, la red será no planificable si un mensaje ya incluido viese su tiempo de respuesta aumentado debido a la inclusión de un mensaje posterior.

5.2.7.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> [plan,t_resp_mens]=comprobar_Red (mensaje,deadline)
```

donde los parámetros de entrada son los siguientes:

- **mensaje:** será una matriz de 1 sola fila y 2 columnas, que incluirán el tiempo de computación del mensaje, así como la red física en la que se debe incluir el mensaje.
- **deadline:** Tiempo máximo de respuesta de la aplicación y que será tomado como el tiempo máximo de respuesta del nuevo mensaje que quiere ser introducido en la red.

y los parámetros de salida serán:

- **plan:** Booleano que nos indica si el mensaje introducido ha sido planificable sobre la red auxiliar física que se pedía. En caso afirmativo se devolverá un '1', en caso negativo un '0'.

- $t_r esp_m ens$**:** tiempo que necesita el mensaje para ser mandado por la red, este tiempo dependerá de lo cargado que este la red con mensajes previos. Este tiempo es función de los tiempos de EC que sean necesarios usar.

5.2.7.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *comprobar_Red.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.29.

1. Se añaden las variables globales que vamos a necesitar usar.

2. Se inicializan las variables de salida

3. Si el mensaje introducido es '-1000', entonces se resetean los búferes auxiliares que estamos usando. Para ello, igualamos variables auxiliares al estado de las variables de las redes físicas, y se finaliza la función. Si no se quieren resetear los búferes se pasa al siguiente paso.

4. Se vuelca el identificador de la red física a utilizar, incluido en el mensaje por parametro, sobre una variable.

5. En primer lugar tratamos el *deadline* para adaptarlo a los sistemas de LSW y EC. Es decir los *deadlines* de las redes trabajarán sobre formato EC, para lo cual dividimos por LSW que es el tiempo que podemos transmitir en un EC.

6. Añadimos en la red auxiliar correspondiente, el nuevo mensaje junto con el *deadline* de la aplicación y el identificador de operación. Cada Mensaje tendrá [C_i deadline num_mens t_resp_mens]

7. Para hacer más fácil el uso de buffer auxiliar procedemos a volcarlo sobre un búfer auxiliar de 2 dimensiones.

8. Creamos la variable que nos indicará el número de mensaje en la red. Será su identificador para poderlo encontrar en la misma y así poder devolver su tiempo de respuesta.

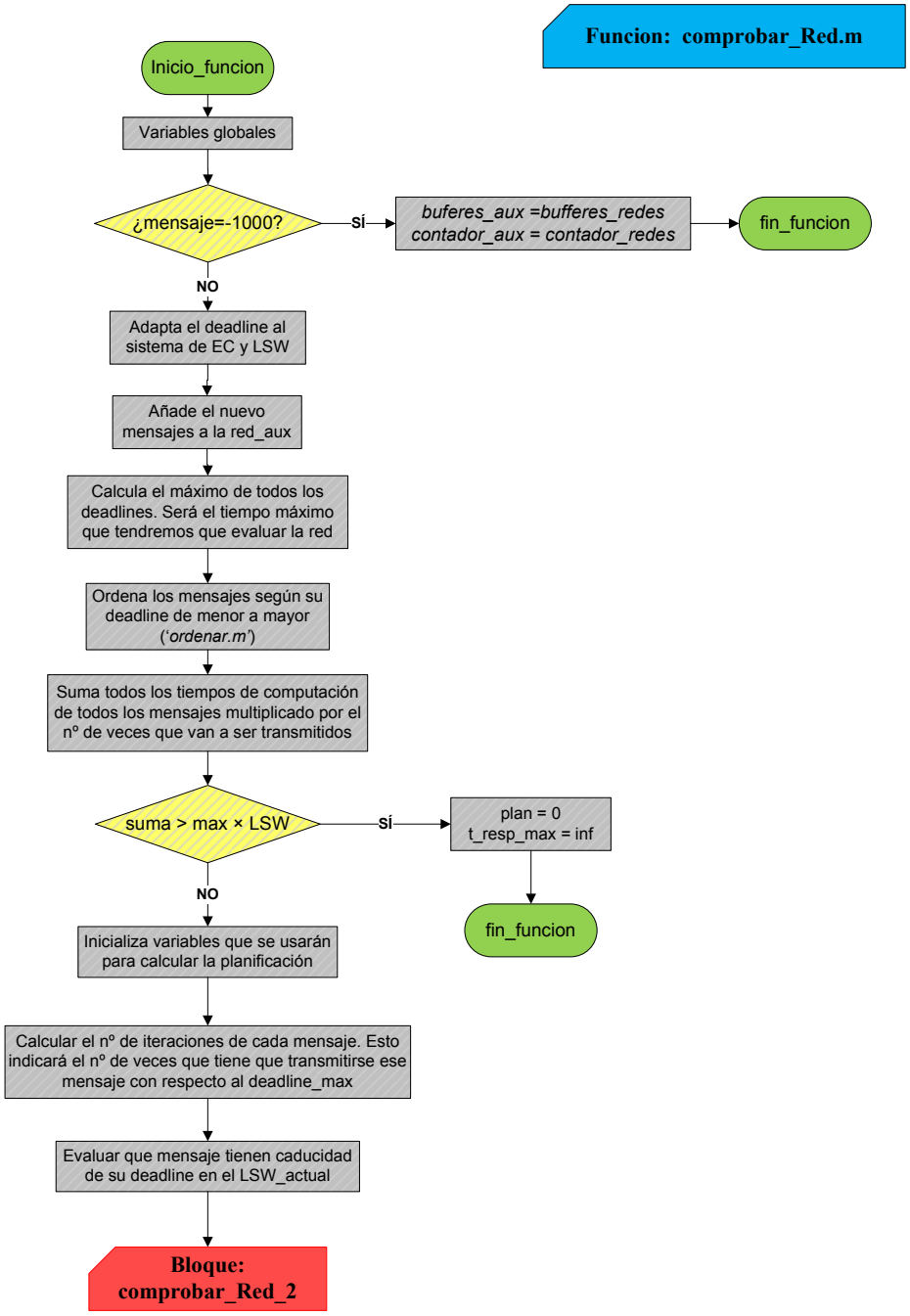

Figura 5.29: Diagrama bloques de la función *comprobar_Red.m*

9. Creamos una variable que tendrá una longitud igual al número de mensajes que haya en la red. Se usará para mostrar los tiempos de respuesta de los distintos mensajes.

10. Búfer que usaremos para almacenar los mensajes cuyo *deadline* caduca en el presente LSW. Con ello evitaremos que se nos pase un mensaje cuya caducidad nos obligaría a declarar la red como no planificable. Con este sistema se da prioridad a los mensajes cuyo *deadline* coincide con el LSW que se está ocupando en cada momento.

11. Calculamos el máximo de todos los deadlines de las tareas, que ya están expresadas en función de LSW, por lo que lo que se está calculando es el máximo de LSWs que hay que comprobar.

12. Antes de operar con los mensajes procedemos a colocarlos según su *deadline* de menor a mayor. En caso de tener el mismo *deadline* se procederá a que se sitúe primero el de menos carga computacional tenga. Para ello se llama a la función ordenar.

13. Primero comprobamos si la suma de los tiempos de computación de todos los mensajes que tenemos que transmitir sobre todos los LSW, es posible transmitirlos usando los LSW del máximo antes calculado. En caso contrario ya podremos indicar que esta red no va a ser planificable.

14. Creamos la variable que nos va a indicar sobre que LSW nos encontramos actualmente.

15. Se crea un vector que muestre la carga computacional introducida en cada LSW.

16. Se crea un booleano para distinguir entre un LSW nuevo y uno sobre el que ya se ha introducido algún mensaje.

17. Vector que nos indica el número de veces que se ha usado ese mensaje. Al estar tratando con mensajes periódicos tendremos que saber el número de ejecuciones que existen de cada mensaje, para contemplar si es necesario otra nueva ejecución con respecto a su *deadline*.

18. Calculo el número de iteraciones de cada mensaje. Esto nos indicará el número de veces que ha de transmitirse ese mensaje con respecto al *deadline* máximo. Exclu-

sivamente va a depender del *deadline* del mensaje y del *deadline* máximo de todos los mensajes de la red.

19. Creación de variables para almacenar el primer elemento que se incluyo en ese LSW para así poder cambiar de LSW en el caso que ningún mensaje pueda ser transmitido en el actual. Es decir que se recorriesen todos los mensajes existentes y volviésemos al almacenado sin haber cambiado de LSW.

20. Bucle para evaluar cuales son los mensajes cuya caducidad corresponden con la del primer LSW. Estos mensajes se ha decidido que tengan prioridad sobre el resto ya que en caso de superar su *deadline* será crítico para la planificación.

21. Creación de diferentes variables para poder gestionar los mensajes al introducirlos a la red y comprobar su planificación.

    A partir de este punto nos encontraremos en la figura 5.30 la cual representará la parte relativa al algoritmo que gestionará la comprobación de planificabilidad en la red.

22. A partir de este punto procedemos a procesar los mensajes a través de la red. Creamos un bucle que se parará en el caso que el LSW que procedamos a evaluar sea superior al máximo de LSWs que tenemos que procesar y que se han calculado en el paso 11.

23. Comprobamos si se diese el caso de haber recorrido todos los mensajes, si es así, se fuerza para volver a empezar con las comprobaciones.

24. Se comprueba si existen mensajes cuya caducidad corresponda con el LSW actual, los cuales tendrán mayor prioridad. En caso que no existan más mensajes que caduquen en este LSW se accede al paso 31.

25. Guardamos la posición del mensaje que se iba a comprobar antes de que existiesen los de mayor prioridad (los que caducan en este LSW), para una vez terminados estos seguir el orden en el que se dejó la ejecución.

26. Se coge como próximo mensaje a evaluar uno de los que caducan en el LSW actual.

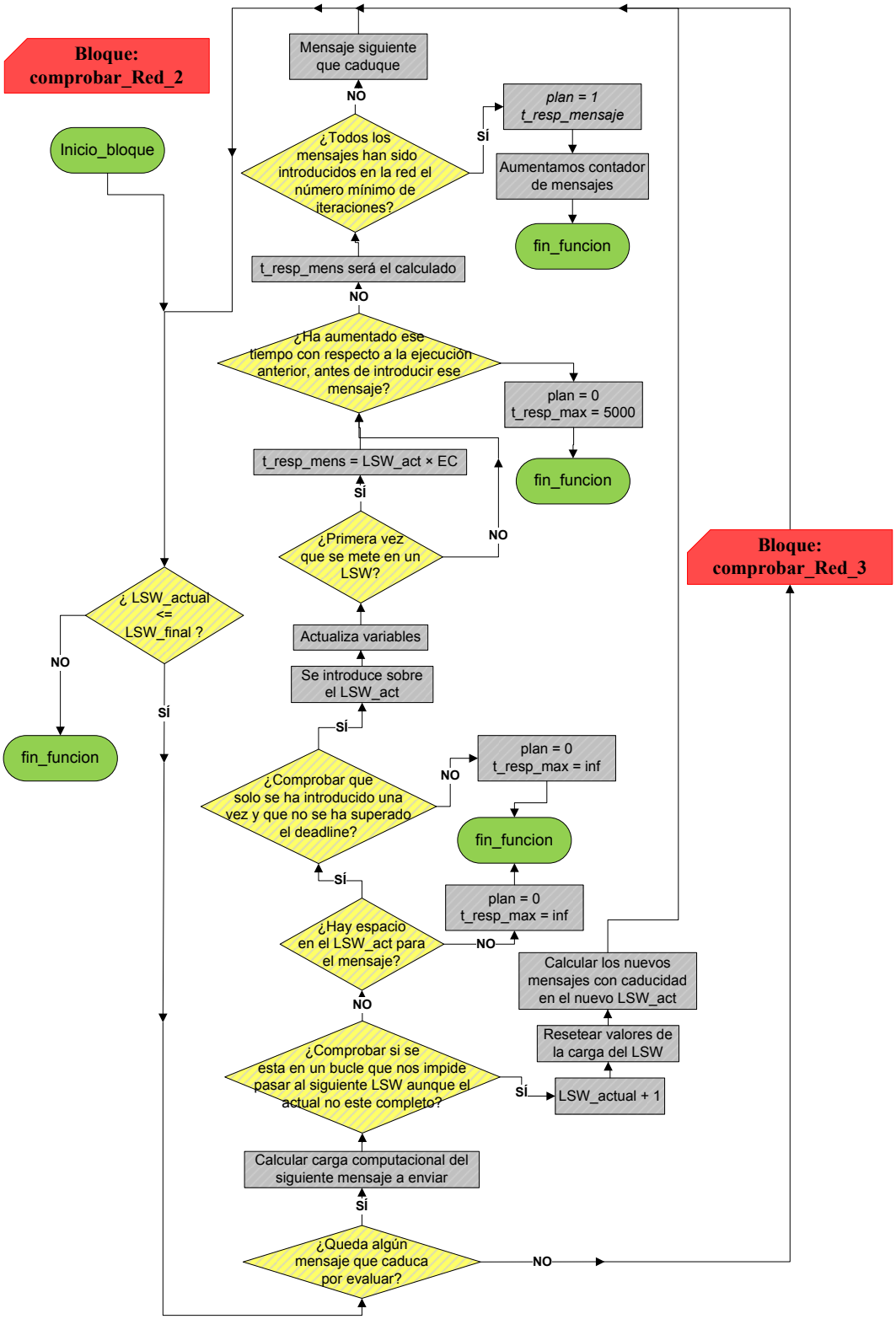

Figura 5.30: Diagrama bloques de la función *comprobar_Red.m* (parte 2)

27. En caso que se comprobara con las distintas variables que nos hemos quedado en un bucle que nos impide rellenar el LSW sobrante, procederemos a pasar al siguiente LSW. Si aun se puede seguir usando el LSW actual se pasa al paso 28.

    a) Se pasaría al siguiente LSW

    b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

    c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

    d) Actualizamos los flags para poder realizar nuevas comprobaciones.

    e) Accedemos al paso 22.

28. Si aun se puede seguir usando el LSW actual. Se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no nos hayamos pasado el *deadline* de este. Si nos lo hemos pasado ya podremos devolver que el mensaje no será planificable y que su tiempo de respuesta será infinito, terminando la función de comprobación de la red.

    a) si todo lo anterior se cumple procederemos a introducir el mensaje sobre el LSW actual. Con lo que se aumentará el tiempo de computación del mismo sumándole el tiempo de computación del nuevo mensaje a introducir, y actualizando las variables que almacenan el último mensaje introducido.

    b) Se incluye el tiempo de respuesta en el vector de tiempos de respuestas y se comprueba que ningún tiempo de respuesta de los mensajes antes almacenados se haya visto incrementado. Si no se ven incrementados se incluye el tiempo de respuesta en el cuarto parámetro del mensaje almacenado. En caso de haber aumentado se declarará la red como no planificable y se establecerá que su tiempo de respuesta es infinito, pudiendo finalizar la función de comprobación de la red.

29. Llegado a este punto los distintos mensajes se han introducido en la red. Si el número de veces que se ha introducido cada uno es igual o superior al número de iteraciones en las que se debía introducir debido a su *deadline* y que se calculo en 18, entonces se puede proceder a declarar la red como planificable. En caso que aun no se haya

alcanzado ese nivel se procederá a seguir comprobando nuevos mensajes para los LSW restantes, para ello vamos al paso 30.

a) Se pone la variable de salida $plan = 1$ ya que el conjunto de mensajes si será planificable

b) Se devuelve el tiempo de respuesta del conjunto de los mensajes que indicarán el tiempo que necesita la red para transmitir ese mensaje.

c) Al haberse comprobado que sí que es planificable se aumenta el puntero para así indicar que hay un nuevo mensaje y modificamos el buffer auxiliar con la nueva configuración.

d) Volvemos a incluir la red modificada en el buffer redes auxiliar.

e) y nos salimos de la función ya que hemos terminado con las comprobaciones.

30. Procedemos a actualizar las variables. Aumentamos el índice que controla el búfer con las variables que caducan en este LSW. Asimismo actualizamos la variable que controla todos los mensajes de la red y la igualamos al último que se dejo sin evaluar en el momento que se paso a los mensajes cuyo *deadline* caduca en el slot actual. Se pasa de nuevo al paso 22.

31. En caso de que no haya más mensajes con caducidad en el LSW actual, procederemos a tratar el resto de los mensajes. Para ello se van evaluando uno a uno con un bucle *while* hasta que terminemos con todos los mensajes que tenemos que evaluar o nos obliguen a abandonar el bucle advirtiéndonos con un booleano. Si hay que abandonar el bucle se pasara al paso 22. Esta parte del algoritmo queda reflejada en la figura 5.31.

32. Se comprueba si el mensaje que se quiere evaluar puede meterse dentro del LSW actual. En caso que no sea posible se accede al paso 34

33. En caso que se comprobara con las distintas variables que nos hemos quedado en un bucle que nos impide rellenar el LSW sobrante, procederemos a pasar al siguiente LSW.

a) Se pasaría al siguiente LSW

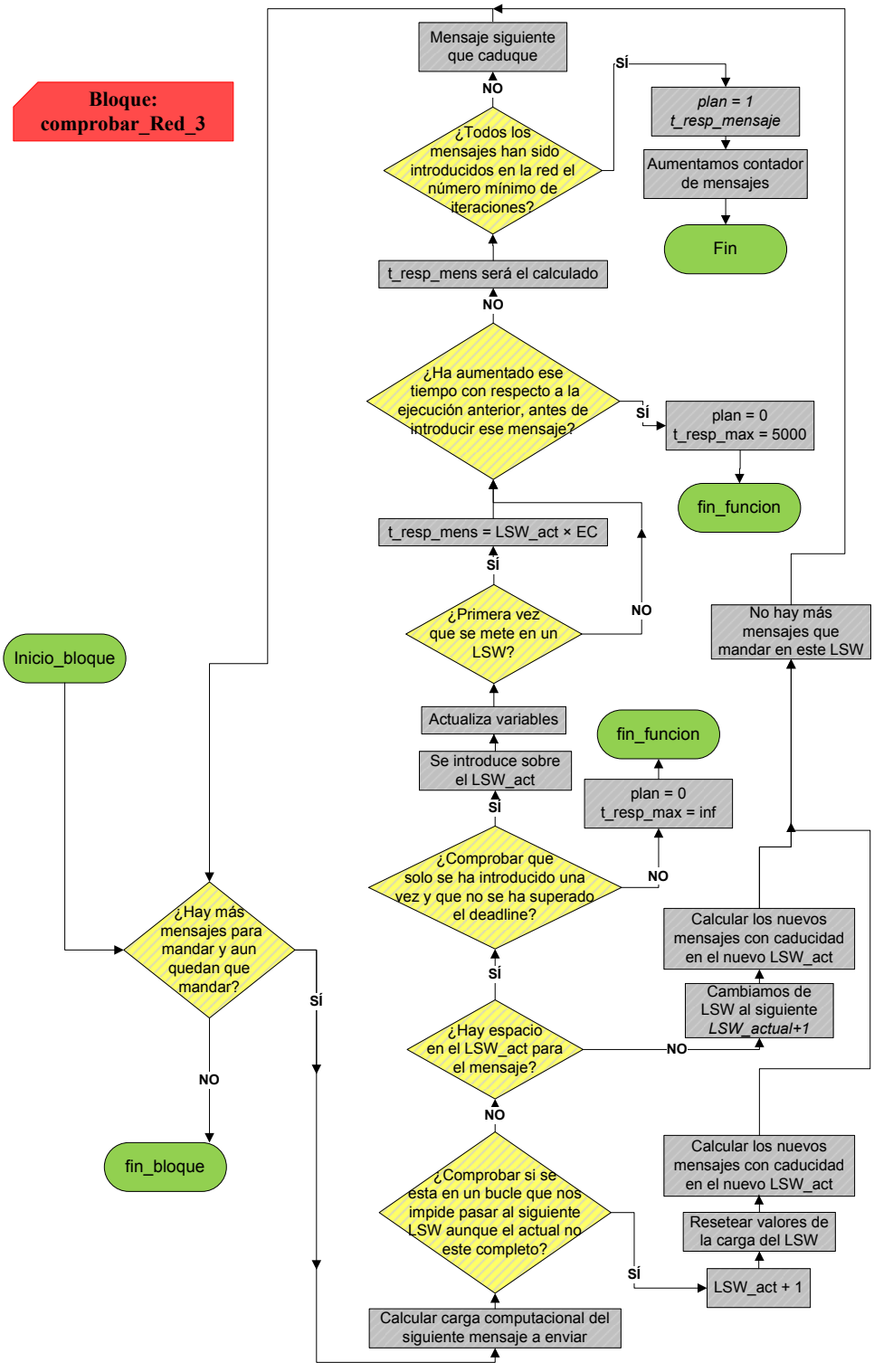

Figura 5.31: Diagrama bloques de la función *comprobar_Red.m* (parte 3)

b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

d) Actualizamos los *flags* para poder realizar nuevas comprobaciones.

e) Como al final del bucle *while* se va a pasar al siguiente mensaje, se resta una unidad de la variable que recorre todos los mensajes, para que continuemos evaluando el mensaje con el que estábamos y con el que tuvimos que pasar al siguiente LSW.

f) Accedemos al paso 31.

34. Si aun se puede seguir usando el LSW actual. Se comprueba que el mensaje actual cabe dentro del LSW actual. Se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no nos hayamos pasado el *deadline* de este. Si nos lo hemos pasado ya podremos devolver que el mensaje no será planificable y que su tiempo de respuesta será infinito, terminando la función de comprobación de la red. Si no cupiese se pasará al paso 35.

    a) si todo lo anterior se cumple procederemos a introducir el mensaje sobre el LSW actual. Con lo que se aumentará el tiempo de computación del mismo sumándole el tiempo de computación del nuevo mensaje a introducir, y actualizando las variables que almacenan el último mensaje introducido.

    b) Se incluye el tiempo de respuesta en el vector de tiempos de respuestas y se comprueba que ningún tiempo de respuesta de los mensajes antes almacenados se haya visto incrementado. Si no se ven incrementados se incluye el tiempo de respuesta en el cuarto parámetro del mensaje almacenado. En caso de haber aumentado se declarará la red como no planificable y se establecerá que su tiempo de respuesta es infinito, pudiendo finalizar la función de comprobación de la red.

35. Será necesario cambiar de LSW porque ya no hay más espacio.

    a) Se pasaría al siguiente LSW

    b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

d) Actualizamos los *flags* para poder realizar nuevas comprobaciones. Activaremos la *flag* que nos obliga a salir en el paso 31.

e) Como al final del bucle *while* se va a pasar al siguiente mensaje, se resta una unidad de la variable que recorre todos los mensajes, para que continuemos evaluando el mensaje con el que estábamos y con el que tuvimos que pasar al siguiente LSW.

36. Llegado a este punto los distintos mensajes se han introducido en la red. Si el número de veces que se ha introducido cada uno es igual o superior al número de iteraciones en las que se debía introducir debido a su *deadline* y que se calculo en 18, entonces se puede proceder a declarar la red como planificable. En caso que aun no se haya alcanzado ese nivel se procederá a seguir comprobando nuevos mensajes para los LSW restantes, para ello vamos al paso 37.

a) Se pone la variable de salida $plan = 1$ ya que el conjunto de mensajes si será planificable

b) Se devuelve el tiempo de respuesta del conjunto de los mensajes que indicarán el tiempo que necesita la red para transmitir ese mensaje.

c) Al haberse comprobado que sí que es planificable se aumenta el puntero para así indicar que hay un nuevo mensaje y modificamos el buffer auxiliar con la nueva configuración.

d) Volvemos a incluir la red modificada en el buffer redes auxiliar.

e) y nos salimos de la función ya que hemos terminado con las comprobaciones.

37. Procedemos a actualizar la variable que controla todos los mensajes de la red aumentándola en una unidad. Se pasa de nuevo al paso 31.

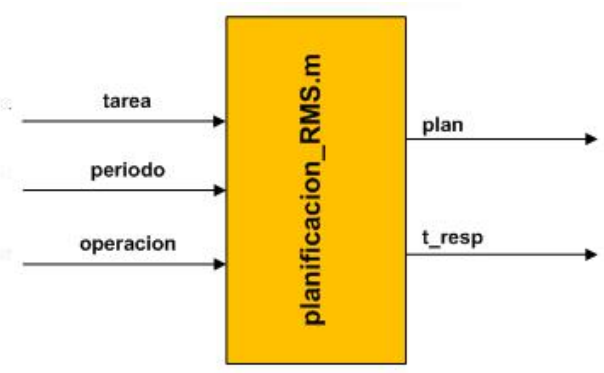

Figura 5.32: Parámetros de entrada/salida de la función $planificacion\_RMS.m$

5.2.8. Planificación RMS

5.2.8.1. Introducción

Esta función se encarga de añadir una tarea sobre el nodo físico solicitado y realizar las comprobaciones para determinar si la nueva tarea será planificable dentro de ese nodo. Aunque se ha realizado ya la comprobación de su planificación, y en un principio se tiene constancia de que ésta va a ser planificable, ya que a la hora de elegir la selección de nodos y redes más óptima se ha realizado ya esta comprobación. Aun así, se vuelve a comprobar para asegurarse que no han cambiado las características de los nodos físicos. Se ha tomado la decisión que una vez introducido un mensaje en una red física, este no podrá ser eliminado.

5.2.8.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

$>>$ `[plan,`$t_r esp$`]=`$planificacion\_RMS$ `(tarea,periodo,operacion)`

donde los parámetros de entrada son los siguientes:

- **tarea:** será una matriz de 1 sola fila y 2 columnas, que incluirán el tiempo de computación del mensaje, así como el nodo físico en el que se debe incluir el mensaje. Cada 'tarea' estará compuesta por [C_i Nodofísico].

- **periodo:** Al tratarse de una tarea periódica y trabajar sobre RMS, el periodo y el *deadline* coincidirán, por lo que es el tiempo máximo de respuesta del nodo y que será tomado como el tiempo máximo de respuesta de la nueva tarea que quiere ser introducida en la red.

- **operación:** Parámetro que se refiere al código de la operación que se quiere realizar sobre ese nodo. Lo que se pretende con este parámetro es que al final cuando se muestran al usuario las configuraciones, el usuario tenga más fácil distinguir las operaciones que se han realizado sobre los diferentes nodos físicos. En este caso los códigos de operación se distribuyen de la siguiente manera:

  - **'1'** se corresponderá con una operación de red 'A'
  - **'2'** se corresponderá con una operación de red 'B'
  - **'3'** se corresponderá con una operación de red 'C'
  - **'4'** se corresponderá con una operación de red 'D'

y los parámetros de salida serán:

- **plan:** Booleano que nos indica si el mensaje introducido ha sido planificable sobre el nodo auxiliar físico que se pedía. En caso afirmativo se devolverá un '1', en caso negativo un '0'.

- $t_r esp$**:** tiempo que necesita la tarea para ser ejecutada en el nodo, este tiempo dependerá de lo cargado que esté el nodo con las tareas previas.

5.2.8.3. Explicación detallada

A continuación se explicará detalladamente el código de la función $planificacion\_RMS.m$ cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.33.

1. Se añaden las variables globales que vamos a necesitar.

2. Se inicializan las variables de salida

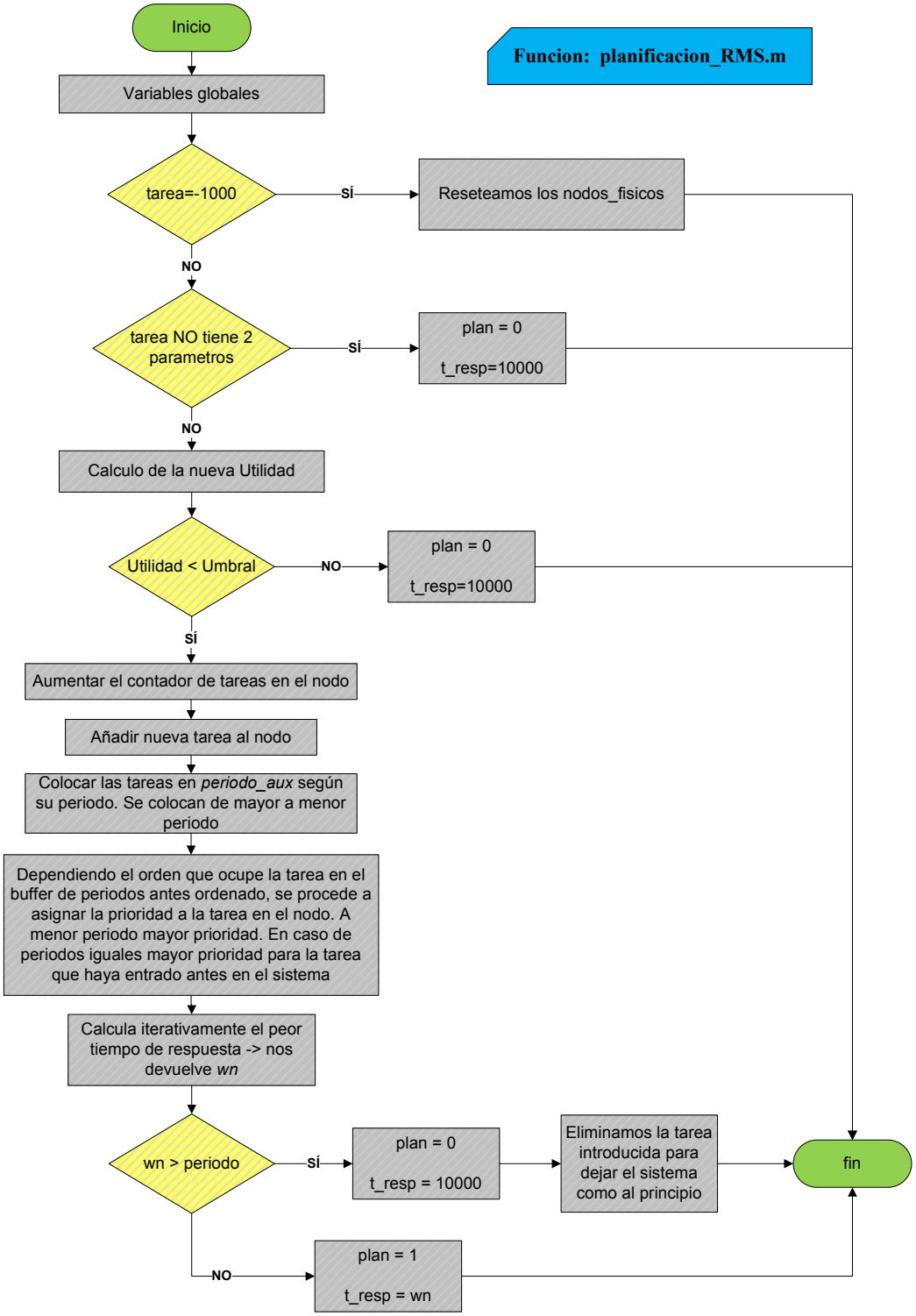

Figura 5.33: Diagrama bloques de la función $planificacion\_RMS.m$

3. Si el mensaje introducido es '-1000', entonces se resetean los búferes globales (nodos físicos) que estamos usando. Esto se realizará a la hora de iniciar un sistema, si queremos que se simule como si los procesadores estuviesen totalmente desocupados. Si no se quieren resetear los nodos se pasa al siguiente paso.

4. Si no se pasan al menos dos parámetros por tarea, esta será incorrecta y se devolverá un mensaje de error.

5. Calcularemos la utilización del nodo físico en cuestión. Para ello usamos la variable global en la que tenemos almacenada las diferentes utilidades de los nodos físicos con las tareas ya incluidas en ellos. U será la variable global que contenga las utilidades de las tareas ya incluidas anteriormente en los nodos físicos. El cálculo de la utilidad del nodo viene definida como el sumatorio de los tiempos de computación de cada tarea partido los periodos de las mismas. Es decir, la utilidad acumulada en la variable global más el tiempo de computación de la nueva tarea partido por su periodo.

6. Se comprueba si la Utilidad del nodo no supera el Umbral máximo establecido. En caso que se supere se devolverá un error advirtiendo que la tarea no es planificable.

7. Al tener una nueva tarea aumentaremos el número de tareas en la variable global del nodo físico que deseamos usar.

8. En primer lugar añadiremos la tarea al nodo físico. Consideraremos una prioridad común ('1') para todas ellas, que posteriormente modificaremos. Al estar trabajando sobre RMS tanto el periodo como el *deadline* son iguales. Cabe recordar que una tarea que se quiera introducir en el nodo viene definida por:

   - `[Ci Ti Di Pr Op]` es decir [T.computación Periodo Deadline Prioridad Operación]

9. Se procede a colocar las tareas dependiendo de su periodo para seleccionar su prioridad. Para ello se calculan las prioridades de cada tarea. Las ya existentes se actualizan debido a la llegada de la nueva tarea.

   - En el caso que sólo existiese una tarea se le colocaría la prioridad '1' y se pasaría al paso 10.

- Si existen más tareas las iremos ordenando por su periodo, ya que a menor periodo, mayor será su prioridad, como se explicó en la ecuación 2.4.

10. Como ya tenemos las tareas ordenadas por su periodo en el buffer de los periodos, ahora procederemos a añadirle su periodo en el nodo físico dependiendo de su posición en el buffer de los periodos. Como sabemos a menor periodo mayor prioridad y en caso de igualdad, tendrá mayor prioridad el que haya entrado antes en el sistema.

11. Finalmente sólo queda calcular el tiempo de respuesta, que se hará de manera iterativa.

12. Iterativamente iremos calculando los tiempos de respuesta, hasta que se cumpla que los tiempos de respuesta de dos iteraciones consecutivas coincidan entre sí. Para el cálculo del tiempo de respuesta se usará la ecuación presentada en 2.19.

13. En caso de haber evaluado 50 iteraciones y no haber encontrado dos iteración consecutivas con el mismo tiempo de respuesta, se puede afirmar que el tiempo de respuesta será infinito, y por lo tanto la tarea no será planificable en el sistema.

14. Una vez obtenido el tiempo de respuesta, se comprobará que este no supera el *deadline* del nodo, ya que de ser así la tarea tampoco sería planificable.

15. Llegado a este punto, se podrá afirmar que la tarea es planificable (plan=1), que el tiempo de respuesta será el obtenido en la iteración y que ha sido satisfactoriamente añadida al nodo físico deseado.

5.2.9. Planificación Red

5.2.9.1. Introducción

Esta función se encarga de añadir un mensaje en la red seleccionada, comprobando asimismo si este seria planificable. Aunque se realice la comprobación de su planificación, en un principio se tiene constancia de que esta va a ser planificable, porque a la hora de elegir la selección de nodos y redes más óptima se ha realizado ya esta comprobación. Aun así, se vuelve a comprobar para asegurarse que no han cambiado las características de la

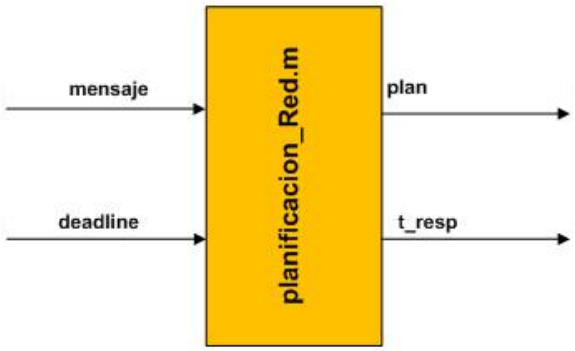

Figura 5.34: Parámetros de entrada/salida de la función *planificacion_Red.m*

red. Se ha tomado la decisión de que una vez introducido un mensaje en una red física, este no podrá ser eliminado.

Asimismo se ha implementado en la función, que un mensaje añadido anteriormente en tiempo no se pueda ver modificado por un mensaje añadido posteriormente en tiempo, es decir, la red será no planificable si un mensaje ya incluido viese su tiempo de respuesta aumentado debido a la inclusión de un mensaje posterior.

5.2.9.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> [plan,t_resp_mens]=planificacion_Red (mensaje,deadline)
```

donde los parámetros de entrada son los siguientes:

- **mensaje:** será una matriz de 1 sola fila y 2 columnas, que incluirán el tiempo de computación del mensaje, así como la red física en la que se debe incluir el mensaje.
- **deadline:** Tiempo máximo de respuesta de la aplicación y que será tomado como el tiempo máximo de respuesta del nuevo mensaje que quiere ser introducido en la red.

y los parámetros de salida serán:

- **plan:** Booleano que nos indica si el mensaje introducido ha sido planificable sobre la red física que se pedía. En caso afirmativo se devolverá un '1', en caso negativo un '0'.

- $t_r esp_m ens$**:** tiempo que necesita el mensaje para ser mandado por la red, este tiempo dependerá de lo cargado que este la red con mensajes previos. Este tiempo es función de los tiempos de EC que sean necesarios usar.

5.2.9.3. Explicación detallada

A continuación se explicará detalladamente el código de la función $planificacion\_Red.m$ cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.35.

1. Se añaden las variables globales que vamos a necesitar usar.

2. Se inicializan las variables de salida

3. Si el mensaje introducido es '-1000', entonces se resetean los búferes auxiliares que estamos usando. Para ello, igualamos variables auxiliares al estado de las variables de las redes físicas, y se finaliza la función. Si no se quieren resetear los búferes se pasa al siguiente paso.

4. Se vuelca el identificador de la red física a utilizar, incluido en el mensaje por parametro, sobre una variable.

5. En primer lugar tratamos el *deadline* para adaptarlo a los sistemas de LSW y EC. Es decir los *deadlines* de las redes trabajarán sobre formato EC, para lo cual dividimos por LSW que es el tiempo que podemos transmitir en un EC.

6. Añadimos a la red física, el nuevo mensaje junto con el *deadline* de la aplicación y el identificador de operación. Cada Mensaje tendrá [C_i deadline num_mens t_resp_mens].

7. Para hacer más fácil el uso de buffer de la red a usar procedemos a volcarlo sobre un búfer auxiliar de 2 dimensiones.

8. Creamos la variable que nos indicará el número de mensaje en la red. Será su identificador para poderlo encontrar en la misma y así poder devolver su tiempo de respuesta.

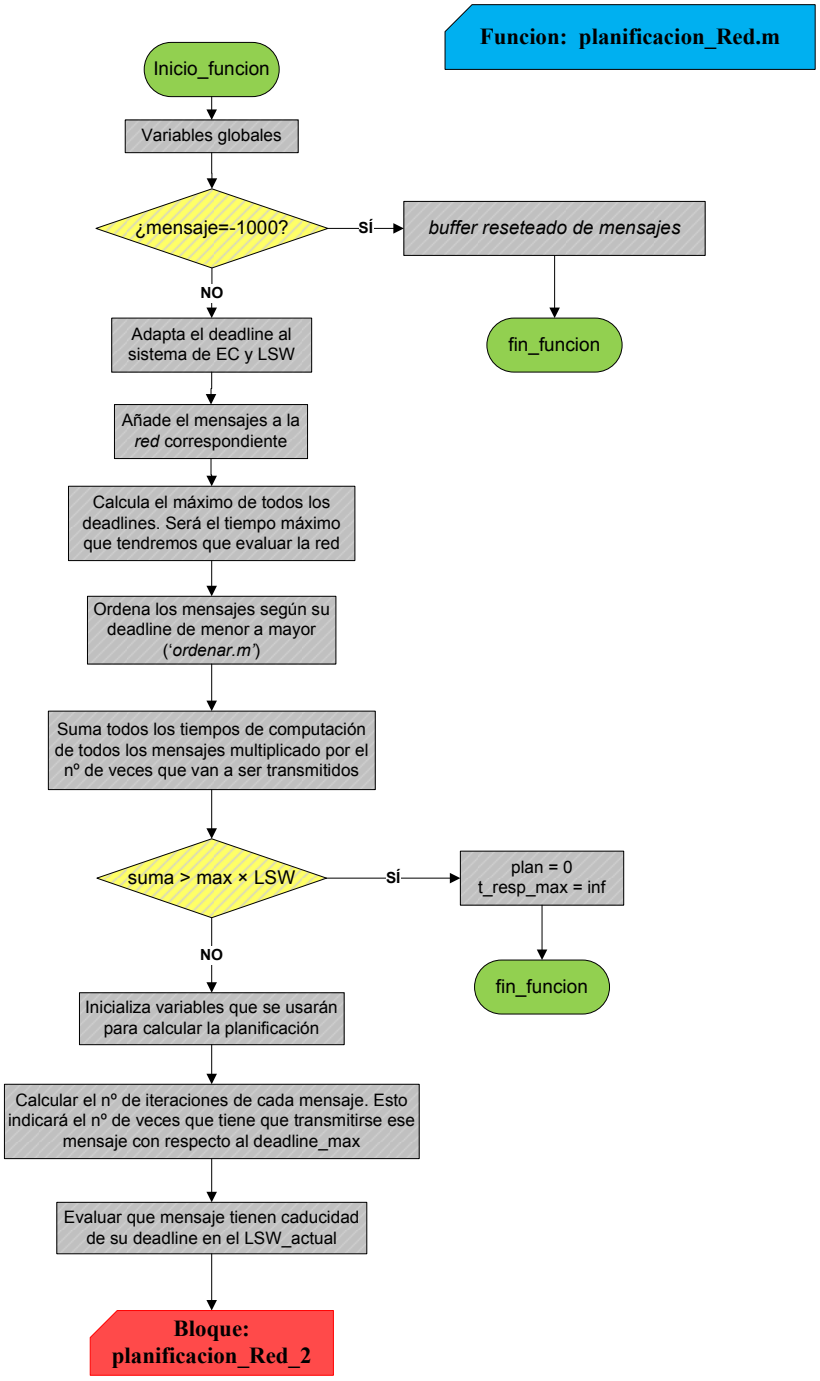

Figura 5.35: Diagrama bloques de la función *planificacion_Red.m*

9. Creamos una variable que tendrá una longitud igual al número de mensajes que haya en la red. Se usará para mostrar los tiempos de respuesta de los distintos mensajes.

10. Búfer que usaremos para almacenar los mensajes cuyo *deadline* caduca en el presente LSW. Con ello evitaremos que se nos pase un mensaje cuya caducidad nos obligaría a declarar la red como no planificable. Con este sistema se da prioridad a los mensajes cuyo *deadline* coincide con el LSW que se está ocupando en cada momento.

11. Calculamos el máximo de todos los deadlines de las tareas, que ya están expresadas en función de LSW, por lo que lo que se está calculando es el máximo de LSWs que hay que comprobar.

12. Antes de operar con los mensajes procedemos a colocarlos según su *deadline* de menor a mayor. En caso de tener el mismo *deadline* se procederá a que se sitúe primero el de menos carga computacional tenga. Para ello se llama a la función ordenar.

13. Primero comprobamos si la suma de los tiempos de computación de todos los mensajes que tenemos que transmitir sobre todos los LSW, es posible transmitirlos usando los LSW del máximo antes calculado. En caso contrario ya podremos indicar que esta red no va a ser planificable.

14. Creamos la variable que nos va a indicar sobre que LSW nos encontramos actualmente.

15. Se crea un vector que muestre la carga computacional introducida en cada LSW.

16. Se crea un booleano para distinguir entre un LSW nuevo y uno sobre el que ya se ha introducido algún mensaje.

17. Vector que nos indica el número de veces que se ha usado ese mensaje. Al estar tratando con mensajes periódicos tendremos que saber el número de ejecuciones que existen de cada mensaje, para contemplar si es necesario otra nueva ejecución con respecto a su *deadline*.

18. Calculo el número de iteraciones de cada mensaje. Esto nos indicará el número de veces que ha de transmitirse ese mensaje con respecto al *deadline* máximo. Exclu-

sivamente va a depender del *deadline* del mensaje y del *deadline* máximo de todos los mensajes de la red.

19. Creación de variables para almacenar el primer elemento que se incluyo en ese LSW para así poder cambiar de LSW en el caso que ningún mensaje pueda ser transmitido en el actual. Es decir que se recorriesen todos los mensajes existentes y volviésemos al almacenado sin haber cambiado de LSW.

20. Bucle para evaluar cuales son los mensajes cuya caducidad corresponden con la del primer LSW. Estos mensajes se ha decidido que tengan prioridad sobre el resto ya que en caso de superar su *deadline* será crítico para la planificación.

21. Creación de diferentes variables para poder gestionar los mensajes al introducirlos a la red y comprobar su planificación.

    Este parte del algoritmo la podemos seguir en la figura 5.30 que aunque forma parte de la función *comprobar_Red*, sirve para entender las explicaciones de la función que estamos tratando.

22. A partir de este punto procedemos a procesar los mensajes a través de la red. Creamos un bucle que se parará en el caso que el LSW que procedamos a evaluar sea superior al máximo de LSWs que tenemos que procesar y que se han calculado en el paso 11.

23. Comprobamos si se diese el caso de haber recorrido todos los mensajes, si es así, se fuerza para volver a empezar con las comprobaciones.

24. Se comprueba si existen mensajes cuya caducidad corresponda con el LSW actual, los cuales tendrán mayor prioridad. En caso que no existan más mensajes que caduquen en este LSW se accede al paso 31.

25. Guardamos la posición del mensaje que se iba a comprobar antes de que existiesen los de mayor prioridad (los que caducan en este LSW), para una vez terminados estos seguir el orden en el que se dejó la ejecución.

26. Se coge como próximo mensaje a evaluar uno de los que caducan en el LSW actual.

27. En caso que se comprobara con las distintas variables que nos hemos quedado en un bucle que nos impide rellenar el LSW sobrante, procederemos a pasar al siguiente LSW. Si aun se puede seguir usando el LSW actual se pasa al paso 28.

    a) Se pasaría al siguiente LSW

    b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

    c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

    d) Actualizamos los flags para poder realizar nuevas comprobaciones.

    e) Accedemos al paso 22.

28. Si aun se puede seguir usando el LSW actual. Se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no nos hayamos pasado el *deadline* de este. Si nos lo hemos pasado ya podremos devolver que el mensaje no será planificable y que su tiempo de respuesta será infinito, terminando la función de comprobación de la red.

    a) si todo lo anterior se cumple procederemos a introducir el mensaje sobre el LSW actual. Con lo que se aumentará el tiempo de computación del mismo sumándole el tiempo de computación del nuevo mensaje a introducir, y actualizando las variables que almacenan el último mensaje introducido.

    b) Se incluye el tiempo de respuesta en el vector de tiempos de respuestas y se comprueba que ningún tiempo de respuesta de los mensajes antes almacenados se haya visto incrementado. Si no se ven incrementados se incluye el tiempo de respuesta en el cuarto parámetro del mensaje almacenado. En caso de haber aumentado se declarará la red como no planificable y se establecerá que su tiempo de respuesta es infinito, pudiendo finalizar la función de comprobación de la red.

29. Llegado a este punto los distintos mensajes se han introducido en la red. Si el número de veces que se ha introducido cada uno es igual o superior al número de iteraciones en las que se debía introducir debido a su *deadline* y que se calculo en 18, entonces se puede proceder a declarar la red como planificable. En caso que aun no se haya

alcanzado ese nivel se procederá a seguir comprobando nuevos mensajes para los LSW restantes, para ello vamos al paso 30.

a) Se pone la variable de salida $plan = 1$ ya que el conjunto de mensajes si será planificable

b) Se devuelve el tiempo de respuesta del conjunto de los mensajes que indicarán el tiempo que necesita la red para transmitir ese mensaje.

c) Al haberse comprobado que sí que es planificable se aumenta el puntero para así indicar que hay un nuevo mensaje y modificamos el buffer auxiliar con la nueva configuración.

d) Volvemos a incluir la red modificada en el buffer redes auxiliar.

e) y nos salimos de la función ya que hemos terminado con las comprobaciones.

30. Procedemos a actualizar las variables. Aumentamos el índice que controla el búfer con las variables que caducan en este LSW. Asimismo actualizamos la variable que controla todos los mensajes de la red y la igualamos al último que se dejo sin evaluar en el momento que se paso a los mensajes cuyo *deadline* caduca en el slot actual. Se pasa de nuevo al paso 22.

31. En caso de que no haya más mensajes con caducidad en el LSW actual, procederemos a tratar el resto de los mensajes. Para ello se van evaluando uno a uno con un bucle *while* hasta que terminemos con todos los mensajes que tenemos que evaluar o nos obliguen a abandonar el bucle advirtiéndonos con un booleano. Si hay que abandonar el bucle se pasara al paso 22. Esta parte es similar a la explicada en la función *comprobar_Red*, la cual quedaba reflejada sobre la figura 5.31.

32. Se comprueba si el mensaje que se quiere evaluar puede meterse dentro del LSW actual. En caso que no sea posible se accede al paso 34

33. En caso que se comprobara con las distintas variables que nos hemos quedado en un bucle que nos impide rellenar el LSW sobrante, procederemos a pasar al siguiente LSW.

a) Se pasaría al siguiente LSW

b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

d) Actualizamos los *flags* para poder realizar nuevas comprobaciones.

e) Como al final del bucle *while* se va a pasar al siguiente mensaje, se resta una unidad de la variable que recorre todos los mensajes, para que continuemos evaluando el mensaje con el que estábamos y con el que tuvimos que pasar al siguiente LSW.

f) Accedemos al paso 31.

34. Si aun se puede seguir usando el LSW actual. Se comprueba que el mensaje actual cabe dentro del LSW actual. Se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no nos hayamos pasado el *deadline* de este. Si nos lo hemos pasado ya podremos devolver que el mensaje no será planificable y que su tiempo de respuesta será infinito, terminando la función de comprobación de la red. Si no cupiese se pasará al paso 35.

    a) si todo lo anterior se cumple procederemos a introducir el mensaje sobre el LSW actual. Con lo que se aumentará el tiempo de computación del mismo sumándole el tiempo de computación del nuevo mensaje a introducir, y actualizando las variables que almacenan el último mensaje introducido.

    b) Se incluye el tiempo de respuesta en el vector de tiempos de respuestas y se comprueba que ningún tiempo de respuesta de los mensajes antes almacenados se haya visto incrementado. Si no se ven incrementados se incluye el tiempo de respuesta en el cuarto parámetro del mensaje almacenado. En caso de haber aumentado se declarará la red como no planificable y se establecerá que su tiempo de respuesta es infinito, pudiendo finalizar la función de comprobación de la red.

35. Será necesario cambiar de LSW porque ya no hay más espacio.

    a) Se pasaría al siguiente LSW

    b) Se resetearían los valores de carga del LSW con el que estamos trabajando.

c) Calcularemos los nuevos mensajes con caducidad en el nuevo LSW.

d) Actualizamos los $flags$ para poder realizar nuevas comprobaciones. Activaremos la $flag$ que nos obliga a salir en el paso 31.

e) Como al final del bucle *while* se va a pasar al siguiente mensaje, se resta una unidad de la variable que recorre todos los mensajes, para que continuemos evaluando el mensaje con el que estábamos y con el que tuvimos que pasar al siguiente LSW.

36. Llegado a este punto los distintos mensajes se han introducido en la red. Si el número de veces que se ha introducido cada uno es igual o superior al número de iteraciones en las que se debía introducir debido a su *deadline* y que se calculo en 18, entonces se puede proceder a declarar la red como planificable. En caso que aun no se haya alcanzado ese nivel se procederá a seguir comprobando nuevos mensajes para los LSW restantes, para ello vamos al paso 37.

a) Se pone la variable de salida $plan = 1$ ya que el conjunto de mensajes si será planificable

b) Se devuelve el tiempo de respuesta del conjunto de los mensajes que indicarán el tiempo que necesita la red para transmitir ese mensaje.

c) Al haberse comprobado que sí que es planificable se aumenta el puntero para así indicar que hay un nuevo mensaje y modificamos el buffer auxiliar con la nueva configuración.

d) Volvemos a incluir la red modificada en el buffer redes auxiliar.

e) y nos salimos de la función ya que hemos terminado con las comprobaciones.

37. Procedemos a actualizar la variable que controla todos los mensajes de la red aumentándola en una unidad. Se pasa de nuevo al paso 31.

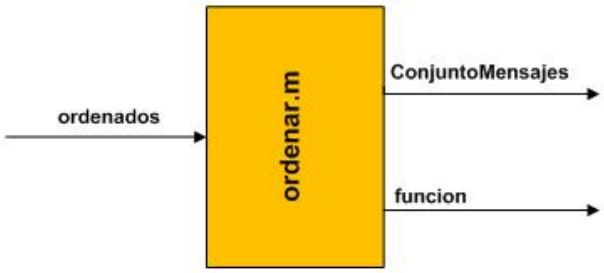

Figura 5.36: Parámetros de entrada/salida de la función *ordenar.m*

5.2.10. Ordenar

5.2.10.1. Introducción

Esta función se encarga de ordenar el conjunto de tareas/mensajes que recibe por parámetro con respecto a su tiempo de computación o respecto a su *deadline*. Para ello irá comparando todos las tareas/mensajes que reciba en el conjunto de mensajes, colocándolos en orden creciente de tiempos de computación/*deadlines*, obteniendo finalmente un conjunto de tareas/mensajes ordenados de menor a mayor tiempo de computación/*deadline*.

5.2.10.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

```
>> [ordenados]=ordenar(ConjuntoMensajes, funcion)
```

donde los parámetros de entrada son:

- **ConjuntoMensajes:** matriz que contendrá las distintas tareas o mensajes que queremos ordenar.
- **función:** con la variable *funcin* lo que se distingue es saber si lo que se está ordenando son tareas o mensajes, ya que tendrán diferente algoritmo para cada uno de ellos. Tendrá como posibles valores 'mensaje' o 'tareas'.

y el parámetro de salida:

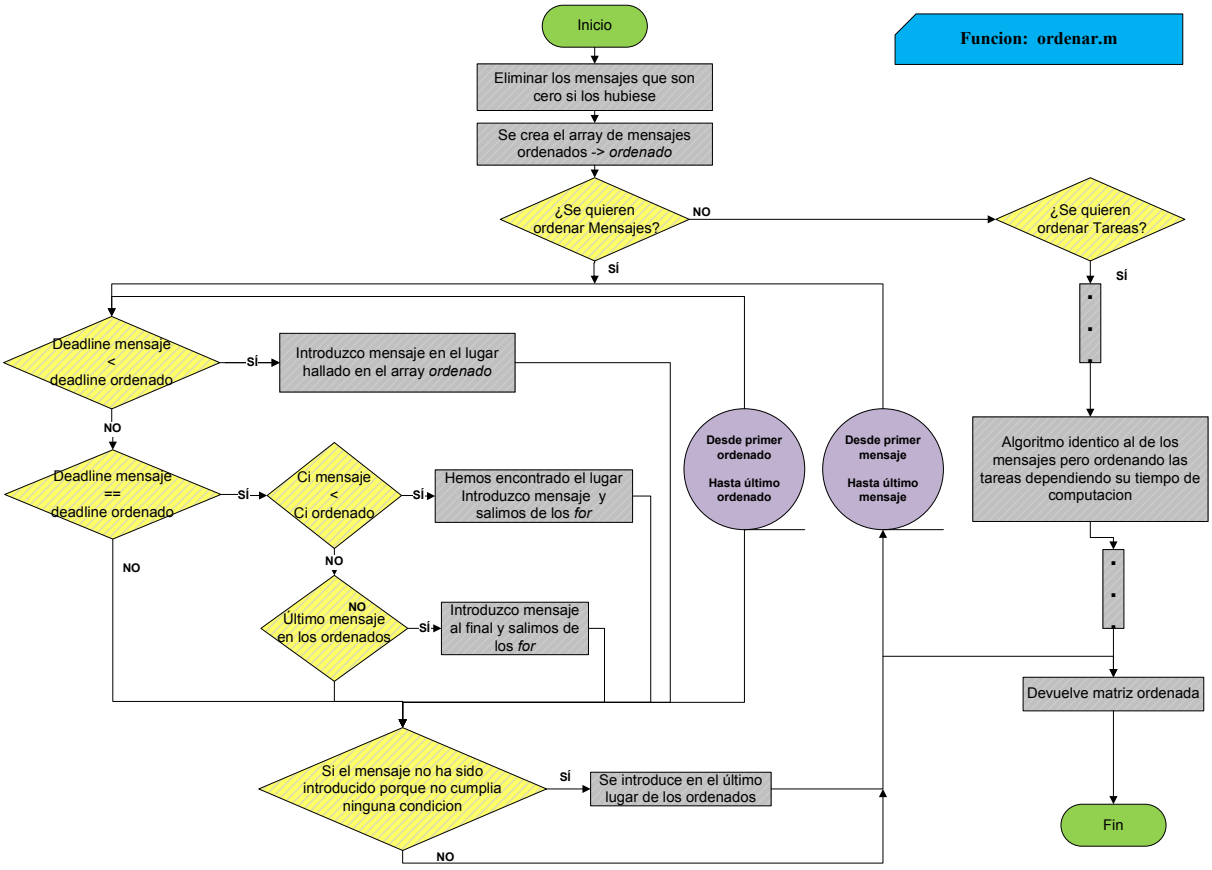

Figura 5.37: Diagrama bloques de la función *ordenar.m*

- **ordenados:** matriz con los mensajes recibidos por parámetro pero una vez ordenadas de menor a mayor *deadline*.

5.2.10.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *ordenar.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.37.

1. En primer lugar se procede a quitar los mensajes que sean cero del conjunto de mensajes para poder trabajar con ellos. En caso de tener algún mensaje que fuera cero siempre sería menor que el resto que quisiéramos evaluar, por lo tanto la ordenación de los mismos sería incorrecta. Se procede a deshacerse de estos mensajes de todo cero debido a que es imposible que exista una tarea/mensaje con esas características.

2. Si el parámetro $funcin$ indica que se quieren ordenar mensajes ('mensaje') procederemos a ello. En caso contrario, nos indicará que quiere ordenar tareas ('tareas') paso 9.

3. Declaramos una variable-vector aux que en primer lugar almacenará el primer mensaje, y que posteriormente irá almacenando todos los mensajes ya ordenados

4. Declaramos dos bucles for que se usarán para ordenar los distintos mensajes dependiendo de su *deadline*. Para ello se procede a estudiar su *deadline* que se encuentra en la posición segunda de cada mensaje.

   El primer bucle for recorrerá todos los mensajes que hemos recibido en el parámetro $ConjuntoMensajes$.

   El segundo bucle for irá recorriendo los ya ordenados, que se encuentran en la variable aux.

5. Se irá comprobando si el *deadline* del que se quiere ordenar es menor que cada uno de los ya ordenados, hasta encontrar su lugar. En tal caso se introducirá en el vector aux en la posición hallada.

6. En caso de que existan dos mensajes con el mismo *deadline*, se procederá a ordenarlos teniendo en cuenta su tiempo de computación. En este caso, a menor tiempo de computación irá antes en la ordenación de los mensajes.

7. Si se llegase al fin del vector aux, y el mensaje a ordenar no hubiese sido introducido, significa que es el de *deadline* mayor, por lo tanto se coloca al final del vector aux.

8. Una vez completado el primer bucle for, tendremos todos los mensajes ordenados por lo que podremos finalizar la función, paso 16.

9. En caso que lo que se quiera ordenar sean tareas, se seguirá un planteamiento similar al de los mensajes. En este caso el criterio de ordenación sería el tiempo de computación de cada tarea.

10. Declaramos una variable-vector aux que en primer lugar almacenará el primer mensaje, y que posteriormente irá almacenando todos los mensajes ya ordenados.

11. En este caso también habría que declarar la variable-vector *aux* para ir almacenando las tareas ya ordenadas. Asimismo se declaran los dos bucles *for* usados para ordenar los distintas tareas dependiendo de su *deadline*. Para ello estudiaremos el tiempo de computación que se encuentra en la posición primera de cada mensaje.

    El primer bucle *for* recorrerá todas las tareas que hemos recibido en el parámetro ConjuntoMensajes.

    El segundo bucle *for* irá recorriendo las que ya se encuentran ordenadas en la variable *aux*.

12. Se irá comprobando si el tiempo de computación del que se quiere ordenar es menor que el de las que ya están ordenadas, hasta encontrar su lugar. Una vez encontrado se introducirá en el vector *aux* en la posición hallada.

13. En caso de que existan dos tareas con el mismo tiempo de computación, se procederá a ordenarlas teniendo en cuenta su *deadline*. En este caso, a menor *deadline* irá antes en la ordenación de las tareas.

14. Si se llegase al fin del vector *aux*, y la tarea a ordenar no hubiese sido introducida, significa que es el de tiempo de computación mayor, por lo tanto se ha de colocar al final del vector *aux*.

15. Una vez completado el primer bucle *for*, ya tendremos todas las tareas ordenadas.

16. Creamos la variable de salida como copia de la variable de entrada y sustituimos las filas donde antes estaban desordenados, por las nuevas filas ordenadas, devolviendo así también las distintas filas que se nos pasaron con ceros.

5.2.11. Mostrar tiempo

5.2.11.1. Introducción

Esta función devuelve en segundos el resultado que se obtiene al invocar a la función 'clock' de *Matlab*. Un tiempo devuelto por 'clock' viene definido en 6 columnas:

[year month day hour minute seconds]

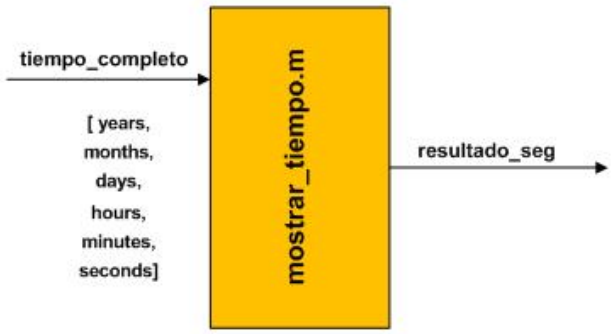

Figura 5.38: Parámetros de entrada/salida de la función *mostrar_tiempo.m*

5.2.11.2. Parámetros de entrada y de salida

Un ejemplo de llamada a la función sería el siguiente:

>> `[resultado]=` *mostrar_tiempo*(*tiempo_resultante*)

donde el parámetro de entrada como se ha definido antes ha de ser un *array* de seis columnas, que contenga:

- **year:** el número de años que se quieren transformar a segundos.
- **month:** el número de meses que se quieren transformar a segundos.
- **day:** el número de días que se quieren transformar a segundos.
- **hour:** el número de horas que se quieren transformar a segundos.
- **minute:** el número de minutos que se quieren transformar a segundos.
- **seconds:** el número de segundos que se quieren añadir a todo lo anterior.

y el parámetro de salida:

- *tiempo_resultante*: será el resultado de transformar las entradas a segundos.

5.2.11.3. Explicación detallada

A continuación se explicará detalladamente el código de la función *mostrar_tiempo.m* cuyo diagrama de bloques explicativo se puede encontrar en la figura 5.39.

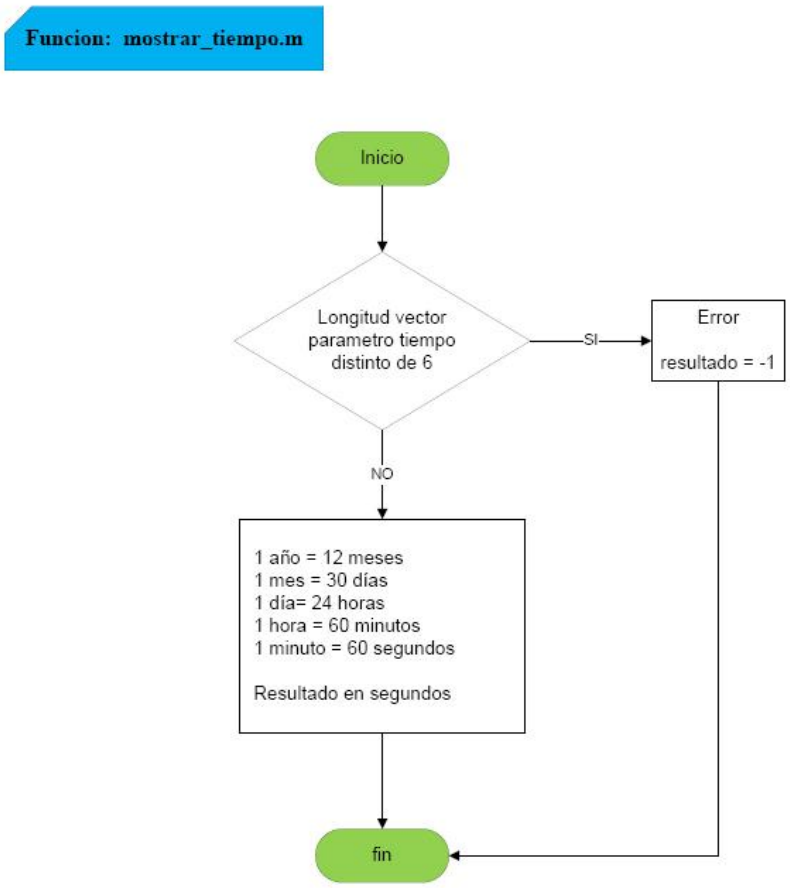

Figura 5.39: Diagrama bloques de la función *mostrar_tiempo.m*

- En primer lugar se comprueba que el parámetro recibido como entrada tiene seis columnas, en caso contrario se devuelve un error.
- Se realiza la transformación del tiempo, para lo cual:
  - Los años se transforman en meses (multiplicar por 12) y se añaden a los meses pasados por parámetro.
  - Los meses resultantes se pasan a días (multiplicar por 30) y se añaden a los dias del parámetro.
  - Los días obtenidos se multiplican por 24 para hacerlos horas y se añaden a las horas del parámetro.
  - Las horas que se obtienen se multiplican por 60 para hacerlos minutos y se añaden a los minutos del parámetro.
  - Finalmente estos minutos multiplicándolos por 60 se transforman en segundos y se añaden a los obtenidos en el parámetro, para así completar la conversión.
- Por último se devuelve el resultado obtenido en segundos.

5.3. Fichero Matlab

5.3.1. Fichero entrada

5.3.1.1. Introducción

Este archivo será un fichero de *Matlab* (*.m*), que se diferenciará de una función en que no recibirá al principio del archivo la definición de ('*function*'), como sí sucedía con el resto de los archivos antes evaluados. Este fichero será el encargado de contener la arquitectura y las características de la aplicación que el usuario desee evaluar en alguno de los sistemas desarrollados. Este archivo estará diseñado siguiendo un modelo en el cual el usuario será el encargado de completar los datos dependiendo de la aplicación que desee ejecutar. En este apartado vamos a expresar los campos que deberán ser rellenados por el usuario una vez haya decidido la arquitectura que desee evaluar. Se puede encontrar en la parte del Apéndice (B), el código de un fichero ejemplo, el cual podrá ser usado por el usuario para configurar sus aplicaciones.

5.3.1.2. Parámetros de entrada y de salida

Como hemos dicho anteriormente al no ser una función, no tendrá ningún parámetro ni de entrada ni de salida. A la hora de ser llamado, se ejecutará como cualquier función de *Matlab*. Sirva como ejemplo el siguiente:

$>>$ *fichero_3_servicios_con_paralelo*

Esto cargará en el Espacio de Trabajo (*WorkSpace*) de *Matlab*, las variables que contenga el fichero ejecutado. Nosotros no lo cargaremos directamente desde la consola de ejecución de *Matlab*, sino que será ejecutado por el sistema una vez éste sea llamado, por lo tanto y como se ha dicho en la parte de la explicación de los sistemas, el fichero será pasado como parámetro al sistema, pero entre comillas, para que así luego pueda ser ejecutado dentro del código del sistema elegido.

5.3.1.3. Explicación detallada

A continuación vamos a explicar la estructura que deberá poseer este fichero para que sea correctamente ejecutado por alguno de los sistemas que se han implementado. Para ello nos apoyaremos en la figura 5.41, que contiene un diagrama de bloque con los apartados fundamentales del fichero. En este apartado vamos a utilizar como ejemplo la aplicación siguiente, que corresponderá a una aplicación con cuatro servicios, tres de ellos en serie junto con un paralelo (figura 5.40).

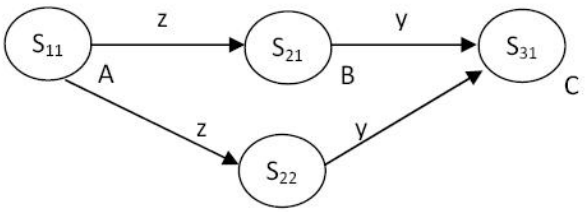

Figura 5.40: Esquema de la aplicación que se va a usar como ejemplo

1. Definición de variables globales obligatorias. Esta parte no podrá ser eliminada por el usuario. Se definen las siguientes variables globales:

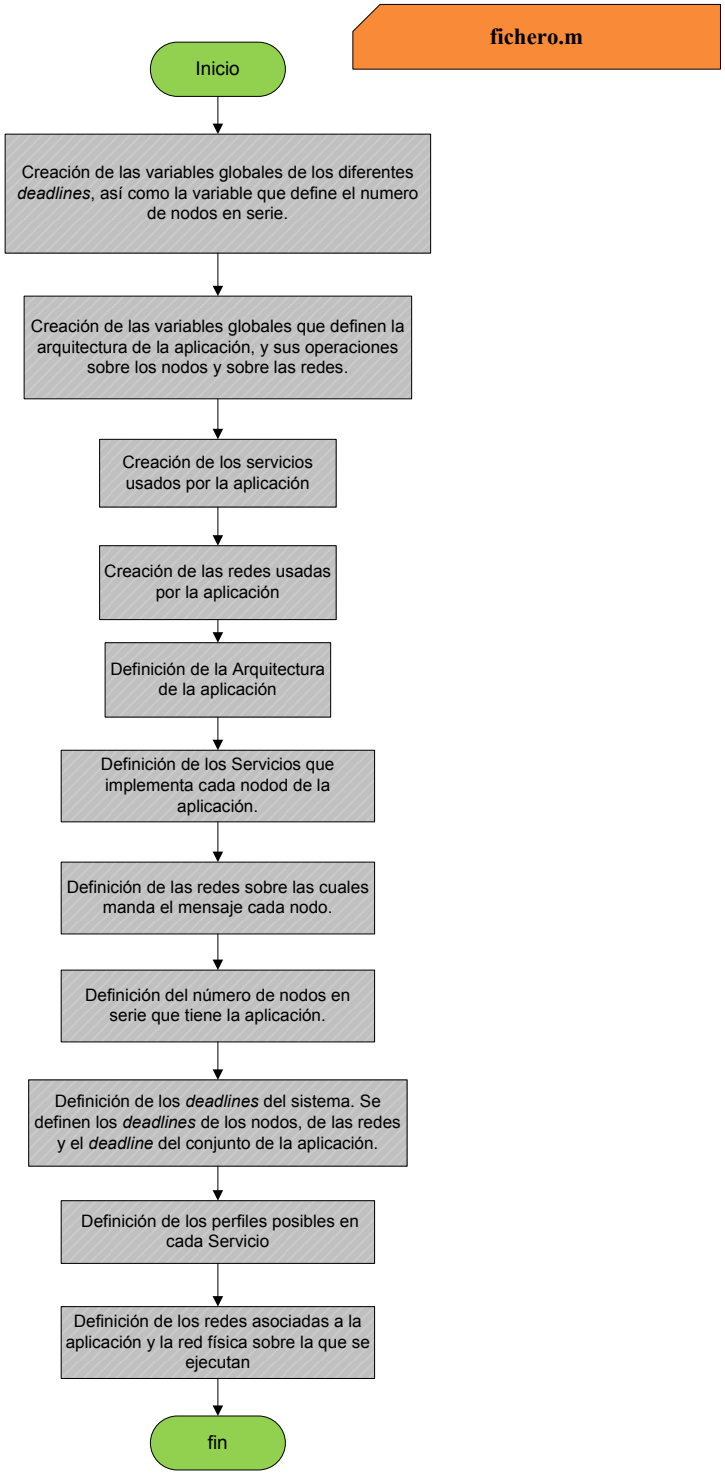

Figura 5.41: Diagrama bloques de cómo debe un usuario crear un fichero de ejecución

- `global deadline_aplicacion:` contendrá el plazo máximo de ejecución de la aplicación. Sera establecido por el usuario posteriormente.
- `global deadline_nodo:` contendrá el plazo máximo de ejecución de las tareas de la aplicación. Puede ser establecida por el usuario o se creará aleatoriamente dependiendo del número de nodos en serie que tenga la aplicación.
- `global deadline_red:` contendrá el *deadline* máximo de ejecución de un mensaje en la red. Será establecido por el usuario posteriormente.
- `global nodosSerie:` que contendrá el número de nodos en serie que tiene la aplicación. Su inicialización se realiza en el paso 6.

2. Definición de variables globales correspondientes a los *arrays* que van a definir tanto la arquitectura de la aplicación, así como los servicios que evaluará dicha aplicación y las redes a usar entre los distintos servicios.

   - `global Op_redes:` *array* que contiene las redes que se usarán entre cada servicio.
   - `global Arquitectura:` *array* que define como será la estructura de la aplicación.
   - `global Operaciones:` *array* que define los servicios que se usaran en la aplicación.

3. Se define la variable *Arquitectura* que contendrá la estructura de la aplicación. Estará compuesta por números que indicarán el siguiente nodo al que mandarán la información una vez haya sido ejecutado su servicio. Para hacer más sencilla la creación, por parte de un usuario de una nueva aplicación, estos números indicarán el número de columna y de fila que vendrá posteriormente a la evaluación del presente servicio. Por ejemplo, imaginemos que nos encontramos en el primer servicio de la aplicación. En ese caso nos referiremos a él como '11'. Si posteriormente a este servicio, viniese otro servicio en serie, tendríamos que indicar en la arquitectura sobre la posición en la tabla primera fila, primera columna, que después de su evaluación vendrá el '21' y así sucesivamente. En el caso de tener un paralelo, éste quedaría expresado en la segunda fila. En la aplicación que hemos tomado como ejemplo (figura 5.40) el *array*

correspondiente a la arquitectura podrá observarse en la tabla 5.1. Asimismo se define cual es el último nodo de la aplicación, lo cual se hace con la inserción del número '99' en la posición que corresponda con el último servicio. El número '0' indica que sobre esa posición no existe ningún nodo en la aplicación.

- `global Arquitectura`

$$\text{Arquitectura} = \begin{bmatrix} 21 & 31 & 99 & 0 & 0; \\ 0 & 31 & 0 & 0 & 0; \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tabla 5.1: Arquitectura de la aplicación usada como ejemplo

4. Se define la variable *Operaciones*, la cual definirá los servicios a realizar en cada uno de los nodos de la aplicación. Durante la realización del proyecto se han considerado la creación y uso de cuatro servicios. Estos serán definidos por las letras 'A', 'B', 'C' y 'D'. En caso de que el usuario desease la creación de más servicios, sólo sería necesario añadir nuevas variables globales con el nombre de los mismos (hágase en el paso 8), así como completar en el paso 10 la definición de los perfiles del nuevo servicio.

- `global Operaciones`

$$\text{Operaciones} = \begin{bmatrix} \text{'A'} & \text{'B'} & \text{'C'} & 0 & 0; \\ 0 & \text{'D'} & 0 & 0 & 0; \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tabla 5.2: Servicios usados por los nodos de la aplicación ejemplo

5. Se define la variable *Op_redes*, la cual definirá las redes a utilizar por cada uno de los distintos nodos de la arquitectura. Durante la realización del proyecto se han considerado la creación y uso de tres redes. Estas serán definidas por las letras 'z', 'y', 'w'. En caso que el usuario desease la creación de más redes, solo sería necesario añadir nuevas variables globales con el nombre de las mismas (hágase en el paso

9), así como completar en el paso 11 la definición del perfil asociado a cada red. El número '0' indica como en la 'Arquitectura' que para ese nodo no existe ninguna red asociada.

- `global Op_redes`

$$\text{Op\_redes} = \begin{bmatrix} \text{'z'} & \text{'y'} & 0 & 0 & 0; \\ 0 & \text{'w'} & 0 & 0 & 0; \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Tabla 5.3: Redes a usar por la aplicación ejemplo

6. Ahora ya podremos completar la variable global *nodosSerie*, ya que al conocer la arquitectura de la aplicación, sólo tendremos que recorrerla hasta encontrar la posición donde se encuentra el '99', que nos indica final de la aplicación.

7. Definición de los *deadlines* del sistema. Se definen los *deadlines* de los nodos, de las redes y el *deadline* del conjunto de la aplicación. Como dijimos en pasos anteriores, estos *deadlines* serán aleatorios ya que se persigue la aleatoriedad a la hora de crear los sistemas. Si el usuario lo desease podría establecerlos él mismo. En el caso del ejemplo que estamos desarrollando, estos *deadlines* son:

   - `deadline_nodo = round(20*(rand+1));`
   - `deadline_red=round(50*(rand)+nodosSerie*40);`
   - `deadline_aplicacion = round(500*(rand)+nodosSerie*40);`

8. Creación de las variables globales que identificarán a los cuatro servicios que se han decidido definir para esta aplicación. En caso de decidir añadir más servicios, sólo será necesario añadir nuevas variables globales y definir sus perfiles en el paso 10

   - `global A`
   - `global B`
   - `global C`
   - `global D`

A = [1 2
4 3
2 1
1 1]

B = [1 3
3 2
5 1
2 3]

C = [2 2
3 1
3 3
3 2]

D = [1 1
2 2
4 3
3 2]

Tabla 5.4: Definición de los perfiles de los diferentes Servicios

9. Creación de variables globales correspondientes a las redes de la aplicación. Aquí se deberán definir tantas variables globales como redes existan, en nuestro caso al trabajar con tres redes físicas, y al tener cada red física una red asociada, tendremos tres definiciones:

   - `global z` que correspondería a la red que se ejecuta sobre la red física 1.
   - `global y` que correspondería a la red que se ejecuta sobre la red física 2.
   - `global x` que correspondería a la red que se ejecuta sobre la red física 3.

10. Este es el paso en el cual se definen los perfiles de cada servicio. Para su definición será necesario incluir por cada fila el tiempo de computación de cada perfil y el nodo físico sobre el cual se ejecuta, es decir, cada fila debe contener [C_i, nodo_fisico]. Siguiendo con el ejemplo 5.40, una posible definición de los perfiles sería la que aparece reflejada en la tabla 5.4.

11. En este último paso se definen las redes creadas anteriormente. Para su definición será necesario incluir por cada fila el tiempo de computación de cada perfil y la red física sobre la cual se ejecuta ([C_i, nodo_fisico]). Siguiendo con el ejemplo 5.40, una posible definición de los perfiles sería la que aparece reflejada en la tabla 5.5. Como podemos observar el C_i se decidió hacerlo aleatorio para así favorecer la aleatoriedad de las redes del sistema para conseguir un estudio más efectivo del mismo.

z = [[round(4*(rand+0.25)) 1]

y = [[round(4*(rand+0.25)) 2]

w = [[round(4*(rand+0.25)) 3]

Tabla 5.5: Definición de las tres redes

5.4. Conclusiones

Para concluir el desarrollo de este capítulo, cabe destacar la utilidad del manual desarrollado. Con ello el futuro usuario tendrá más sencillo entender el programa implementado, así como su posible mejora o adaptación a futuros entornos en los que se pueda desarrollar. Asimismo es importante volver a destacar que con la sola modificación del fichero que contiene la arquitectura (explicado en 5.3), sería suficiente para un usuario, que desease simular una determinada aplicación sobre los diferentes algoritmos que han sido implementado para la composición inicial de servicios.

Capítulo 6

Evaluación

En este capítulo se van a mostrar los resultados obtenidos al evaluar los distintos algoritmos que hemos programado. Se van a realizar una serie de experimentos para comprobar su correcto desarrollo, de los cuales se van a obtener las características en el funcionamiento de cada uno de ellos.

6.1. Introducción

En esta parte de la memoria se recogen los resultados de una serie de pruebas que se han realizado sobre los distintos algoritmos desarrollados e implementados en el presente proyecto. Estas pruebas nos dan como resultado los tiempos de respuesta, tiempos de ejecución y número de evaluaciones necesarias para la ejecución de las diferentes aplicaciones.

6.1.1. Definición de las pruebas

En primer lugar cabe destacar que se van a mostrar las cuatro pruebas que hemos considerado más aclaratorias a la hora de explicar el funcionamiento de los algoritmos, así como de definir su punto de utilización óptimo. Cabe destacar asimismo que se ha utilizado como variable diferenciadora para cada una de las pruebas el número de combinaciones posibles entre los distintos perfiles de los diferentes servicios. Se ha decidido tener en cuenta esta variable debido a que su evaluación representa en los algoritmos el principal

aporte en el consumo de tiempo de ejecución. Para hallar el número de combinaciones de una aplicación hay que aplicar la siguiente ecuación:

$$num_{comb} = \prod_{i=1}^{N} P_i \tag{6.1}$$

donde i va recorriendo los N servicios/operaciones que tiene la aplicación, y P_i se refiere al número de perfiles que posee cada servicio. Como se puede apreciar en la ecuación 6.1 el número de combinaciones dependerá del número de servicios que cada aplicación aporte. En un principio parecería más razonable y económico (menor número de operaciones a realizar) tener en cuenta para las distintas pruebas el número de servicios que cada aplicación tiene, el cual es calculado al inicio de cada sistema. Con ello ahorraríamos la operación de la ecuación 6.1, la cual sería innecesaria, pero se llegó a la conclusión que la sola evaluación de los servicios no aporta información suficiente, así como la sola evaluación del número de combinaciones no podría ser concluyente, ya que a igual número de combinaciones, en tiempo es más pesado evaluar una combinación con mayor número de operaciones. Por lo tanto finalmente se ha decidido tomar una variable a la cual hemos denominado Ω que representará el producto del número de combinaciones por el número de servicios de la aplicación. Se ha realizado un estudio para verificar la adecuación de la nueva variable, comprobándose que para aplicaciones con distinto número de combinaciones y distinto número de nodos, pero con igual Ω, sus tiempos de ejecución son muy similares. Este estudio se ha realizado para un rango de Ωs representativo de las aplicaciones que se pueden llegar a usar en el sistema implementado, obteniéndose resultados que ratifican el uso de la variable para la selección de los tramos.

Una vez elegida la variable diferenciadora de las pruebas, fue necesario acotar esta para definir las franjas. Para ello se decidió dividir Ω en tres franjas, las cuales estarían estrictamente ligadas a los tiempos de ejecución que se obtienen en las realizaciones de los algoritmos. Se ha denominado Ω *baja* a aquellos sistemas en los cuales la variable Ω no supera las 1050 evaluaciones, para las cuales el algoritmo exhaustivo no ha de invertir más de 0,45 segundos, lo cual es ideal para sistemas que necesiten un tiempo acotado bajo en su ejecución. Posteriormente se definió como franja de Ω *media*, a aquella que va desde que la variable Ω es 1050, hasta que esta no supere las 15650, cuya evaluación ocupa al algoritmo exhaustivo desde 0,45 segundo hasta los 5,6 segundos aproximadamente, lo

cual acota el tiempo de respuesta para nuestro sistema en aplicaciones con mayor exigencia computacional. Finalmente se estableció como Ω *alta*, a aquellas aplicaciones cuya Ω supere 15650, ya que para nuestras aplicaciones el estar por encima de ese tiempo de ejecución hace que se deban tratar adecuadamente.

Por lo tanto, establecido ya la variable de control de las diferentes pruebas, se pueden explicar en que han consistido estas. Las cuatro pruebas realizadas han ejecutado los cuatro algoritmos (explicados en el apartado 4.2), para aplicaciones con Ω baja (3 nodos en serie y 64 combinaciones posibles y por lo tanto una Ω de 192), Ω media (5 nodos en serie y 1024 combinaciones posibles, es decir una Ω de 5120) y una Ω alta (7 nodos en serie y 16384 combinaciones posibles, con una Ω de 114688). La cuarta prueba consistió en realizar un estudio para una aplicación con servicios en paralelo y un número medio de combinaciones (3 nodos en serie y 2 en paralelo con 1024 combinaciones posibles y Ω igual a 5120). Para realizar la ejecución simultanea y bajo las mismas condiciones de los cuatro algoritmos se implementó una *function* en *Matlab* que reuniese los cuatro desarrollos. Esta *function* será ejecutada de forma recurrente hasta obtener cinco aplicaciones no planificables, lo cual significa haber obtenido una utilización máxima del sistema. Como lo que se pretende es que los cuatro algoritmos se ejecuten sobre las mismas condiciones del sistema, se irán ejecutando cada uno de ellos, pero una vez finalizada su ejecución y almacenados los datos de interés se volverá a restaurar el sistema como estaba en un principio para que este sea idéntico para el algoritmo posterior. Cuando se terminen de evaluar los cuatro algoritmos se planificará la combinación óptima obtenida en el algoritmo exhaustivo para así proceder a aumentar la utilidad del sistema y poder realizar otra nueva evaluación con una nueva aplicación. Este procedimiento se repite en al menos diez ocasiones para así poder realizar sobre los resultados valores medios y sus correspondientes varianzas. Se ha decidido realizar al menos 10 ejecuciones distintas para cada prueba ya que los tiempos de ejecución de cada una de ellas son muy costosos, sobre todo cuando la Ω va superando los valores medios. Sirva como ejemplo, que una sola ejecución de la tercera prueba correspondiente a una aplicación cuyo Ω sea alto, tarda alrededor de 55 minutos en su ejecución. A esto hay que añadirle el tiempo de pos procesado de los datos obtenidos, así como la necesidad de obtener 10 pruebas en las cuales el número de ejecuciones de cada algoritmo coincida, ya que al tratar con aplicaciones aleatorias, no siempre se asegura que la prueba realiza

el mismo número de ejecuciones de los algoritmos. Todo esto hace que sea imposible la realización de mayor número de ejecuciones en el ámbito de un proyecto fin de carrera. Por ello se consideró que con 10 ejecuciones realizadas es suficiente para obtener datos que nos permitan dar valores medios correctos. Además cabe destacar que las varianzas obtenidas son muy bajas, lo cual nos asegura la proximidad de los datos a la hora de las repeticiones.

Para establecer cuando ha de dejar de evaluarse una prueba se ha decidido que se finalizará la misma cuando se obtengan cinco aplicaciones que no sean planificables en el sistema de la prueba. Es decir, la prueba irá ejecutando la función creada en *Matlab* que contiene los cuatro algoritmos desarrollados, la ejecución de estos ira cargando la utilidad del sistema, llegando un punto en el cual el sistema estará a punto de saturarse y no permitirá la ejecución de ninguna otra aplicación, rechazándolas por no planificables, debido a que no es posible encontrar una combinación planificable en el sistema o porque los tiempos de respuesta superan al máximo *deadline* de la aplicación. Sin embargo no se han tenido en cuenta para estas cinco aplicaciones, aquellas en las cuales la no planificabilidad viene debida al aumento del tiempo de respuesta de mensajes ya introducidos en el sistema, es decir aquellas aplicaciones que no son planificables en la red. En el momento que se obtengan cinco aplicaciones no planificables, se considerará que el sistema está saturado y no se podrán ejecutar nuevas aplicaciones.

Los deadlines que se han establecido para las pruebas han sido aleatorios, y se han definido de la siguiente manera:

```
deadline_nodo = round(20*(rand+1));
```

esto nos llevará a tener *deadlines* de nodo entre 20 y 40 unidades de ejecución.

El *deadeline* de la red estará definido como:

```
deadline_red = round(50*(rand)+nodosSerie*40);
```

El *deadline* de aplicación vendrá definido por la siguiente fórmula, y es función del número de nodos en serie:

```
deadline_aplicacion = round(200*(rand)+nodosSerie*40);
```

Finalmente hay que definir que para todas las pruebas realizadas se han utilizado tres nodos físicos (procesadores) para incluir las operaciones y tres redes. A la hora de definir

los 'Servicios' se decidió que estos fuesen cuatro, que estarán representados por los letras mayúsculas 'A', 'B', 'C' y 'D', y tres redes que en este casi sí que representarán a las tres redes físicas. Estas redes serán identificadas con las letras minúsculas 'z', 'y', y 'w'.

A la hora de definir las cuatro operaciones se ha decidido que cada una de ellas este representada por cuatro perfiles distintos. Cada uno de los perfiles estará caracterizado por el tiempo de computación de la tarea correspondiente ('C_i') y por el nodo físico sobre el que se ejecuta. Para hacernos una idea de cómo serían los experimentos, reproducimos a continuación un ejemplo aleatorio de los perfiles obtenidos para cada servicio. Estos perfiles son creados aleatoriamente para cada aplicación y para cada sistema, favoreciendo la obtención de resultados más generalistas. Para su creación se utiliza la siguiente expresión:

```
[t_comp_mens nodo_fisico] = [ceil(5*(rand)) ceil(3*(rand))]
```

Un ejemplo de esta creación sería el siguiente:

A		**B**		**C**		**D**	
C_i	*Nodo físico*	C_i	*Nodo físico*	C_i	*Nodo físico*	C_i	*Nodo físico*
1	2	1	3	2	2	1	1
4	3	3	2	3	1	2	2
2	1	5	1	3	3	4	3
1	1	2	3	3	2	3	2

Tabla 6.1: Ejemplo aleatorio de un conjunto de perfiles utilizados por los distintos servicios

En el caso de las redes, como se introdujo antes, cada una se referirá a una red física, por lo que no existirán diferentes perfiles para cada una de ellas. La tabla de correspondencias entre la representación y la red física es:

Representación	**Red física**
z	1
y	2
w	3

Tabla 6.2: Correspondencia entre las representaciones y las redes físicas

Respecto a los mensajes, los tiempos de computación máximos también serán creados para cada aplicación de forma aleatoria bajo la siguiente ecuación:

```
t_comp_mens = round(4*(rand+0.25))
```

6.1.2. Medidas de cada prueba

Dentro de cada una de las pruebas se han realizado cuatro medidas. Estas han consistido en:

Utilidades y tiempos de respuesta: Se mostrará cómo se van cargando los 3 nodos físicos a lo largo de la realización de la prueba. Se pretende apreciar como el propio diseño del sistema desarrollado automáticamente va compensado la carga de las nuevas tareas entre los tres nodos, favoreciendo una utilización óptima de los recursos del sistema en conjunto. Asimismo se mostrará en esta primera parte el tiempo de respuesta del sistema. Todos los tiempos de respuesta que se muestren serán de aplicaciones planificables y serían totalmente predecibles conociendo las características de la aplicación (*deadlines*, tareas, etc.). El tiempo de respuesta es la variable que se ha tenido en cuenta a la hora del diseño e implementación de los distintos algoritmos. Como anteriormente a la realización de estas pruebas, ya se probó que los diseños funcionasen correctamente, en estas pruebas esta variable no tendrá tanta importancia ya que lo que se pretende es comparar los algoritmos para las distintas aplicaciones que se pueden desarrollar.

Tiempos de ejecución: Esta parte de la prueba consistirá en evaluar los tiempos de ejecución de cada uno de los algoritmos para las distintas utilidades a las que se enfrenta el sistema.

Número de evaluaciones: En esta tercera parte se pretende estudiar el número de combinaciones que cada algoritmo evalúa antes de devolver al sistema la combinación con menor tiempo de respuesta. Esta prueba es un complemento indispensable para entender los tiempos de ejecución obtenidos en el cálculo anterior, ya que como se presupone, tiempos de ejecución y combinaciones evaluadas, están estrechamente relacionadas entre sí. En esta prueba se pretende comprobar cómo el algoritmo exhaustivo siempre comprobará

todas las posibles evaluaciones. El algoritmo heurístico solo comprobará las combinaciones procedentes de los bloques seleccionados hasta un máximo de un cuarto de las combinaciones posibles, $num_{comb\_heu} = (tam_{bloque})^N$, donde N es el número de servicios de la aplicación. El mejorado comprobará todas las evaluaciones a no ser que contengan algún patrón de no planificabilidad y el heurístico mejorado comprobará las correspondientes a los bloques, dejando de comprobar aquellas que tengan algún patrón de no planificabilidad y hasta un máximo de un cuarto de las totales.

Error en la selección de la combinación optima: Como última medida se calcula el error que produce en la selección de la mejor combinación, la utilización de los algoritmos heurísticos frente a los exhaustivos. Como ya se explicó en el apartado 4.2.3, un algoritmo heurístico garantiza mejores tiempos de respuesta, dejando de garantizar la obtención de la solución óptima y sin determinar a qué distancia de la misma nos encontramos. Por lo tanto lo que se pretende con esta medida es caracterizar la proximidad de estos algoritmos a la solución óptima para las diferentes pruebas. Para ello, a la hora de la evaluación del algoritmo Exhaustivo, se procede a ordenar los distintos tiempos de respuesta de las diferentes combinaciones en orden de menor a mayor, calculando posteriormente la distancia entre las combinaciones obtenidas por los algoritmos heurísticos y la óptima devuelta por el exhaustivo.

6.2. Pruebas realizadas

6.2.1. Aplicaciones con Ω baja

En la figura 6.1 podemos encontrar la aplicación que vamos a testear. Como se puede apreciar es una aplicación con un número pequeño de servicios, todos ellos en serie. Tiene tres servicios que efectuarán en el siguiente orden las operaciones ‘A’, ‘B’, ‘C’ y existirán dos redes ‘z’ que unirá los primeros servicios e ‘y’ que unirá el servicio central con el último.

De todas las ejecuciones realizadas se han usado para realizar el estudio aquellas en las que el sistema ha admitido 15 aplicaciones antes de saturarse. Se han recopilado los datos de 10 ejecuciones de 15 aplicaciones de 3 nodos cada una. Posteriormente se han procesado y a continuación mostramos los resultados. Estos resultados vienen dados como media

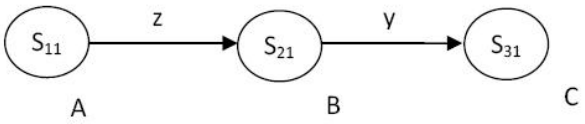

Figura 6.1: Aplicación sencilla, con bajo número de servicios

de las 10 ejecuciones, con su respectiva varianza (σ), calculadas a partir de la siguiente expresión:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(X_i - u)^2}{n}} \tag{6.2}$$

En primer lugar, mostraremos la utilidad de los tres nodos físicos junto con el tiempo de respuesta medio, en unidades de tiempo, la cual queda reflejada en la tabla 6.3.

Como se puede apreciar en la tabla 6.3, los nodos físicos se van rellenando de tareas poco a poco, aumentando su carga sin llegar en ningún momento a 1, el cual sería el máximo. Asimismo como se comentó en los capítulos previos, el uso del algoritmo exhaustivo a la hora de introducir las tareas hace que se realice una repartición de carga entre los tres nodos, facilitando así ver el comportamiento de los distintos algoritmos para nodos cargados equitativamente.

Podemos apreciar en los tiempos de respuesta en unidades de tiempo, que pueden existir fluctuaciones, ya que hay tareas posteriores que ofrecen menor tiempo de respuesta que tareas precedentes. Esto es debido al uso de tiempos de ejecución de los mensajes en la red, así como de los diferentes *deadlines*, aleatorios, lo cual favorece que no siempre los tiempos de respuesta hayan de ir aumentando.

Para los posteriores estudios, vamos a definir una nueva variable que hemos llamado *Utilidad media*. Como su propio nombre indica es la media de las utilidades de los tres nodos físicos utilizados. Se utiliza esta variable para a la hora de las representaciones posteriores se puedan realizar conclusiones de cómo están actuando los distintos algoritmos dependiendo la carga del sistema. Esta variable será usada para la obtención de la tabla 6.4 que mostrará los tiempos de ejecución de los cuatro algoritmos implementados y cuya representación

nº ejec.	**Utilidad**						**t resp**	
	n.fis 1		**n.fis 2**		**n.fis 3**			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0000	0,0000	0,0570	0,0046	0,1425	0,0114	42,00	0,00
2	0,0886	0,0093	0,0868	0,0036	0,1724	0,0098	44,84	4,00
3	0,1706	0,0195	0,1548	0,0196	0,2103	0,0105	50,64	7,86
4	0,2487	0,0289	0,2343	0,0141	0,2498	0,0080	54,00	9,56
5	0,2797	0,0279	0,2957	0,0171	0,2805	0,0076	64,37	7,65
6	0,3025	0,0193	0,3669	0,0236	0,3117	0,0075	86,81	19,39
7	0,4002	0,0220	0,3995	0,0239	0,3443	0,0080	102,45	10,95
8	0,4529	0,0372	0,4717	0,0211	0,4116	0,0701	95,04	13,27
9	0,5207	0,0230	0,5251	0,0209	0,4417	0,0719	116,59	17,20
10	0,5531	0,0251	0,5896	0,0311	0,4741	0,0739	106,85	15,10
11	0,5588	0,0207	0,6275	0,0269	0,5903	0,0324	127,92	21,82
12	0,6640	0,0280	0,6627	0,0253	0,6253	0,0358	143,96	15,29
13	0,7297	0,0299	0,7283	0,0290	0,6580	0,0398	141,45	27,86
14	0,7990	0,0385	0,7864	0,0241	0,6899	0,0394	152,57	25,48
15	0,8184	0,0318	0,8356	0,0129	0,7509	0,0619	176,80	27,14

Tabla 6.3: Utilidades y tiempo de respuesta para una Ω baja

nº ejec.	Utilidad media		tiempo ejecución (seg)							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0667	0,0592	0,078	0,00	0,019	0,01	0,056	0,03	0,019	0,01
2	0,1099	0,0407	0,075	0,01	0,019	0,01	0,059	0,03	0,015	0,01
3	0,1771	0,0286	0,087	0,01	0,024	0,01	0,060	0,03	0,021	0,01
4	0,2442	0,0205	0,087	0,01	0,019	0,01	0,056	0,03	0,019	0,01
5	0,2852	0,0207	0,094	0,00	0,022	0,01	0,068	0,03	0,021	0,01
6	0,3259	0,0338	0,112	0,03	0,026	0,04	0,083	0,05	0,027	0,04
7	0,3804	0,0328	0,106	0,01	0,027	0,02	0,074	0,04	0,027	0,02
8	0,4447	0,0528	0,109	0,00	0,027	0,01	0,085	0,03	0,027	0,01
9	0,4943	0,0579	0,120	0,02	0,032	0,03	0,099	0,05	0,030	0,03
10	0,5367	0,0671	0,112	0,01	0,022	0,01	0,078	0,04	0,023	0,01
11	0,5916	0,0391	0,116	0,01	0,021	0,01	0,075	0,04	0,020	0,01
12	0,6504	0,0349	0,115	0,01	0,018	0,01	0,065	0,03	0,021	0,01
13	0,7045	0,0470	0,121	0,02	0,028	0,01	0,065	0,04	0,027	0,01
14	0,7568	0,0597	0,122	0,01	0,027	0,01	0,053	0,03	0,027	0,01
15	0,8008	0,0542	0,111	0,02	0,024	0,02	0,039	0,01	0,022	0,01

Tabla 6.4: Utilidad media y tiempos de ejecución para una Ω baja

gráfica se muestra en la figura 6.2. En ella podemos apreciar que para situaciones de pocas evaluaciones posibles, cualquiera de los cuatro algoritmos tendrá tiempos de ejecución buenos independientemente de la utilidad del sistema.

Otra característica que hemos evaluado es el número de combinaciones evaluadas por los cuatro algoritmos, cuyos resultados se muestran en la tabla 6.5 y se representan en la figura 6.3. Podemos observar como los algoritmos tienen un funcionamiento como se esperaba de su implementación, explicado en el apartado 6.1.2. Cabe destacar el posible uso de los algoritmos mejorados para situaciones de sistemas a punto de saturarse, ya que se comprueba su utilidad a la hora de reducir el número de combinaciones que evalúan.

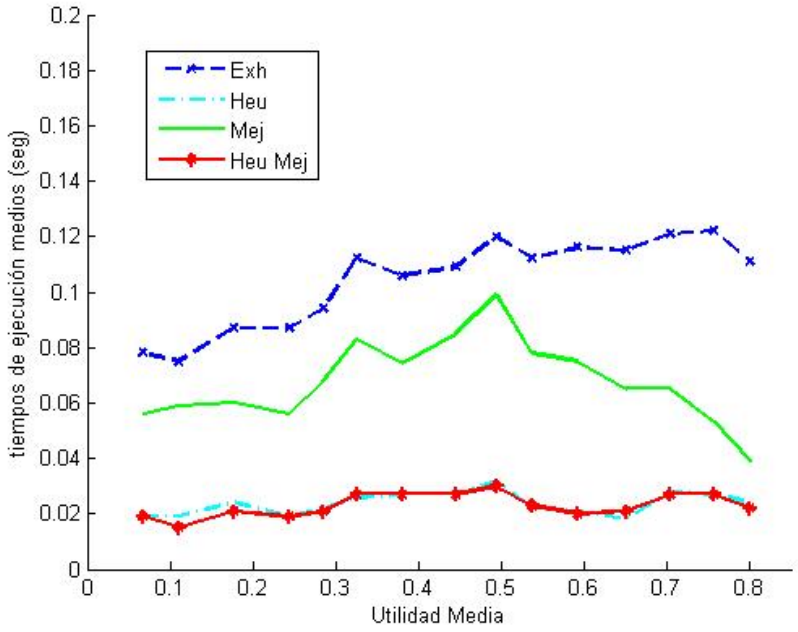

Figura 6.2: Tiempos de ejecución frente a la utilidad para una Ω baja

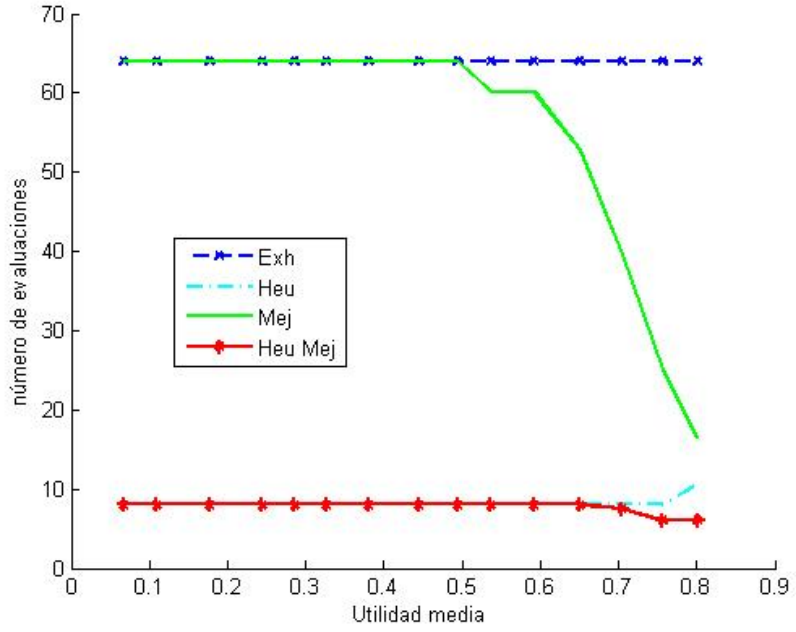

Figura 6.3: Número de evaluaciones frente a la utilidad para una Ω baja

nº ejec.	Utilidad media		Número de comb. evaluadas							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0667	0,0592	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
2	0,1099	0,0407	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
3	0,1771	0,0286	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
4	0,2442	0,0205	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
5	0,2852	0,0207	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
6	0,3259	0,0338	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
7	0,3804	0,0328	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
8	0,4447	0,0528	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
9	0,4943	0,0579	64,00	0,00	8,00	0,00	64,00	0,00	8,00	0,00
10	0,5367	0,0671	64,00	0,00	8,00	0,00	60,17	6,80	8,00	0,00
11	0,5916	0,0391	64,00	0,00	8,00	0,00	60,17	6,80	8,00	0,00
12	0,6504	0,0349	64,00	0,00	8,00	0,00	52,80	10,60	8,00	0,00
13	0,7045	0,0470	64,00	0,00	8,00	0,00	39,58	12,06	7,55	0,80
14	0,7568	0,0597	64,00	0,00	8,00	0,00	24,99	10,15	6,03	1,47
15	0,8008	0,0542	64,00	0,00	10,56	3,92	16,52	1,62	6,03	1,47

Tabla 6.5: Utilidad media y número de combinaciones evaluadas para una Ω baja

Por último se ha evaluado lo óptimo de la combinación seleccionada por los algoritmos heurísticos. Para ello, a la hora de la evaluación del algoritmo Exhaustivo se ha procedido a ordenar los distintos tiempos de respuesta de las diferentes combinaciones en orden de menor a mayor. Se ha considerado para esta evaluación, la distancia en combinaciones que separa la combinación seleccionada por los dos algoritmos heurísticos, con respecto a la óptima del algoritmo Exhaustivo. Los resultados obtenidos se muestran en la tabla 6.6y en la figura 6.4.

nº ejec.	**Utilidad media**		**Distancia a la óptima**			
			Alg Heu		**Alg Heu Mej**	
	media	*var*	*media*	*var*	*media*	*var*
1	0,0667	0,0592	25,88	8,63	25,88	8,63
2	0,1099	0,0407	25,19	9,52	25,19	9,52
3	0,1771	0,0286	10,53	16,27	10,53	16,27
4	0,2442	0,0205	9,71	10,73	9,71	10,73
5	0,2852	0,0207	3,00	0,00	3,00	0,00
6	0,3259	0,0338	22,43	6,01	22,43	6,01
7	0,3804	0,0328	26,58	7,63	26,58	7,63
8	0,4447	0,0528	15,62	17,33	15,62	17,33
9	0,4943	0,0579	3,29	0,80	3,29	0,80
10	0,5367	0,0671	2,35	0,49	2,35	0,49
11	0,5916	0,0391	5,93	2,33	5,93	2,33
12	0,6504	0,0349	6,60	1,47	6,60	1,47
13	0,7045	0,0470	3,73	1,41	3,73	1,41
14	0,7568	0,0597	4,79	1,41	4,79	1,41
15	0,8008	0,0542	3,78	2,33	3,78	2,33

Tabla 6.6: Utilidad media y número de combinaciones entre la óptima y la seleccionada por los algoritmos heurísticos para una Ω baja

Como podemos observar solo encontrará combinaciones cercanas a la óptima cuando la utilidad sea mayor que 0.5, lo cual nos lleva a concluir, junto con todo lo anterior, que

para una Ω baja, los mejores algoritmos serán los que nos proporcionan una combinación óptima (exhaustivo y mejorado). También habría que destacar que aunque los algoritmos heurísticos se alejen de obtener la combinación óptima, su selección se encuentra muy cercana en tiempo de respuesta a la más favorable (óptima).

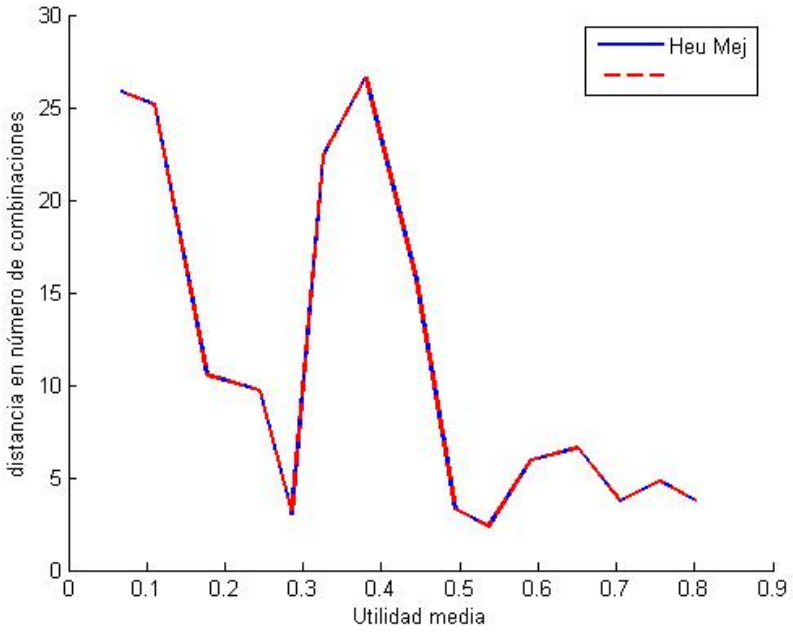

Figura 6.4: Número de combinaciones entre la óptima y la obtenida para una Ω baja

6.2.2. Aplicaciones con Ω media

En la figura 6.5 podemos encontrar la aplicación que vamos a evaluar como aplicación con un número de servicios medio. Como se puede apreciar es una aplicación con todos los servicios en serie. Tiene cinco servicios que efectuarán en el siguiente orden las operaciones 'A', 'B', 'C','D','C' y existirán tres redes, 'z' que unirá los primeros servicios, 'y' que unirá los servicios segundo y tercero, así como el cuarto con el último y 'y' que unirá tercero y cuarto.

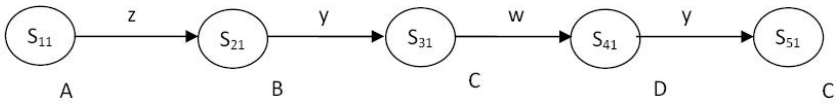

Figura 6.5: Aplicación sencilla, con un número de servicios medio

nº ejec.	Utilidad						t resp	
	n.fis 1		n.fis 2		n.fis 3			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0000	0,0052	0,0622	0,0026	0,0622	0,0026	83,91	4,00
2	0,2569	0,0248	0,1285	0,0124	0,1285	0,0124	93,19	12,65
3	0,2664	0,0376	0,3079	0,0543	0,1973	0,0077	96,49	12,24
4	0,3732	0,0506	0,3876	0,0781	0,2809	0,0430	132,81	24,35
5	0,4963	0,0796	0,4597	0,0642	0,3628	0,0333	162,33	28,03
6	0,5506	0,0817	0,5704	0,0537	0,4993	0,0302	211,89	43,37
7	0,6415	0,1361	0,6480	0,0469	0,6027	0,0388	201,69	24,59
8	0,7170	0,1117	0,7222	0,0440	0,7209	0,0601	232,50	56,76
9	0,8090	0,0785	0,8088	0,0177	0,7915	0,0599	261,22	46,58

Tabla 6.7: Utilidades y tiempo de respuesta para una Ω media

De todas las ejecuciones realizadas se han usado para realizar el estudio aquellas en las que el sistema ha admitido 9 aplicaciones antes de saturarse. Se han recopilado los datos de 10 ejecuciones de 9 aplicaciones de 5 nodos cada una. Al igual que en el caso anterior, los resultados vienen dados como media de las 10 ejecuciones, con su respectiva varianza.

En primer lugar, mostraremos la utilidad de los tres nodos físicos junto con el tiempo de respuesta medio, en unidades de tiempo en la tabla 6.7.

Posteriormente, volviendo a hacer uso de la variable *Utilidad media*, se evalúan los tiempos de ejecución para los cuatro algoritmos implementados en la tabla 6.8, cuya representación se encuentra en la figura 6.6. Observamos como el tiempo de ejecución ha

aumentado con respecto a las pruebas anteriores. También se puede observar lo efectivo que resulta el algoritmo mejorado para sistemas muy cargados.

nº ejec.	**Utilidad media**		**tiempo ejecución (seg)**							
			Alg Exh		**Alg Heu**		**Alg Mej**		**Alg Heu Mej**	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0830	0,0296	1,888	0,01	0,072	0,01	1,872	0,01	0,068	0,01
2	0,1619	0,0633	2,132	0,02	0,079	0,02	2,125	0,01	0,080	0,02
3	0,2530	0,0611	2,328	0,02	0,093	0,01	2,306	0,02	0,090	0,01
4	0,3438	0,0765	2,506	0,02	0,112	0,01	2,490	0,02	0,103	0,01
5	0,4358	0,0850	2,694	0,01	0,100	0,01	2,681	0,02	0,097	0,01
6	0,5392	0,0668	2,866	0,02	0,106	0,01	2,828	0,07	0,102	0,02
7	0,6304	0,0891	3,014	0,09	0,112	0,01	2,510	0,45	0,098	0,02
8	0,7201	0,0775	2,618	0,59	0,120	0,02	1,571	0,70	0,082	0,04
9	0,8030	0,0585	2,104	0,51	0,118	0,03	0,521	0,29	0,071	0,01

Tabla 6.8: Utilidad media y tiempos de ejecución para una Ω media

Para esta nueva aplicación se ha de estudiar el efecto que ejercerá el aumento de carga en los nodos físicos para la evolución del número medio de evaluaciones necesarias para encontrar una combinación planificable. Esto se muestra en la tabla 6.9 y en la representación 6.7. Podemos volver a observar que la disminución de tiempo destacada para el algoritmo mejorado en sistemas cargados es consecuencia directa de una disminución elevada de combinaciones que evalúa.

Finalmente mostramos los resultados de evaluar las combinaciones obtenidas para los algoritmos heurísticos frente al óptimo obtenido por el algoritmo exhaustivo. Esto se muestra en la tabla 6.10 y en la figura 6.8. Durante esta prueba hemos observado el hecho que cuando esta poco cargado el sistema, los algoritmos heurísticos obtienen la combinación óptima. Esto es consecuencia directa de la buena selección a la hora de la elección de la figura de mérito parcial (C_i).

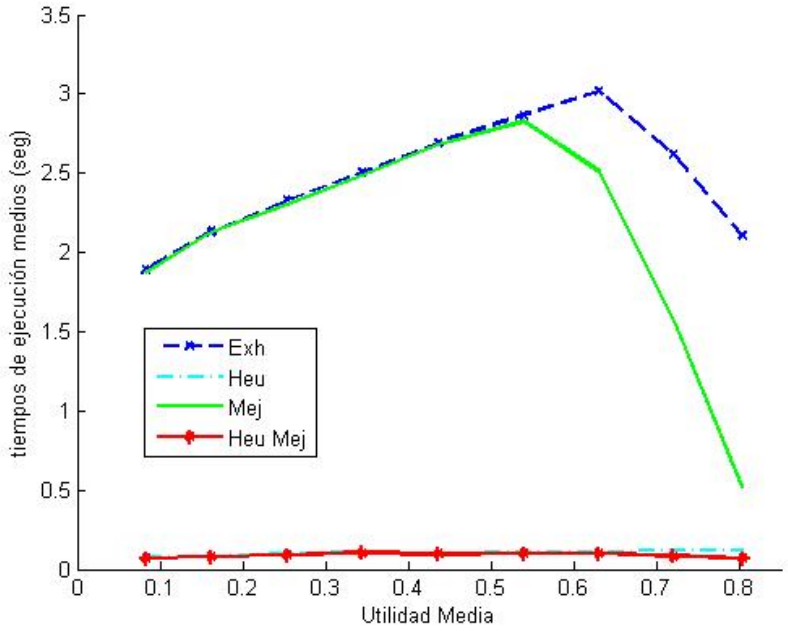

Figura 6.6: Tiempos de ejecución frente a la utilidad

nº ejec.	**Utilidad media**		**Número de comb. evaluadas**							
			Alg Exh		**Alg Heu**		**Alg Mej**		**Alg Heu Mej**	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0830	0,0296	1024,00	0,00	32,00	0,00	1024,00	0,00	32,00	0,00
2	0,1619	0,0633	1024,00	0,00	32,00	0,00	1024,00	0,00	32,00	0,00
3	0,2530	0,0611	1024,00	0,00	32,00	0,00	1024,00	0,00	32,00	0,00
4	0,3438	0,0765	1024,00	0,00	32,00	0,00	1024,00	0,00	32,00	0,00
5	0,4358	0,0850	1024,00	0,00	32,00	0,00	1022,40	2,33	32,00	0,00
6	0,5392	0,0668	1024,00	0,00	32,00	0,00	964,79	43,92	30,33	2,06
7	0,6304	0,0891	1024,00	0,00	32,00	0,00	731,02	182,43	26,36	5,59
8	0,7201	0,0775	1024,00	0,00	32,00	0,00	372,27	215,81	18,73	8,70
9	0,8030	0,0585	1024,00	0,00	36,76	12,80	89,82	66,17	17,20	2,80

Tabla 6.9: Utilidad media y número de combinaciones evaluadas para una Ω media

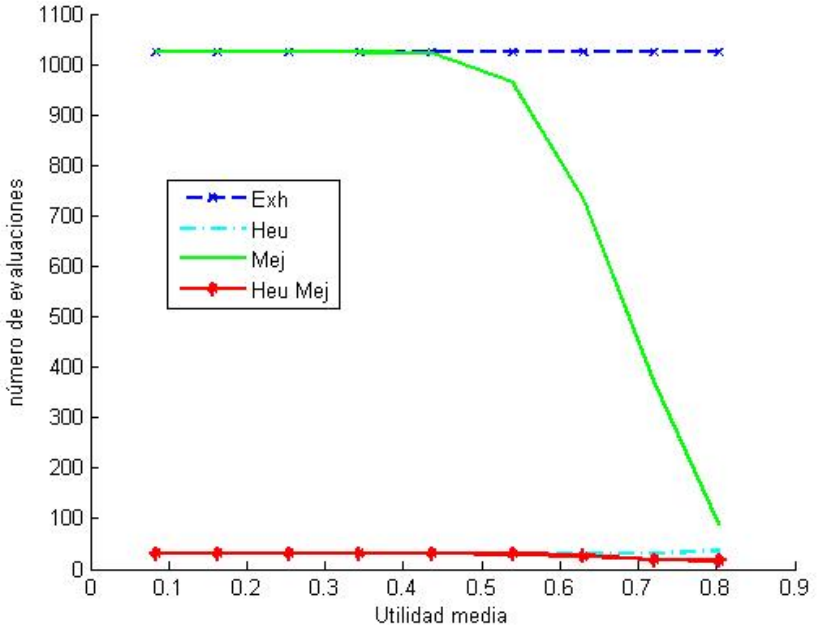

Figura 6.7: Número de evaluaciones frente a la utilidad para una Ω media

nº ejec.	**Utilidad media**		**Distancia a la óptima**			
			Alg Heu		**Alg Heu Mej**	
	media	*var*	*media*	*var*	*media*	*var*
1	0,0830	0,0296	0,00	0,00	0,00	0,00
2	0,1619	0,0633	0,00	0,00	0,00	0,00
3	0,2530	0,0611	0,00	0,00	0,00	0,00
4	0,3438	0,0765	0,00	0,00	0,00	0,00
5	0,4358	0,0850	10,80	2,71	10,80	2,71
6	0,5392	0,0668	11,60	2,24	11,60	2,24
7	0,6304	0,0891	1,20	0,75	1,20	0,75
8	0,7201	0,0775	2,40	3,20	2,40	3,20
9	0,8030	0,0585	0,40	0,49	2,00	0,63

Tabla 6.10: Utilidad media y número de combinaciones entre la óptima y la seleccionada por los algoritmos heurísticos, para una Ω media

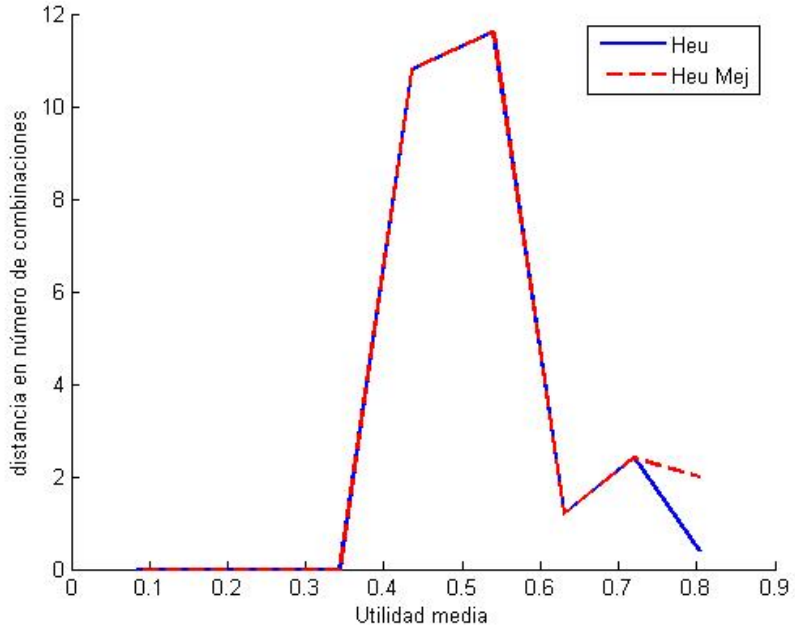

Figura 6.8: Número de combinaciones entre la óptima y la obtenida para una Ω media

6.2.3. Aplicaciones con Ω alta

En la figura 6.9 podemos encontrar la aplicación que vamos a testear como aplicación con un número de servicios medio. Como se puede apreciar es una aplicación con todos los servicios en serie. Tiene siete servicios que efectuarán en el siguiente orden las operaciones 'A', 'B', 'C', 'D', 'B', 'A', 'C' y existirán tres redes que unirán los servicios anteriores en el siguiente orden: 'z', 'y', 'w', 'z', 'y', 'w'.

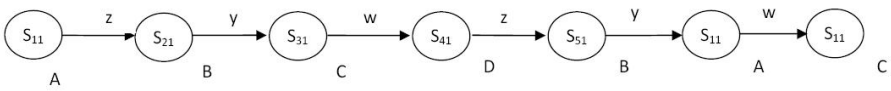

Figura 6.9: Aplicación sencilla, con un número de servicios medio

De todas las ejecuciones realizadas se han usado para realizar el estudio aquellas en las que el sistema ha admitido 7 aplicaciones antes de saturarse. Se han recopilado los datos de 10 ejecuciones de 7 aplicaciones de 5 nodos cada una. Al igual que en el caso anterior, los resultados vienen dados como media de las 10 ejecuciones, con su respectiva varianza.

En primer lugar, mostraremos la utilidad de los tres nodos físicos junto con los tiempos de respuesta en unidades de tiempo de cada aplicación en la tabla 6.11.

nº ejec.	Utilidad						t resp	
	n.fis 1		n.fis 2		n.fis 3			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1259	0,0148	0,0630	0,0074	0,1259	0,0148	126,90	5,00
2	0,2835	0,0185	0,1342	0,0103	0,2685	0,0205	140,75	14,79
3	0,3618	0,0579	0,3314	0,0398	0,3342	0,0303	197,78	43,23
4	0,4816	0,0475	0,4930	0,0744	0,4046	0,0253	276,76	31,42
5	0,5628	0,0251	0,5537	0,0781	0,5253	0,0416	287,68	70,00
6	0,7018	0,0646	0,6503	0,0578	0,7071	0,0417	330,76	57,73
7	0,8319	0,0613	0,7595	0,0428	0,7714	0,0390	323,74	55,19

Tabla 6.11: Utilidades y tiempo de respuesta para una Ω alta

Posteriormente, volviendo a hacer uso de la variable *Utilidad media*, se evalúan los tiempos de ejecución para los cuatro algoritmos implementados en la tabla 6.12, cuya representación se encuentra en la figura 6.10.

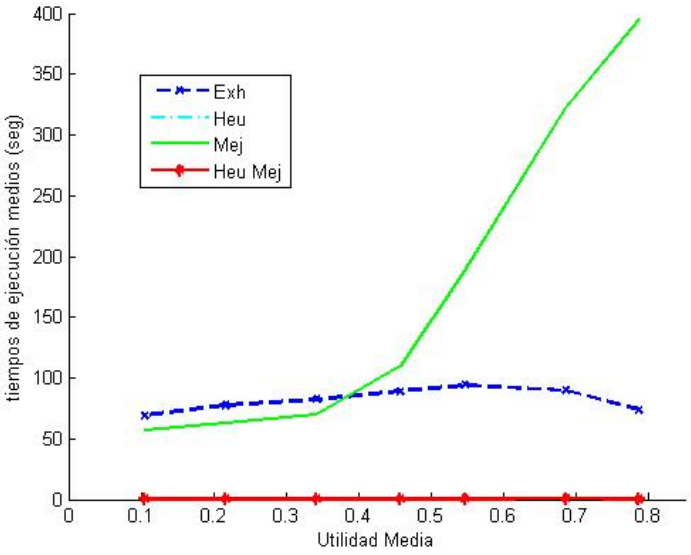

Figura 6.10: Tiempos de ejecución frente a la utilidad para una Ω alta

nº ej.	Util. media		tiempo ejecución (seg)							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1057	0,0325	69,078	2,30	0,529	0,13	56,925	3,00	0,432	0,07
2	0,2170	0,0694	77,572	4,27	0,465	0,03	62,577	2,24	0,414	0,02
3	0,3422	0,0467	82,410	4,00	0,450	0,01	69,718	3,86	0,437	0,01
4	0,4580	0,0669	88,943	5,02	0,486	0,01	109,787	110,37	0,500	0,02
5	0,5471	0,0556	94,114	4,18	0,549	0,02	187,942	78,95	0,538	0,03
6	0,6859	0,0611	89,494	5,63	0,748	0,42	321,700	74,90	0,689	0,11
7	0,7870	0,0584	73,936	7,91	0,618	0,06	395,266	116,39	0,344	0,11

Tabla 6.12: Utilidad media y tiempos de ejecución

Se puede observar cómo se dispara el tiempo de ejecución en el caso del algoritmo mejorado. Esto nos lleva a la conclusión que este algoritmo no será compatible con Omegas elevadas, por lo tanto se ha de no usar en estos casos.

Otra de las características evaluadas es el número de combinaciones evaluadas por los cuatro algoritmos, cuyos resultados se muestran en la tabla 6.13 y se representan en la figura 6.11. Podemos observar que el algoritmo mejorado sigue funcionando correctamente disminuyendo el número de combinaciones a evaluar pero sin embargo aumenta su tiempo de ejecución. Esto nos muestra la incompatibilidad a la que nos hemos referido anteriormente.

Finalmente mostramos los resultados de evaluar las combinaciones obtenidas para los algoritmos heurísticos frente al óptimo obtenido por el algoritmo exhaustivo. Esto se muestra en la tabla 6.14 y en la figura 6.12. Durante esta prueba hemos observado el hecho que cuando este poco cargado el sistema, los algoritmos heurísticos obtienen la combinación óptima. Esto es consecuencia directa de la buena selección a la hora de la elección de la figura de mérito parcial (C_i).

nº ej.	Util. media		Número de comb. evaluadas							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1057	0,0325	16384,00	0,00	128,00	0,00	16384,00	0,00	128,00	0,00
2	0,2170	0,0694	16384,00	0,00	128,00	0,00	16384,00	0,00	128,00	0,00
3	0,3422	0,0467	16384,00	0,00	128,00	0,00	16382,75	2,17	128,00	0,00
4	0,4580	0,0669	16384,00	0,00	128,00	0,00	15715,25	1084,70	126,99	1,73
5	0,5471	0,0556	16384,00	0,00	128,00	0,00	15187,35	1272,25	128,00	0,00
6	0,6859	0,0611	16384,00	0,00	128,00	0,00	5558,49	3508,30	76,68	25,07
7	0,7870	0,0584	16384,00	0,00	152,22	55,43	1378,70	978,41	41,40	8,67

Tabla 6.13: Utilidad media y número de combinaciones evaluadas para una Ω alta

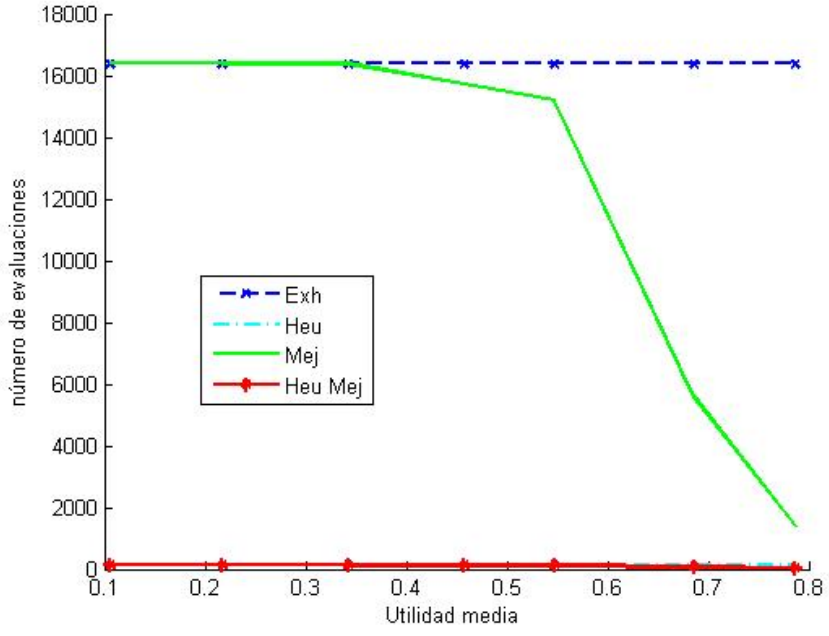

Figura 6.11: Número de evaluaciones frente a la utilidad para una Ω alta

nº ejec.	Utilidad media		Distancia a la óptima			
			Alg Heu		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*
1	0,1057	0,0325	0,00	0,00	0,00	0,00
2	0,2170	0,0694	0,00	0,00	0,00	0,00
3	0,3422	0,0467	0,00	0,00	0,00	0,00
4	0,4580	0,0669	71,60	6,31	71,60	6,31
5	0,5471	0,0556	18,60	8,26	18,60	8,26
6	0,6859	0,0611	2,40	1,02	3,20	1,17
7	0,7870	0,0584	1,20	0,98	1,80	0,75

Tabla 6.14: Utilidad media y número de combinaciones entre la óptima y la seleccionada por los algoritmos heurísticos para una Ω alta

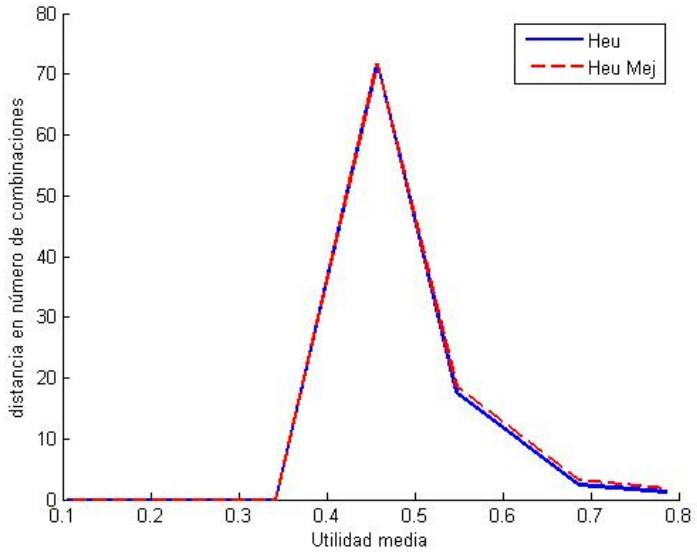

Figura 6.12: Número de combinaciones entre la óptima y la obtenida para una Ω alta

6.2.4. Aplicaciones con servicios en paralelo y Ω media

En la figura 6.13 podemos encontrar la aplicación que vamos a testear como aplicación con servicios en paralelo. Como se puede apreciar es una aplicación con tres servicios en

serie y otros dos en paralelo entre el primer y el tercer nodo. Tiene cinco servicios que efectuarán las operaciones 'A', 'B', 'C', 'D' y existirán tres redes que unirán los servicios: 'z', 'y', 'w'.

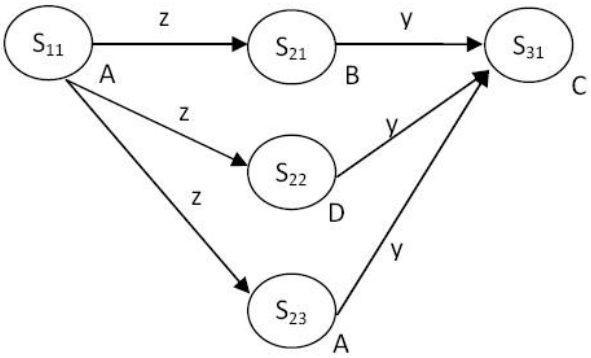

Figura 6.13: Aplicación sencilla, con un número de servicios medio

En primer lugar hemos calculado la utilidad de los tres nodos físicos junto con los tiempos de respuesta en unidades de tiempo de cada aplicación. Aparece reflejado en la tabla 6.15. Cabe destacar que para esta prueba se han se han seleccionado para su estudio aquellos sistemas que han admitido 7 aplicaciones antes de saturarse. Al igual que en los casos anteriores, los resultados vienen dados como media de las 10 ejecuciones, con su respectiva varianza.

Los resultados aquí obtenidos se podrían comparar con el caso de aplicaciones con un número medio de combinaciones (apartado 6.2.2), ya que ambos tratan con 5 nodos. Podemos observar que los tiempos de respuesta que obtenemos son más similares a los obtenidos en la tabla 6.3 correspondientes a un número bajo de combinaciones que a los obtenidos en la tabla 6.7. Esto es debido a que como se explicó antes nosotros estamos trabajando con solo 3 operaciones en serie, y las otras dos forman parte del paralelo, lo cual implica que al trabajar en paralelo el tiempo de respuesta del mismo es el máximo de todos los elementos que lo forman.

nº ejec.	Utilidad						t resp	
	n.fis 1		n.fis 2		n.fis 3			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0404	0,0070	0,0807	0,0141	0,3632	0,0634	42,00	0,00
2	0,1546	0,0152	0,1995	0,1012	0,4753	0,1398	44,00	0,00
3	0,2491	0,0289	0,3420	0,0780	0,5603	0,1786	59,28	16,59
4	0,3530	0,0787	0,4189	0,1005	0,6464	0,2079	74,71	15,25
5	0,5462	0,0768	0,6454	0,1062	0,6658	0,1901	113,29	18,99
6	0,6665	0,0997	0,7079	0,0980	0,7307	0,1557	107,52	12,09
7	0,7723	0,1130	0,7802	0,0973	0,8459	0,0494	144,38	20,83

Tabla 6.15: Utilidades y tiempo de respuesta para aplicación con servicios en paralelo

Otra de las medidas importantes a la hora de evaluar son los tiempos de respuesta para la aplicación sometida a estudio. Estos tiempos de respuesta aparecen en la tabla 6.16 y la representación de los tiempos de respuesta se realiza en la gráfica 6.14. En este caso podemos observar que aunque los tiempos de respuesta del exhaustivo sí que se parecen a los obtenidos en la tabla 6.8 de número medio de combinaciones, el resto ha disminuido.

nº ejec.	Utilidad media		tiempo ejecución (seg)							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1642	0,1508	2,101	0,27	0,115	0,17	1,259	0,70	0,111	0,12
2	0,2447	0,1804	2,311	0,08	0,085	0,01	1,064	0,94	0,093	0,01
3	0,3627	0,1843	2,577	0,08	0,097	0,01	1,136	1,03	0,097	0,01
4	0,4572	0,1993	2,845	0,13	0,102	0,01	1,111	1,10	0,098	0,01
5	0,6168	0,1469	1,883	0,84	0,147	0,04	0,430	0,52	0,127	0,04
6	0,7012	0,1248	2,862	0,42	0,209	0,38	1,301	0,25	0,204	0,34
7	0,7988	0,0956	2,383	0,23	0,132	0,01	0,424	0,41	0,080	0,02

Tabla 6.16: Utilidad media y tiempos de ejecución

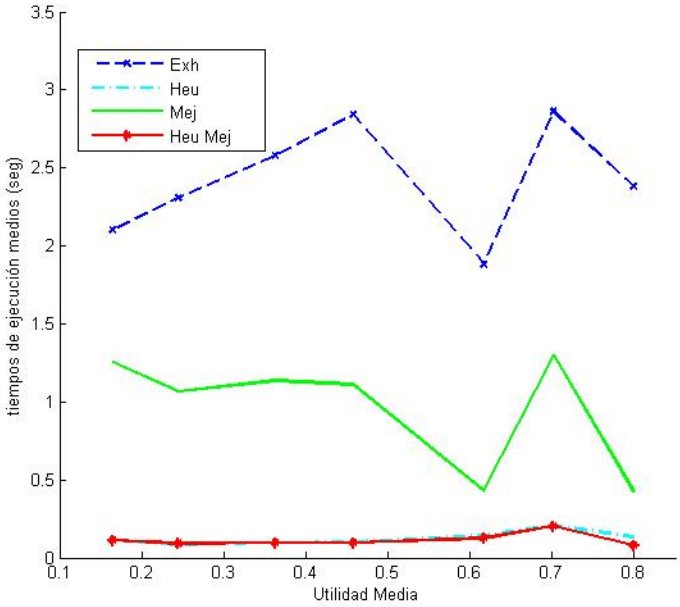

Figura 6.14: Tiempos de ejecución frente a la utilidad para aplicación con servicios en paralelo

Respecto al cálculo del número de evaluaciones necesario para cada algoritmo, se han obtenido los resultados que se muestran en la tabla 6.17 y en la figura 6.15. Se puede apreciar comparándolo con la prueba de los cinco nodos en serie (apartado 6.2.2) que obviamente el número de combinaciones máximo (exhaustivo) es el mismo ya que ambos tienen cinco servicios, pero se aprecia comparándolo con la figura 6.7 de los tiempos, que el algoritmo mejorado empieza a reducir combinaciones con utilidades menores. Esto es consecuencia de los paralelos antes mencionados.

Por último se mostraran las distancias relativas entre la combinación óptima del algoritmo exhaustivo y la combinación obtenida por los dos algoritmos heurísticos. Estos resultados se muestran en la tabla 6.18 y figura 6.16. Si procedemos a compararlo con la prueba de combinaciones medias (figura 6.8) llegaremos a la misma conclusión que en el caso de los tiempos de respuesta, que tiene un parecido mucho mayor con la prueba de aplicaciones con Ω baja (figura 6.4). Esto es consecuencia al mayor peso de las operaciones en serie sobre los paralelos.

nº ejec.	Utilidad media		Número de comb. evaluadas							
			Alg Exh		Alg Heu		Alg Mej		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1642	0,1508	1024,00	0,00	32,00	0,00	1024,00	0,00	32,00	0,00
2	0,2447	0,1804	1024,00	0,00	32,00	0,00	1020,24	4,49	32,00	0,00
3	0,3627	0,1843	1024,00	0,00	32,00	0,00	994,04	37,20	32,00	0,00
4	0,4572	0,1993	1024,00	0,00	32,00	0,00	869,22	128,98	32,00	0,00
5	0,6168	0,1469	1024,00	0,00	47,85	20,78	395,33	172,60	38,44	14,26
6	0,7012	0,1248	1024,00	0,00	32,00	0,00	313,58	36,65	24,64	8,41
7	0,7988	0,0956	1024,00	0,00	38,05	13,86	148,72	54,42	17,58	5,02

Tabla 6.17: Utilidad media y número de combinaciones evaluadas para aplicación con servicios en paralelo

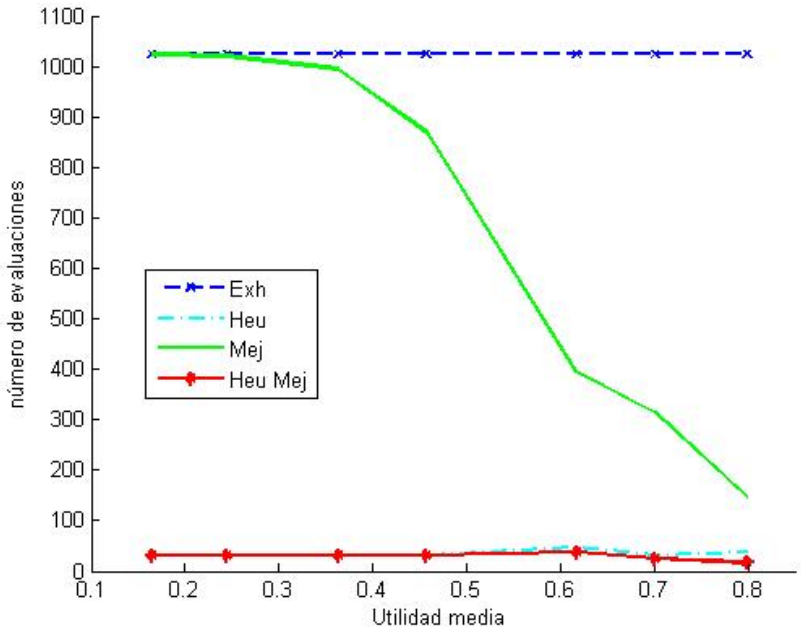

Figura 6.15: Número de evaluaciones frente a la utilidad para una Ω media con nodos en paralelo

nº ejec.	Utilidad media		Distancia a la óptima			
			Alg Heu		Alg Heu Mej	
	media	*var*	*media*	*var*	*media*	*var*
1	0,1642	0,1508	251,60	34,30	251,60	34,30
2	0,2447	0,1804	125,60	76,69	125,60	76,69
3	0,3627	0,1843	155,00	29,87	155,00	29,87
4	0,4572	0,1993	209,40	40,24	209,40	40,24
5	0,6168	0,1469	13,20	2,14	13,20	2,14
6	0,7012	0,1248	28,80	11,29	28,80	11,29
7	0,7988	0,0956	7,20	2,71	6,80	2,40

Tabla 6.18: Utilidad media y número de combinaciones entre la óptima y la seleccionada por los algoritmos heurísticos para una aplicación con servicios en paralelo

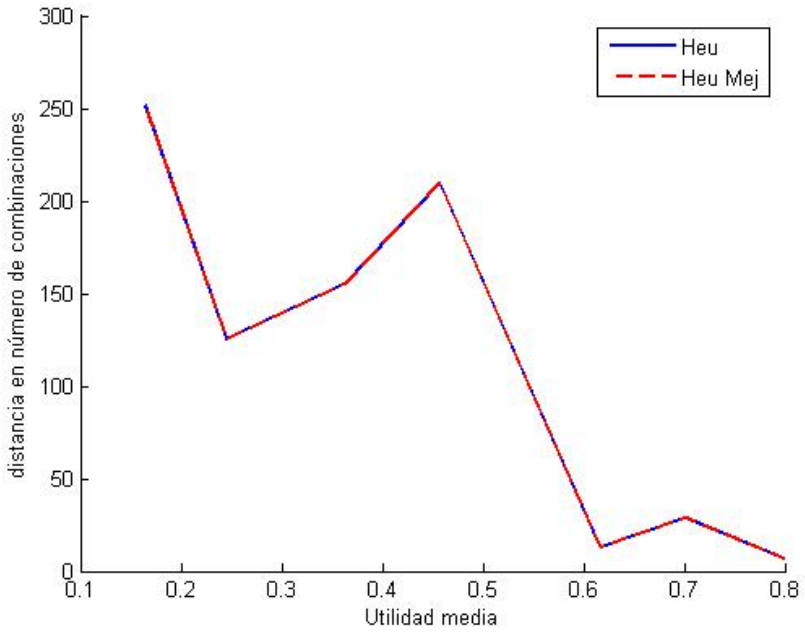

Figura 6.16: Número de combinaciones entre la óptima y la obtenida para una Ω media con servicios en paralelo

6.3. Resultados obtenidos

Para finalizar el estudio de los distintos algoritmos y para facilitar las conclusiones se muestran a continuación la comparativa del uso de los cuatro algoritmos en función del tiempo de ejecución y en función del número de evaluaciones necesarias.

6.3.1. Tiempos

Algoritmo Exhaustivo Realizando en un solo gráfico una comparación de los resultados obtenidos para el caso del uso del algoritmo exhaustivo, reflejados en tiempo se obtiene la figura 6.17.Como se puede apreciar en la imagen, no se deberá utilizar el algoritmo exhaustivo en ningún caso para números de combinaciones altas, debido al gran consumo de tiempo en su ejecución. Para evaluarlo en el resto de los casos se puede observar la figura 6.18. En ella se puede observar que su uso será muy recomendable, debido a sus características, cuando se trabaje con cargas de combinaciones bajas. Para el caso de Ω media habría que compararlo con el resto de los algoritmos.

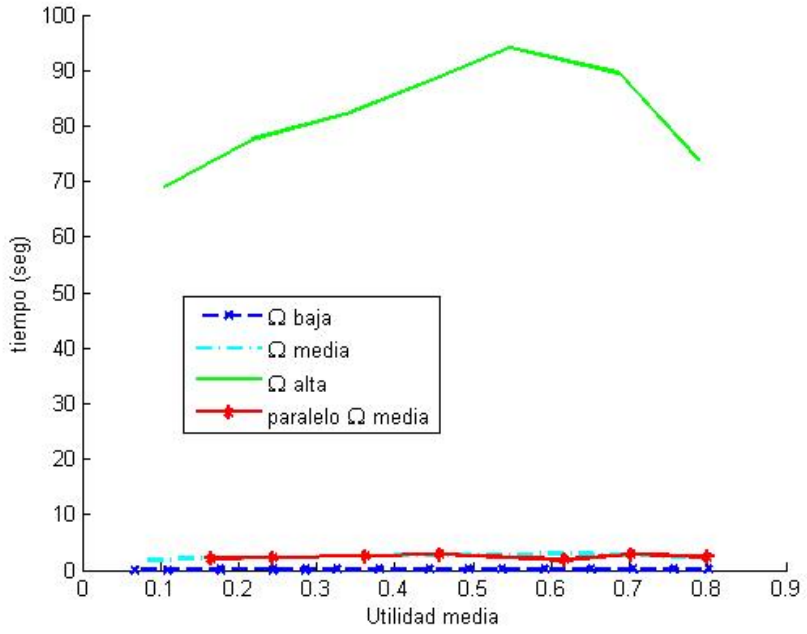

Figura 6.17: Tiempo en seg. del algoritmo exhaustivo para distinta Ω

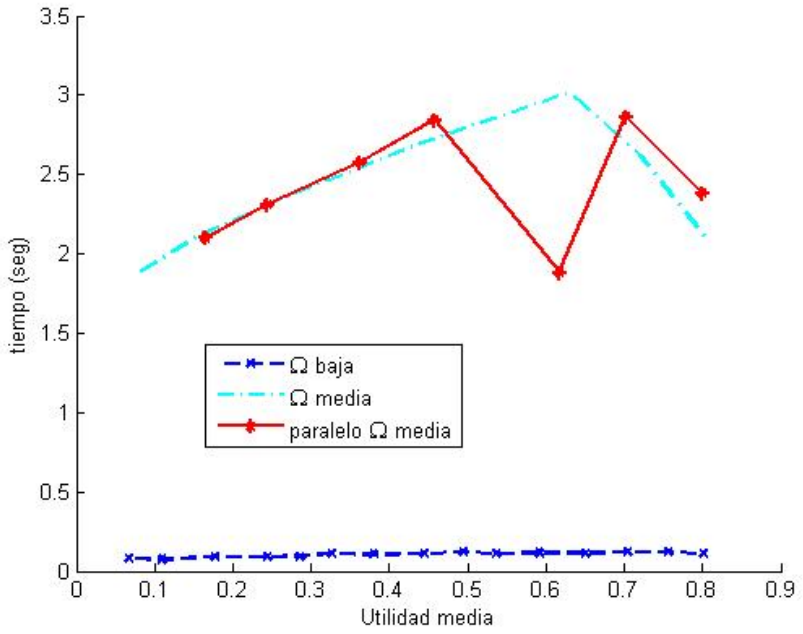

Figura 6.18: Tiempo en seg. del algoritmo exhaustivo para distintas Ω

Algoritmo Heurístico En el caso del algoritmo heurístico el grafico de comparación de tiempos de ejecución es el representado en la figura 6.19.

En la gráfica 6.19, se puede observar que solo en el caso de estar trabajando con Omegas elevadas el algoritmo heurístico nos dará perores resultados, pero en ningún caso estos serán no utilizables. Además se puede comprobar que el uso de este algoritmo, exclusivamente basándonos en su tiempo de ejecución, es recomendable para el resto de los casos.

Algoritmo Mejorado Si evaluamos el algoritmo mejorado el grafico de comparación de tiempos de ejecución es el representado en la figura 6.20.

Como se puede observar en la gráfica 6.20, este algoritmo no se debe utilizar bajo ningún concepto en el caso de estar trabajando con sistemas ya cargados y estar evaluando Omegas altas, ya que como podemos apreciar el tiempo de ejecución va aumentando al aumentar los valores de las características antes citadas. En el caso de combinaciones medias y bajas se puede ver mejor reflejado en la figura 6.21, en la cual observamos que para una Ω media este algoritmo es ideal, ya que aparte de obtener la combinación óptima, tienen tiempos de ejecución bajos.

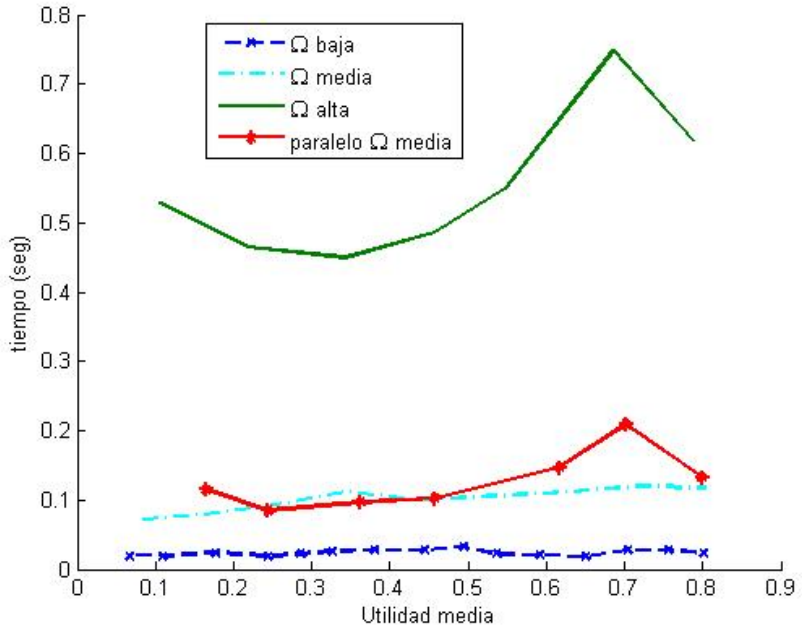

Figura 6.19: Tiempo en seg. del algoritmo heurístico para distintas Ω

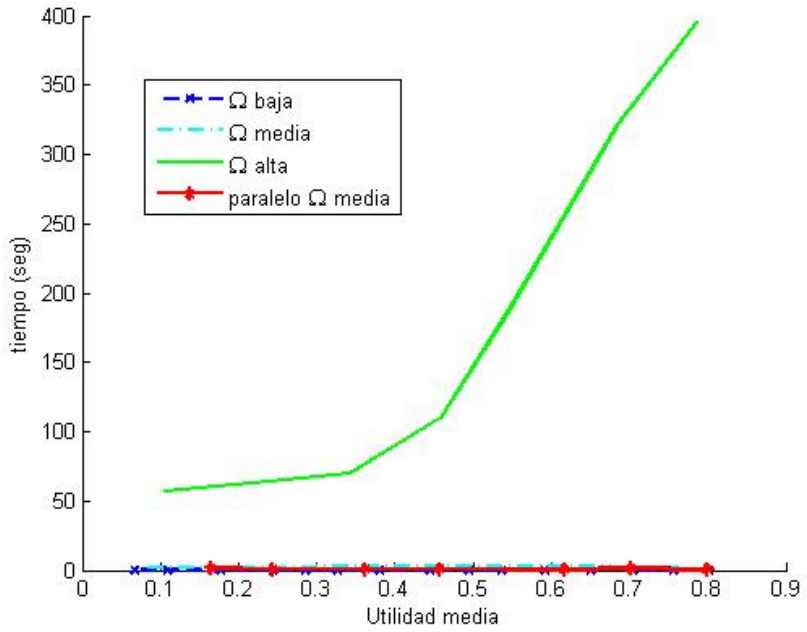

Figura 6.20: Tiempo en seg. del algoritmo mejorado para distinta Ω

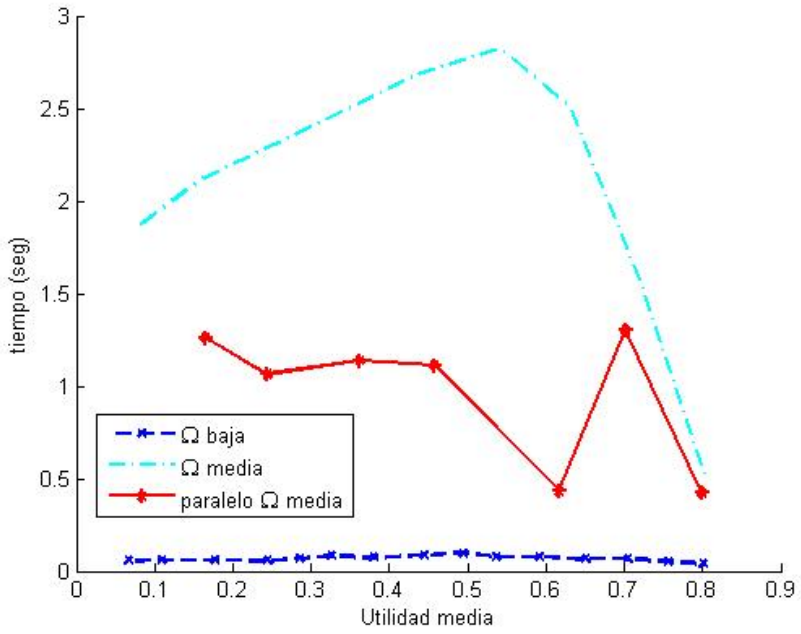

Figura 6.21: Tiempo en seg. del algoritmo mejorado para distinta Ω

Algoritmo Heurístico Mejorado Finalmente observaremos los tiempos obtenidos para la ejecución del algoritmo heurístico mejorado. Estos se muestran en la figura 6.22. En la figura se puede apreciar que los tiempos obtenidos son de gran interés, sobre todo para sistemas a punto de saturación (utilidades>0,7), siendo estos no tan buenos para utilidades medias (entre 0,4 y el 0,7 antes citado).

6.3.2. Número de Evaluaciones

Para el caso de las comparaciones del número de evaluaciones, los resultados conjuntos para cada algoritmo son los siguientes:

Algoritmo Exhaustivo Como se puede apreciar en la figura 6.23, el número de evaluaciones en caso del algoritmo exhaustivo confirma lo expuesto en el apartado 3.4.1, es decir, independientemente de la utilidad o de otros factores, este algoritmo comprobará el número total de combinaciones posibles. Como podemos apreciar el número de combinaciones crece potencialmente al ir aumentando el número de servicios y/o el número de perfiles de

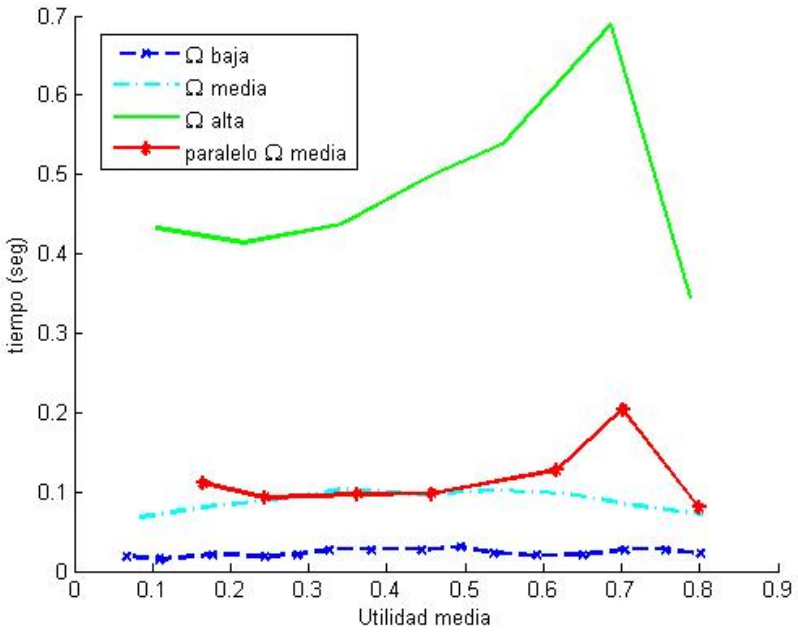

Figura 6.22: Tiempo en seg. del algoritmo heurístico mejorado para distinta Ω

cada nodo. Esto se puede ver reflejado en la ecuación 6.3 que muestra como se calcula el número de combinaciones posibles.

$$num_{comb} = \prod_{i=1}^{N} P_i \tag{6.3}$$

donde i va recorriendo los N servicios/operaciones que tiene la aplicación, y P_i se refiere al número de perfiles que posee cada servicio.

Algoritmo Heurístico En el caso del algoritmo heurístico el grafico de comparación de número de evaluaciones es el representado en la figura 6.24.En esta gráfica podemos apreciar como para la mayor parte de tiempo el sistema solo evaluará el número de combinaciones correspondiente a la selección de un solo bloque (dos perfiles) por cada servicio. Esto hace que el número de evaluaciones sea mucho menor que en el caso anterior. Cuando el sistema empieza a entrar en saturación es necesaria la evaluación de algún otro bloque en algún otro servicio.

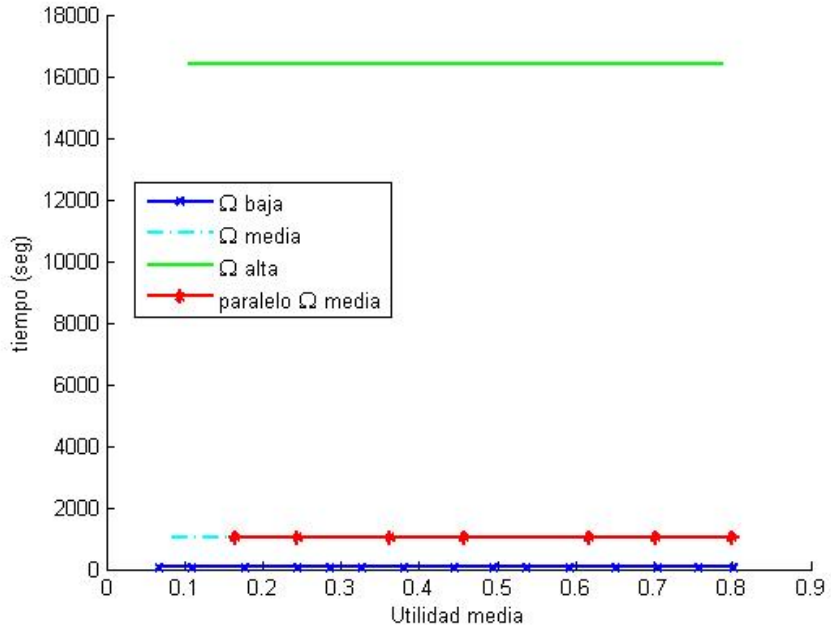

Figura 6.23: Número de evaluaciones del algoritmo exhaustivo para distinta Ω

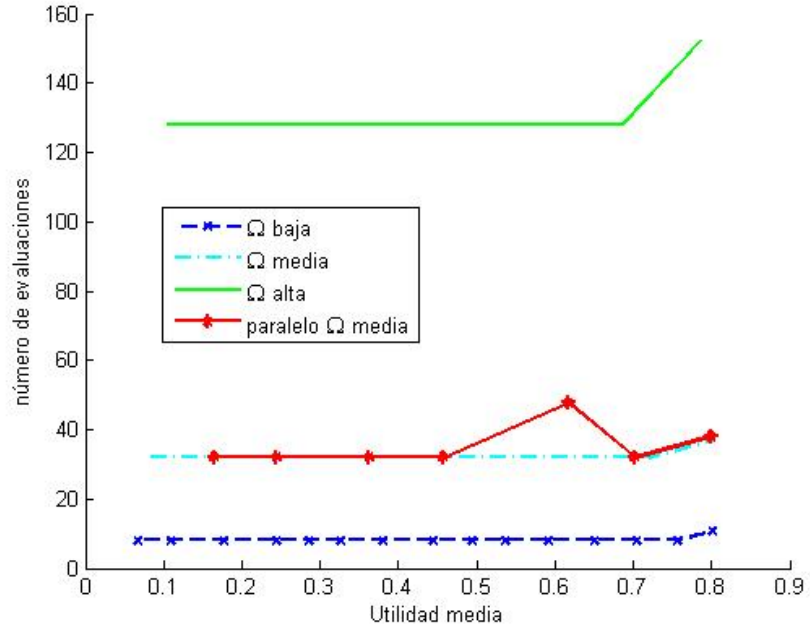

Figura 6.24: Número de evaluaciones del algoritmo heurístico para distinta Ω

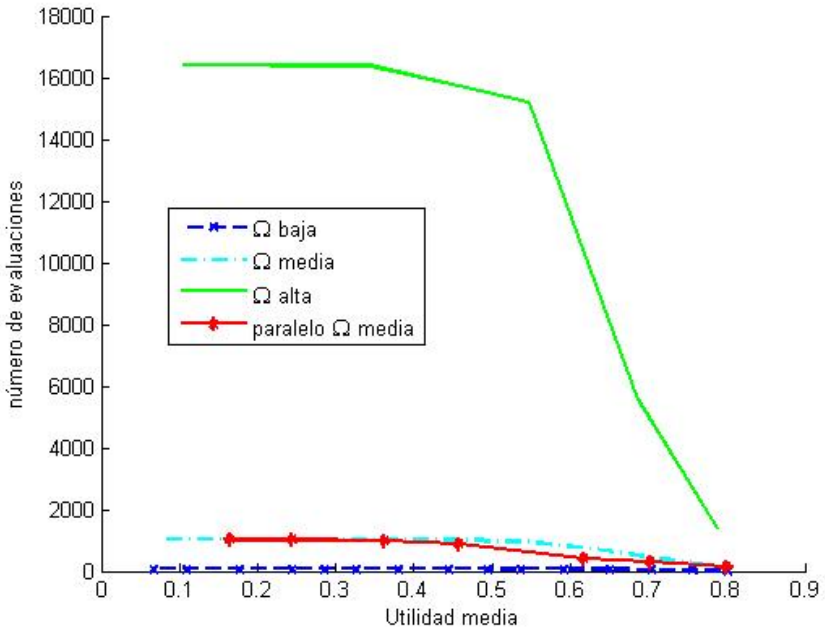

Figura 6.25: Número de evaluaciones del algoritmo mejorado para distinta Ω

Algoritmo Mejorado El grafico del número de combinaciones a evaluar para el algoritmo mejorado está reflejado en la figura 6.25. En él se puede observar que este algoritmo va reduciendo significativamente el número de combinaciones evaluadas cuando va aumentando la carga del sistema.

Algoritmo Heurístico Mejorado Finalmente observaremos el número de evaluaciones para las distintas ejecuciones del algoritmo heurístico mejorado. Estos se muestran en la figura 6.26. Podemos apreciar que como ya se comento es una mezcla de los dos algoritmos anteriores. Tiene la característica de un número bajo de evaluaciones del heurístico y que cuando se va acercando el sistema a la saturación, estas evaluaciones van disminuyendo (característica del mejorado).

6.3.3. Error entre la combinación óptima y la seleccionada

Finalmente nos queda por mostrar los distintos gráficos comparando el error cometido entre la combinación óptima y la seleccionada, para los algoritmos heurísticos que son los únicos que no han de seleccionar en todo momento la combinación óptima. Este error se

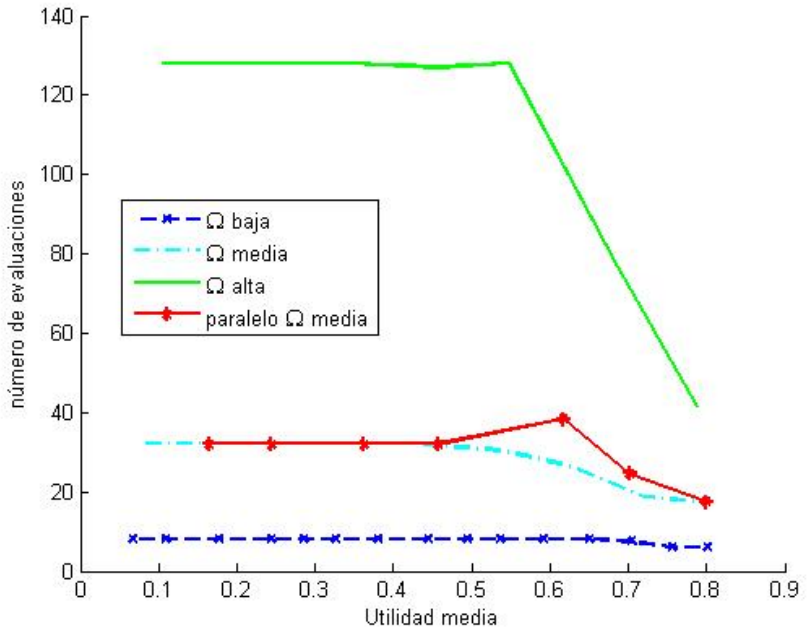

Figura 6.26: Número de evaluaciones del algoritmo heurístico mejorado para distinta Ω

medirá en distancia entre las combinaciones. Siendo la distancia el número de combinaciones existentes entre una y otra, cuando estas está ordenadas de menor a mayor tiempo de respuesta.

En primer lugar se evalúan los resultados para el algoritmo heurístico. Como se puede apreciar en la gráfica 6.27, excepto para un número bajo de combinaciones, en un principio (hasta utilidades mayores de 0,3), la combinación seleccionada será la óptima. Con ello se observa la buena elección de la figura de merito parcial y su gran relación con la figura de merito global. Su uso para utilidades medias queda empeora en el caso de estar trabajando con aplicaciones con un número medio/alto de combinaciones, reduciéndose esa distancia cuando el sistema se va encaminando a su saturación (utilidades mayores que 0,6).

Para el algoritmo heurístico mejorado (figura 6.28) se puede realizar la misma reflexión que para el algoritmo heurístico (figura 6.27) ya que ambos toman valores idénticos o muy similares.

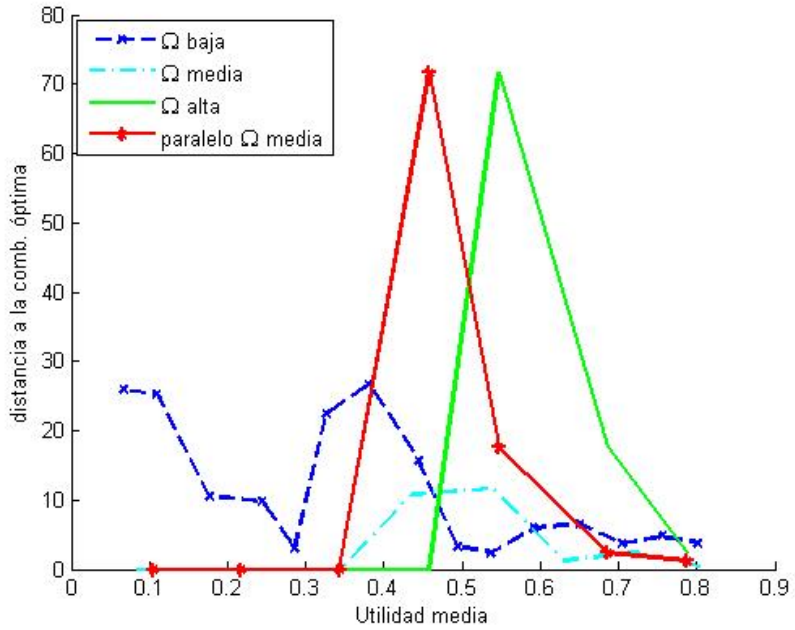

Figura 6.27: Distancia en combinaciones entre la combinación heurística y la óptima para distinta Ω para el algoritmo héurístico

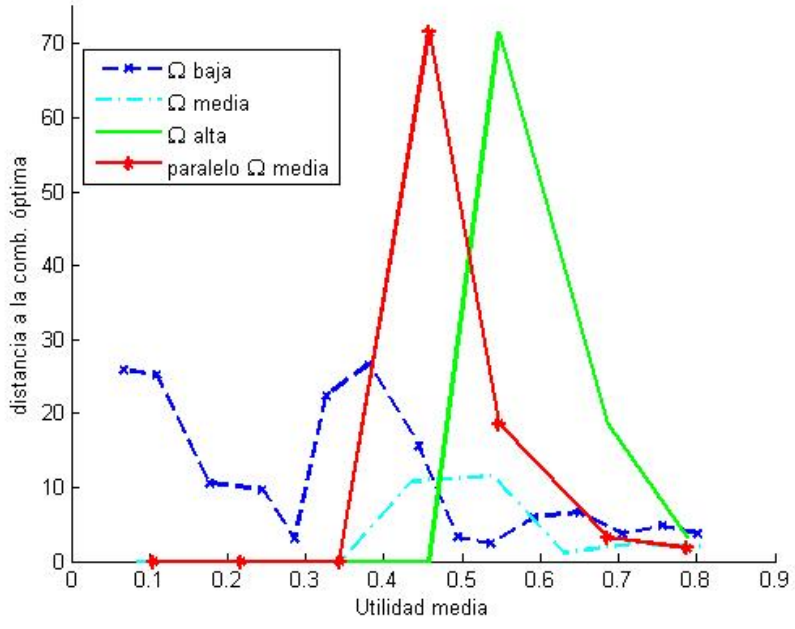

Figura 6.28: Distancia en combinaciones entre la combinación heurística mejorada y la óptima para distinta Ω para el algoritmo héurístico mejorado

6.4. Conclusión

Para concluir creemos necesario resumir toda la información anterior en la tabla 6.19, en la cual se mostrara en qué casos es mejor la utilización de cada algoritmo, teniendo en cuenta, tanto la Ω de la aplicación, como la utilidad media del sistema.

		Utilidad			
		0 - 0,3	0,3 - 0,5	0,5 - 0,7	0,7 - 1
Ω	baja	E, M	E, M	E, M	E, M
	media	H, HM	H, HM	M, HM	M, HM
	alta	H, HM	H, HM	HM	HM

Tabla 6.19: Tabla resumen de los algoritmos á utilizar para las distintas situaciones

En primer lugar hay que definir la nomenclatura utilizada en la tabla, donde 'E' hará referencia al algoritmo exhaustivo, 'H' al algoritmo heurístico, 'M' al mejorado y 'HM' al heurístico mejorado. Respecto a la tabla hay que destacar, que aunque para una Ω baja cualquier algoritmo muestra respuestas válidas, es recomendable el uso del algoritmo exhaustivo o en su defecto el mejorado ya que aportan en todo momento la combinación óptima. En el caso de Omegas medias hemos destacado para utilidades bajas, la utilización de cualquiera de los heurísticos ya que aparte de tardar un tiempo mucho menor, dan con la combinación optima. Aunque para utilizaciones medias no encuentran la utilización óptima, creemos que el hecho de acotar el tiempo de ejecución, al reducir la Ω, es más favorable para un sistema de tiempo real, por lo que haríamos uso de ellos. En el caso de utilizaciones altas creemos que sería de gran utilidad el uso del mejorado o el heurístico mejorado.

Finalmente para Omegas elevadas, sólo se podría hacer uso de cualquiera de los heurísticos ya que el tiempo de ejecución de los algoritmos exhaustivos es muy elevando y no es factible para la utilización en sistemas de tiempo real.

Como se ha comprobado que el tiempo de ejecución está estrechamente relacionado con la Ω.

Respecto al uso de aplicaciones con servicios en paralelo, han de ser tratadas como el

resto de las aplicaciones sencillas (solo servicios en serie), ya que como hemos comprobado tienen gran similitud con estas, y sus respuestas ante los distintos algoritmos son idénticas a las estudiadas para las aplicaciones en serie.

Capítulo 7

Sistema completo

En este último capítulo de evaluación se muestran los resultados obtenidos al evaluar un sistema completo, que trabaje con los cuatro algoritmos que se han implementado. Para ello vamos a realizar una serie de experimentos que mostrarán los resultados obtenidos a evaluar el sistema completo para distintas configuraciones, siempre persiguiendo que el sistema tenga un tiempo de ejecución bajo, que es una de las características que suele desear un usuario.

7.1. Introducción

Apoyándonos en el apartado de conclusiones del capítulo anterior 6.4, hemos desarrollado un último sistema que será más complejo que todos los anteriores. Este sistema contendrá los cuatro algoritmos desarrollados y dependiendo la situación en la que se encuentre el sistema general decidirá la planificación utilizando uno de los cuatro algoritmos. Lo que se pretende es simular un sistema real, en el cual el usuario no se vea obligado a conocer y posteriormente elegir cuál de los cuatro algoritmos se adapta mejor a su aplicación, o cual debería aplicar si tuviese diferentes aplicaciones. La característica principal del programa que se ha desarrollado es que proporcionará una combinación óptima o muy próxima, en un tiempo de ejecución acotado y bajo. Se ha decidido tomar el tiempo de ejecución como la característica más importante del mismo debido a que suele ser una de las condiciones que los usuarios suelen imponer. Si se quisiera solo tener en cuenta la

obtención de la mejor combinación solo sería necesario el uso de los algoritmos Exhaustivo y Mejorado, los cuales nos darían la óptima. En el caso de querer dotar al sistema de capacidad de decisión sobre otras características, sería necesario definir nuevas figuras de mérito. Esto formaría parte de una posible línea de trabajo futuro.

7.1.1. Definición de las pruebas

En este último experimento se van a realizar las mismas pruebas presentadas en el capitulo anterior (apartado 6.1.1). Es decir se va a evaluar las utilidades de los nodos físicos, los tiempos de respuesta, los tiempos de ejecución, el numero de combinaciones evaluadas para aplicaciones con Ω bajas, medias y altas. En este caso no evaluaremos el error que se comete al seleccionar un algoritmo heurístico, ya que este error fue analizado en el capitulo 6, por lo que simplemente dependiendo las características del sistema en ese instante, observando las graficas 6.4, 6.8 y 6.4, podríamos obtener el error cometido, asi como en las gráficas resumen 6.27, 6.28, del apartado 6.3.3. Asimismo como se ha introducido anteriormente, la característica principal de este sistema es que acota el tiempo de ejecución a valores bajos, por lo tanto aunque se cometa un error, este no es determinante, ya que aun así se consigue encontrar una combinación que sea planificable haciendo uso de los algoritmos heurísticos, los cuales tienen un tiempo de ejecución bastante menor.

7.1.2. Elección del algoritmo

A la hora de elegir el algoritmo a seleccionar para la evaluación de la aplicación, se ha tenido en cuenta las conclusiones adquiridas a la finalización de los pruebas presentadas en el capitulo anterior (6.4). Para ello se creó una tabla (6.19) donde se expresaban los algoritmos que se deberían seleccionar dependiendo las características del sistema y del algoritmo. En esta tabla se mostraban los algoritmos más recomendados debido a sus características. Para este experimento, ha sido necesario quedarse con un solo algoritmo para cada situación, ya que lo que se pretende es que la propia función decida por el usuario que algoritmo será más interesante utilizar, por lo tanto se ha realizado la selección, mostrada en la tabla 7.1, de los algoritmos para las diferentes características.

		Utilidad			
		0 - 0,3	0,3 - 0,5	0,5 - 0,7	0,7 - 1
Ω	baja	E	E	M	M
	media	H	H	HM	M
	alta	H	H	HM	HM

Tabla 7.1: Tabla resumen de los algoritmos á utilizar para las distintas situaciones

Como se puede apreciar al comparar la tabla finalmente seleccionada 7.1, frente a la inicial 6.19, podemos observar que para bajas Utilidades medias (Utilidad <0,5) se ha seleccionado el algoritmo Exhaustivo frente al Mejorado, así como el Heurístico frente al Heurístico mejorado. Esto se debe a que lo normal cuando se está trabajando con utilidades bajas es que durante la ejecución de estos algoritmos no existirán combinaciones no planificables, por lo tanto no tiene sentido aplicar una mejora cuyo propósito es encontrar patrones de no planificabilidad para evitar evaluaciones. Con la selección de los algoritmos sin mejorar, nos ahorramos la evaluación de la parte de código correspondiente a la mejora y por lo tanto concluyendo en un menor tiempo de ejecución. Al contrario de lo que descrito anteriormente, para el caso de utilidades altas es más razonable el uso de los algoritmos mejorados (Mejorado y Heurístico Mejorado) ya que cuando se está trabajando con el sistema más cargado, las posibilidades de encontrar patrones de no planificabilidad son mayores, y por lo tanto podremos ahorrar al sistema evaluar combinaciones que ya sabemos de antemano que no van a resultar planificables. En el caso concreto de una utilidad entre 0,5 y 0,7 y para una Ω media, se ha decidido usar el algoritmo Heurístico Mejorado que aunque nos proporciona un error en la elección de la combinación optima, si lo que se desea es acotar el tiempo de ejecución, este nos proporcionará un tiempo bajo. En el caso de trabajar para las mismas Ω, pero con utilidades mayores, se ha decidido usar el algoritmo Mejorado ya que tiene tiempos de ejecución acotados bajos debido al gran número de patrones de no planificabilidad que reducen considerablemente el tiempo que se invierte en las evaluaciones. Además sabemos que este algoritmo proporciona siempre la combinación óptima.

7.2. Pruebas realizadas

Al igual que en las pruebas realizadas en el capítulo 6 se van a dividir estas dependiendo la Ω que tenga la aplicación a evaluar. Como se explico en ese mismo capítulo se ha denominado Ω al producto del número de combinaciones por el número de servicios de la aplicación, donde el número de combinaciones se puede calcular en la ecuación 6.1. En este experimento se van a usar las mismas aplicaciones usadas en dicho capítulo a excepción de la aplicación con nodos en paralelo, la cual como se comprobó han de ser tratadas como el resto de las aplicaciones sencillas que sólo tienen servicios en serie, ya sus respuestas son idénticas, por lo tanto no será necesario estudiarla nuevamente.

Respecto al modo de realizar las pruebas es idéntico al de las pruebas precedentes, es decir, se van ejecutando aplicaciones en el sistema hasta obtener cinco aplicaciones no planificables, lo cual significa haber obtenido una utilización máxima del sistema. La razón de porque se decidió escoger cinco aplicaciones no planificables como punto de parada está explicada en el apartado 6.1.1. Este bucle de aplicaciones se realiza durante al menos diez ocasiones para así poder realizar sobre los resultados valores medios y sus correspondientes varianzas. Aunque los tiempos de ejecución en esta prueba son menores, se ha considerado que con 10 veces es suficiente ya que podemos comprobar que aunque estamos trabajando con tareas, mensajes y *deadlines* aleatorios, los resultados que obtenemos son siempre similares ya que las varianzas son pequeñas.

7.2.1. Aplicaciones con Ω baja

En primer lugar, vamos a evaluar una aplicación con una Ω baja. Esta sería por ejemplo la utilizada en el capitulo anterior que tenía tres servicios en serie y que aportaba 64 combinaciones posibles, por lo que su Ω es 192, la cual se encuentra dentro del primer grupo de Ω bajas, es decir menores de 1050.

Para esta prueba se han realizado 10 ejecuciones del bucle que nos ha proporcionado 14 aplicaciones planificables, antes de declarar el sistema como saturado. En la tabla 7.2 podemos apreciar la utilidad de los tres nodos físicos junto con el tiempo de respuesta medio, en unidades de tiempo. Se puede apreciar como los nodos van haciendo repartición una repartición de carga entre ellos. Asimismo se puede observar que existen aplicaciones

nº ejec.	Utilidad						t resp	
	n.fis 1		n.fis 2		n.fis 3			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,0000	0,0000	0,0721	0,0104	0,1803	0,0260	42,00	0,00
2	0,0869	0,0065	0,1015	0,0094	0,2097	0,0249	43,00	0,00
3	0,1608	0,0174	0,1757	0,0151	0,2472	0,0246	48,76	4,90
4	0,2245	0,0320	0,2398	0,0283	0,2796	0,0262	54,50	7,48
5	0,2607	0,0290	0,3117	0,0217	0,3155	0,0245	59,92	11,66
6	0,2977	0,0276	0,3712	0,0187	0,3679	0,0285	87,58	14,70
7	0,3848	0,0614	0,4068	0,0196	0,4183	0,0411	98,15	23,15
8	0,4419	0,0628	0,4634	0,0227	0,4469	0,0399	94,41	18,09
9	0,5236	0,0543	0,5250	0,0281	0,4821	0,0439	97,24	25,58
10	0,5698	0,0296	0,5901	0,0333	0,5513	0,0690	93,69	7,77
11	0,5903	0,0192	0,6450	0,0264	0,6322	0,0397	127,75	23,58
12	0,6945	0,0262	0,6799	0,0251	0,6671	0,0395	121,08	21,35
13	0,7499	0,0536	0,7394	0,0223	0,7317	0,0756	139,58	16,48
14	0,8056	0,0352	0,7954	0,0195	0,7629	0,0740	162,26	39,55

Tabla 7.2: Utilidades y tiempo de respuesta para una Ω baja

posteriores con menor tiempo de respuesta que las predecesoras. Esto como se explicó anteriormente es debido a la aleatoriedad de los *deadlines* utilizados.

Para concluir este estudio, se ha decidido expresar en un tabla, 7.3, los diferentes resultados de la prueba. En ella podemos apreciar las medias y varianzas de los tiempos de ejecución y el número de combinaciones evaluadas, así como el algoritmo que se ha usado en cada caso. Para comprender que algoritmo ha sido usado, hay que definir que el '1' representa el Exhaustivo, el '2' el Heurístico, el '3' el Mejorado y finalmente el '4' el Heurístico Mejorado. También cabe destacar que si sobre una utilidad media aparecen dos algoritmos, estos indican que al estar la utilidad en una frontera entre franjas, a lo largo de la evaluación de las 10 repeticiones se ha observado que ambos algoritmos se han ejecutado.

nº ejec.	**Utilidad media**		**A.usado**	**t. ejecución (seg)**		**num. evaluaciones**	
	media	*var*		*media*	*var*	*media*	*var*
1	0,0850	0,0766	1	0,078	0,00	64,00	0,00
2	0,1228	0,0575	1	0,090	0,01	64,00	0,00
3	0,1911	0,0426	1	0,091	0,01	64,00	0,00
4	0,2469	0,0369	1	0,091	0,01	64,00	0,00
5	0,2948	0,0353	1	0,099	0,01	64,00	0,00
6	0,3439	0,0421	1	0,100	0,01	64,00	0,00
7	0,4031	0,0458	1	0,103	0,01	64,00	0,00
8	0,4506	0,0456	1	0,106	0,01	64,00	0,00
9	0,5098	0,0478	1, 3	0,113	0,01	64,00	0,00
10	0,5701	0,0496	3	0,115	0,01	62,75	2,40
11	0,6221	0,0379	3	0,106	0,02	53,41	13,43
12	0,6804	0,0328	3	0,096	0,03	47,95	11,94
13	0,7403	0,0555	3	0,076	0,02	31,86	7,47
14	0,7878	0,0515	3	0,039	0,01	16,81	2,53

Tabla 7.3: Utilidad media, algoritmo usado, tiempo de ejecución y nº de evaluaciones para una Ω baja

Podemos observar en la tabla 7.3, como los tiempos de ejecución están acotados por debajo de las 120 milésimas, y que se puede apreciar el buen uso del algoritmo Mejorado que basándose en sus patrones hace que se evalúen menos combinaciones que las máximas.

7.2.2. Aplicaciones con Ω media

En segundo lugar, vamos a evaluar una aplicación con una Ω media. Esta sería la utilizada en el capítulo anterior que tenía cinco servicios en serie y que aportaba 1024 combinaciones posibles, por lo que tiene una Ω de 5120, la cual se encuentra dentro del grupo de Ω medias, es decir mayores de 1050 y menores de 15650.

Para la prueba se realizaron 10 ejecuciones del bucle que nos ha proporcionado 9 aplicaciones planificables, antes de declarar el sistema como saturado. En la tabla 7.4 podemos apreciar la utilidad de los tres nodos físicos junto con el tiempo de respuesta medio, en unidades de tiempo. Asimismo, en esta nueva prueba se pueden apreciar las características señaladas para la prueba anterior.

nº ejec.	**Utilidad**						**t resp**	
	n.fis 1		**n.fis 2**		**n.fis 3**			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1253	0,0180	0,0627	0,0090	0,0627	0,0090	83,91	4,00
2	0,2691	0,0337	0,1345	0,0168	0,1345	0,0168	94,77	16,00
3	0,2879	0,0482	0,3050	0,0345	0,2060	0,0168	101,75	15,05
4	0,3935	0,0467	0,4465	0,0579	0,2877	0,0082	123,95	18,63
5	0,5272	0,0443	0,5524	0,0713	0,3620	0,0169	145,60	27,64
6	0,6420	0,0189	0,6521	0,0510	0,4152	0,0266	182,32	39,21
7	0,7745	0,0229	0,7185	0,0517	0,4819	0,0199	234,31	32,13
8	0,8530	0,0212	0,7929	0,0295	0,5728	0,0187	252,36	59,97
9	0,8872	0,0212	0,8502	0,0401	0,7491	0,0346	232,51	59,77

Tabla 7.4: Utilidades y tiempo de respuesta para una Ω media

En la tabla 7.5 se expresan las medias y varianzas de los tiempos de ejecución y el

número de combinaciones evaluadas, así como el algoritmo que se ha usado en cada caso. En esta prueba podemos observar que se han ejecutado tres algoritmos, el '2' el Heurístico en un principio, el '4' el Heurístico Mejorado posteriormente y finalmente el '3' el Mejorado. Asimismo podemos observar que al estar usando principalmente los heurísticos no se llegan a evaluar todas las combinaciones posibles. Para el algoritmo Mejorado, en el momento en el cual se hace uso de él, sus patrones entran en funcionamiento, ya que el sistema está bastante saturado y por lo tanto habrá gran cantidad de combinaciones no planificables que dan como resultado un número de evaluaciones también bajo.

nº ejec.	**Utilidad media**		**A.usado**	**t. ejecución (seg)**		**num. evaluaciones**	
	media	*var*		*media*	*var*	*media*	*var*
1	0,0843	0,0324	2	0,066	0,01	32,00	0,00
2	0,1808	0,0682	2	0,078	0,00	32,00	0,00
3	0,2685	0,0567	2	0,087	0,01	32,00	0,00
4	0,3781	0,0802	2	0,091	0,01	32,00	0,00
5	0,4829	0,0992	2	0,099	0,01	32,00	0,00
6	0,5708	0,1150	2, 4	0,100	0,01	32,00	0,00
7	0,6592	0,1317	4	0,090	0,01	26,87	2,61
8	0,7400	0,1228	4	0,074	0,01	19,88	4,76
9	0,8295	0,0668	3	0,425	0,05	72,75	10,37

Tabla 7.5: Utilidad media, algoritmo usado, tiempo de ejecución y nº de evaluaciones para una Ω media

7.2.3. Aplicaciones con Ω alta

Finalmente se evaluó una aplicación con una Ω alta. Se usó la aplicación con siete servicios en serie y que aportaba 16384 combinaciones posibles, por lo que tiene una Ω de 114688, la cual pertenece al grupo Ω altas, es decir mayores de 15650.

Para esta última prueba se realizaron 10 ejecuciones del bucle que nos ha proporcionado 6 aplicaciones planificables, antes de declarar el sistema como saturado. En la tabla 7.6 se observa la utilidad de los tres nodos físicos junto con el tiempo de respuesta medio.

nº ejec.	Utilidad						t resp	
	n.fis 1		n.fis 2		n.fis 3			
	media	*var*	*media*	*var*	*media*	*var*	*media*	*var*
1	0,1552	0,0079	0,0776	0,0039	0,1552	0,0079	122,00	0,00
2	0,3151	0,0198	0,1576	0,0099	0,3151	0,0198	157,51	12,00
3	0,3566	0,0185	0,4042	0,0318	0,3979	0,0190	215,01	28,40
4	0,5033	0,0225	0,5542	0,0875	0,5258	0,0340	227,27	40,58
5	0,6134	0,0195	0,6932	0,0347	0,6139	0,0232	254,84	25,40
6	0,7324	0,0174	0,8383	0,0470	0,7263	0,0218	327,32	39,84

Tabla 7.6: Utilidades y tiempo de respuesta para una Ω alta

En la tabla 7.7 se expresan las medias y varianzas de los tiempos de ejecución y el número de combinaciones evaluadas, así como el algoritmo que se ha usado en cada caso. En esta prueba se han ejecutado los algoritmos, Heurístico ('2') y Heurístico Mejorado ('4'), ya que como se demostró en las pruebas del capítulo anterior, son los únicos que nos pueden asegurar unos tiempos de ejecución acotados reducidos como deseamos.

nº ejec.	Utilidad media		A.usado	t. ejecución (seg)		num. evaluaciones	
	media	*var*		*media*	*var*	*media*	*var*
1	0,1295	0,0373	2	0,392	0,01	128,00	0,00
2	0,2631	0,0764	2	0,462	0,03	128,00	0,00
3	0,3870	0,0321	2	0,494	0,02	128,00	0,00
4	0,5305	0,0604	2	0,562	0,02	128,00	0,00
5	0,6407	0,0462	4	0,609	0,09	118,64	6,11
6	0,7663	0,0607	4	0,293	0,02	57,58	1,50

Tabla 7.7: Utilidad media, algoritmo usado, tiempo de ejecución y nº de evaluaciones para una Ω alta

7.3. Resultados obtenidos

Para finalizar el estudio y para facilitar las conclusiones, se muestran gráficamente los tiempos obtenidos así como las combinaciones a evaluar por el sistema dependiendo la Ω de la aplicación.

7.3.1. Tiempos

La gráfica de tiempos seria la siguiente:

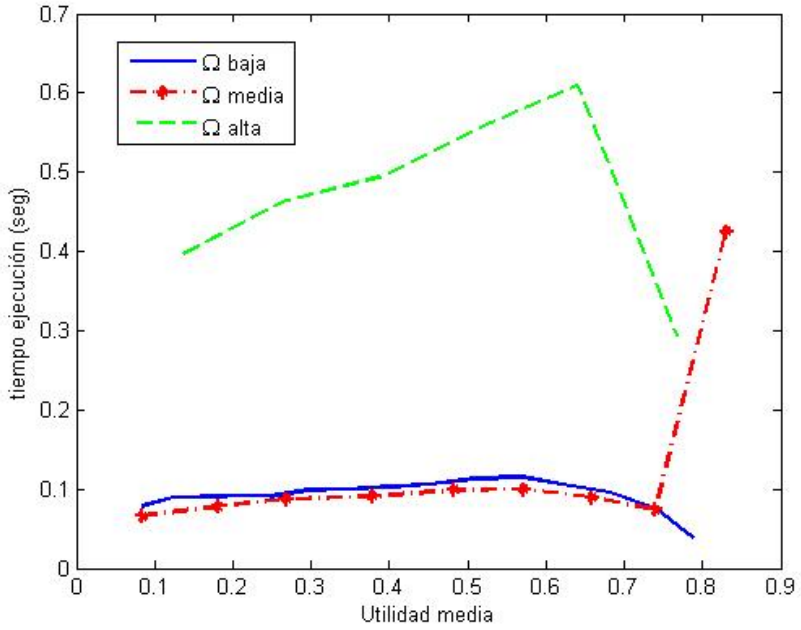

Figura 7.1: Tiempo en seg. para el sistema completo, y distintos valores de Ω

En ella podemos apreciar que el tiempo del sistema para las aplicaciones probadas está acotado a 0,6 segundos. Este tiempo será una cota máxima para estas aplicaciones ya que si aumentásemos la Ω iría aumentando el tiempo de ejecución como es lógico al ir aumentando las combinaciones a evaluar o los servicios de un sistema. También se puede observar perfectamente que tanto aplicaciones con Ω baja y alta, usan para utilidades altas un algoritmo mejorado de los usados para utilidades menores, sin embargo en el caso de Ω media, aunque se usa para utilidades altas el algoritmo Mejorado, al haberse utilizado antes un Heurístico hace que como se aprecia el tiempo aumente. Si se quisiera trabajar

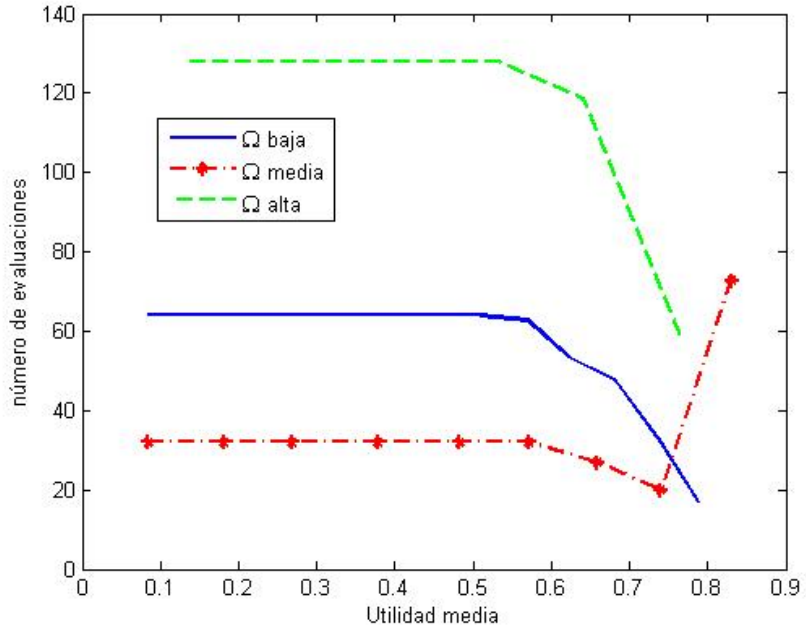

Figura 7.2: Número de evaluaciones para el sistema completo, y distintos valores de Ω

con tiempos de ejecución muy bajos, solo sería posible usando aplicaciones que tuviesen Ω bajas o medias, ya que para Ω más altas los tiempos como se ha demostrado empiezan a aumentar considerablemente.

Respecto al número de evaluaciones realizadas en cada uno de los casos, puede observarse la gráfica 7.2. Cabe destacar que como se comentó en las pruebas realizadas en el capítulo 6, que los tiempos de ejecución y el número de evaluaciones evaluadas estarán estrechamente relacionadas, ya que a mayor combinaciones que tengas que evaluar, mayor tiempo que se ha de invertir. Esto queda reflejado en la gráfica, así como el resto de características que se expresaron para la figura relativa a los tiempos (fig. 7.1).

7.4. Conclusión

Finalmente para concluir se puede destacar la importancia que pueden adquirir este tipo de sistemas para los diferentes usuarios, si estos son capaces de definir la figura de merito a la cual quieren dar mayor importancia. Esto ha quedado demostrado en el presente experimento, en el cual se ha desarrollado un programa con el cual se desea obtener un

tiempo de ejecución acotado y bajo, independientemente de la aplicación que se introduzca. Esto obliga al desarrollador a configurar el sistema de tal manera que la selección que se realice de los algoritmos a ejecutar en cada situación, permita obtener los requisitos impuestos. A partir de este experimento, empiezan a tomar verdadera importancia las figuras de merito, tanto parciales como globales, ya que su elección será la que nos permita satisfacer las expectativas de los usuarios a la hora de la obtención de la característica más importante del sistema.

Capítulo 8

Conclusiones y lineas futuras

Con este capítulo se concluye el presente proyecto fin de carrera. En él se van a resumir las ideas que se han planteado a lo largo del mismo, destacando las contribuciones que se han realizado, y terminando proponiendo las lineas de los posibles trabajos futuros

8.1. Conclusiones generales

El trabajo aquí presentado está encuadrado en las nuevas tendencias de aplicar soluciones de flexibilidad de paradigmas ya implementados, a sistemas distribuidos de tiempo real. Para ello se ha buscado continuar con los estudios iniciados en la tesis de Iria Estévez Ayres [1], implementando algoritmos que nos permitan aprovechar los estudios realizados en paradigmas como el de orientación a servicios, para el desarrollo de sistemas de tiempo real. Los principales objetivos serán poder asegurar a las aplicaciones garantías temporales, así como nuevas vías en la flexibilidad de las mismas que hagan a las aplicaciones tolerantes a fallos, y que ofrezcan capacidades para gestionar calidades de servicio. Por lo tanto lo que se ha pretendido es demostrar qué características del paradigma de orientación a servicios pueden ser adaptadas de forma exitosa a sistemas de tiempo real distribuidos, y que, dependiendo las necesidades de las aplicaciones, pueden existir diferentes algoritmos que gestionen el sistema para garantizar sus tiempos de respuesta y ejecución, así como la calidad de servicio que se quiera obtener.

Para conseguir todo lo anterior, se siguió el diseño de la tesis de Iria Estévez Ayres [1], en la cuál se definían modelos, tanto de sistema como de aplicación, servicio y perfil de servicio. Estos diseños se adaptaron e implementaron en lenguaje *Matlab*, del mismo modo que se hizo con los dos algoritmos que en dicha tesis se estudiaron. Para completar el análisis se diseñaron y desarrollaron dos nuevos algoritmos que mejoraban los dos anteriores. Todos estos algoritmos son de composición de aplicaciones que, haciendo uso de análisis de planificabilidad, permiten no sólo seleccionar perfiles de servicios en función de sus características funcionales, sino que también proporcionasen la planificación de las aplicaciones compuestas al mismo tiempo que aseguran que las prestaciones del sistema, tanto los nodos físicos, como las redes físicas, no se degradan al incluir una nueva aplicación en el mismo.

8.2. Resultados

Los estudios realizados y resultados obtenidos con la realización del presente proyecto se presentan a continuación:

1. **Estudio del estado del arte:** análisis de la situación actual en los sistemas de tiempo real, en especial de los sistemas distribuidos. Se abordó este estudio desde una perspectiva global para que cualquier usuario que desease introducirse en el tema pudiese comprender las decisiones que se fueron tomando en la realización del proyecto. Asimismo, como ya hemos destacado, se estudió el paradigma de orientación a servicios, centrándonos en las características que podrían ser aprovechadas de manera beneficiosa en sistemas de tiempo real. También se estudio las características que ofrecen los entornos distribuidos.

2. **Estudio de los modelos diseñados por Iria Estévez Ayres en [1]:** en esta parte se estudiarán los modelos, tanto del sistema general, como los modelos de servicio, perfil y aplicación. Todos ellos están enfocados a aplicaciones de tiempo real no crítico. Los modelos que se estudiaron tenían las siguientes características, [1]:

    - Las aplicaciones estarán definidas por los servicios que procesan y por las relaciones que existan entre éstos.

- Los servicios son definidos por su funcionalidad, y son implementados por perfiles, que tendrán unas características temporales distintas y que residirán en los nodos físicos.
- Cada uno de los perfiles es una única tarea en el nodo físico. Si un perfil forma parte de varias aplicaciones o es usado varias veces en la misma aplicación, serán implementadas por tareas distintas asociadas a ese perfil ya que serán invocaciones diferentes.
- Una aplicación en ejecución se definirá por un grafo ordenado de tareas y mensajes intercambiados entre éstas.
- Las aplicaciones estarán gobernadas por tiempo, es decir, las tareas y mensajes de cada una de las aplicaciones tendrán instantes de activación predefinidos y siempre mayores que los tiempos de respuesta en el peor caso de las acciones precedentes.

3. **Elección de composición:** se decidió la elección de la composición estática ya que se consideró que sería la más adecuada para las pretensiones del proyecto. Una composición estática se caracteriza porque una vez realizada no se modifica en tiempo de ejecución, por lo tanto 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.

4. **Estudio de los algoritmos diseñados en [1] por Iria Estévez Ayres:** se realizó un estudio de los dos algoritmos que se proponen en dicha tesis. Estos algoritmos de composición estarán diseñados para aplicaciones de tiempo real basadas en servicios. Los algoritmos dan solución a la búsqueda del camino óptimo de una aplicación, al mismo tiempo que aseguran la planificación del mismo, es decir, que se cumpla que todas las tareas y mensajes involucrados sean planificables y que la aplicación completa cumpla los requisitos temporales que se hayan marcado. Además se ha de observar que no se ponga en peligro el sistema completo debido a la instanciación de una nueva aplicación. Los dos algoritmos que se proponen en la tesis son el exhaustivo y un algoritmo heurístico.

5. **Adaptación del sistema, servicios, perfiles y aplicaciones:** se estudió las maneras de adaptar y definir, tanto el modelo de sistemas y aplicaciones, así como el de servicios y perfiles, a la futura implementación sobre lenguaje de programación *Matlab* sin que se comprometiesen las características expresadas en la tesis y resumidas en el punto 2.

6. **Implementación del sistema, servicios, perfiles y aplicaciones:** una vez adaptados fue necesario implementar en *Matlab* el sistema sobre el que se ejecutarían las aplicaciones. Por lo tanto, primeramente, se programaron los modelos de los servicios y mensajes, así como los soportes físicos, nodos físicos y redes físicas, sobre los cuales se deseaba simular la ejecución. Una vez realizada esta implementación, se hubo de proyectar las características de las futuras aplicaciones para que pudiesen ser verificadas sobre el sistema.

7. **Adaptación e implementación de los algoritmos:** implementación sobre *Matlab* de los dos algoritmos, exhaustivo y heurístico que se expresaron en el punto 4. Para ello fue necesario adaptarlos al nuevo lenguaje de programación evitando en todo momento modificar cualquiera de las características de los mismos.

8. **Estudio, diseño y desarrollo de nuevos algoritmos:** apoyándonos en los puntos anteriores, al haber sido necesario un estudio y adaptación de todo lo desarrollado en la tesis, se planteó la posibilidad de hacer mejoras sobre los algoritmos ya estudiados. Por lo tanto, se realizaron dos nuevos diseños que mejorarán las características de los algoritmos exhaustivo y heurístico. Estos nuevos algoritmos se han denominado, algoritmo mejorado y algoritmo heurístico mejorado los cuales también son desarrollados para su utilización en *Matlab*. Estos algoritmos favorecen la búsqueda de soluciones a la composición en un tiempo acotado. Además se ha comprobado que trabajan sobre los casos en que los algoritmos de [1] se mostraban más desfavorables.

9. **Implementación de una herramienta de simulación:** todos los puntos anteriores quedarán recogidos en el desarrollo de la herramienta que nos ha permitido interactuar con los algoritmos que se han implementado. Con ello se ha podido hacer un seguimiento de la ejecución de los mismos en diferentes situaciones y que ha ayudado a la realización de las pruebas necesarias para la presentación de resultados.

10. **Validación del comportamiento de los algoritmos propuestos:** se realizó una simulación de los algoritmos implementados para distintas configuraciones obteniéndose resultados satisfactorios que nos permitieron definir una serie de conclusiones para definir las aplicaciones para las cuales funcionará mejor cada algoritmo.

11. **Diseño de un sistema completo:** finalmente, apoyándonos en los resultados y las conclusiones obtenidas en las pruebas de cada algoritmo, se decidió la implementación de un sistema completo que delegase la decisión de selección del algoritmo a evaluar en el propio sistema. Esto facilita la posibilidad de un uso por parte de usuarios con el simple cumplimiento de las características de entrada en la arquitectura de la aplicación, las cuales están definidas en el apartado 5.3 y en el apéndice B, sin necesidad de conocer expresamente el funcionamiento de cada uno de los algoritmos.

8.3. Trabajos Futuros

Siguiendo el camino abierto por la tesis de Iria Estévez Ayres [1], en relación con la aplicación del paradigma de la orientación a servicios en sistemas de tiempo real, este proyecto fin de carrera ha servido para realizar un pequeño avance en su estudio. Además se ha contribuido al desarrollo de nuevos algoritmos y a la implementación de sistemas, algoritmos y aplicaciones sobre un nuevo lenguaje de programación. Por lo tanto, este nuevo proyecto deja aún abiertas líneas similares de investigación, así como contribuye a la apertura a nuevos trabajos que ayuden a la mejora de estos estudios.

8.3.1. Futuras lineas de investigación

Mejoras en los algoritmos

- Se abre una nueva puerta a la creación de nuevos algoritmos de composición, así como a la mejora de los ya existentes, pudiéndose hacer más eficiente el código, al igual que el propio diseño del algoritmo.

- Implantación de nuevas figuras de merito que puedan tener en cuenta factores como la utilidad global del sistema o la calidad de servicio de la aplicación, para su utilización en campos concretos de la aplicación.

- Desarrollo de algoritmos de composición que apliquen alguno de los heurísticos de asignación de prioridades existentes y que evalúen el coste que supone soportar esta característica.

- Mejora en las variables que definen que tipo de algoritmo se ha de usar. Estudio de la viabilidad de las variables "Utilidad media" y "Ω", a la hora de decidir los tipos de algoritmos a utilizar por un sistema completo orientado al uso por usuarios. Asimismo otro trabajo futuro de gran utilidad sería la realización de sistemas completos más complejos que el desarrollado en el capitulo 7 para facilitar el uso por usuarios.

Mejoras en el sistema

- La principal mejora a la que hay que enfrentarse, es dotar de flexibilidad al sistema implementado, para facilitar la posibilidad de aplicar composición dinámica, que nos facilitase una reconfiguración en tiempo de ejecución, así como una mayor tolerancia a fallos.

- Otro trabajo futuro de relativa importancia, es poder dotar al sistema la capacidad de ejecutarse sobre un modelo orientado a eventos, o un modelo mixto que conjugue aplicaciones gobernadas por tiempo y eventos.

- Asimismo, 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. También 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.

APÉNDICES

APÉNDICE A

Presupuesto del Proyecto

En este apéndice se presentan justificados los costes globales de la realización de este Proyecto Fin de Carrera. Tales costes, imputables a gastos de personal y de material, se pueden deducir de las Tablas A.1, A.2 y A.3.

En la Tabla A.1 se muestran las fases del proyecto y el tiempo aproximado para cada una de ellas. Así pues, se desprende que el tiempo total dedicado por el proyectando ha sido de 1.200 horas, de las cuales aproximadamente un 20 % han sido compartidas con el tutor del proyecto, por lo que el total asciende a 1.440 horas. Teniendo en cuenta que la tabla de honorarios del Colegio Oficial de Ingenieros de Telecomunicación establece unas tarifas de 60 €/hora(teniendo en cuenta que son horas dentro de la jornada laboral, tal y como viene en el punto 19 de la baremación hecha por el COIT) el coste de personal, teniendo en cuenta el trabajo de un Ingeniero de Telecomunicación y un Jefe de Proyecto, se sitúa en 91.800 €. El coste de personal aparece desglosado en la tabla A.2.

En la Tabla A.3 se recogen los costes de material desglosados en 3 equipos informáticos y 3 redes de ordenadores. Ascienden a un total de 2.919,93€.

A partir de estos datos, el presupuesto total es el mostrado en la Tabla A.4.

Tabla A.1: *Fases del Proyecto*

Fase 1	*Documentación*	320 *horas*
Fase 2	*Implementación de los métodos propuestos*	560 *horas*
Fase 3	*Realización de las pruebas*	90 *horas*
Fase 4	*Redacción de la memoria del proyecto*	230 *horas*

Tabla A.2: *Costes de personal*

Personal	**Horas de trabajo**	**€/hora**	**Total (€)**
1 Jefe de Proyecto	220	90	19.800
1 Ing. Telecomunicación	1.200	60	72.000
			91.800

Tabla A.3: *Costes del material*

Equipo	**Precio**	**Total (€)**
3 PC gama alta	890 €	2.670
3 redes de ordenadores		
3 Switch	72,51€	217,53
30m. cable eth.	1,08€/m	32,4
		2.919,93 €

Tabla A.4: *Presupuesto*

Concepto	**Importe**
Costes personal	91.800€
Costes material	2.919,93€
I.V.A. (16 %)	15.155,19€
TOTAL	**109.875,12 €**

APÉNDICE B

Fichero '.m'

En este apendice incluiremos el fichero que servirá como ejemplo para que el usuario sea capaz de implementar sus propias aplicaciones para poder evaluarlas en los sistemas implementados. Este fichero corresponderá a la aplicación puesta como ejemplo en el capitulo 5, en el apartado dedicado al fichero de *Matlab* (apartado 5.3) y cuya representacion aparece en la figura 5.40. El nombre del fichero es *fichero_3_servicios_con_paralelo*.

A continuación se muestra el código implementado:

```
% Arquitectura con 3 nodos en serie y 1 en paralelo

% Creación de las variables globales de los diferentes deadlines, así como

% la variable que define el número de nodos en serie.

global deadline_aplicacion;

global deadline_nodo;

global deadline_red;

global nodosSerie;

% Creación de las varibales globales que definen la arquitectura de la

% aplicación, y sus operaciones sobre los nodos y sobre las redes.

global Arquitectura;

global Operaciones;

global Op_redes;
```

```
% Creación de los servicios usados por la aplicación

global A;

global B;

global C;

global D;

% Creación de las redes usadas por la aplicación

global z;

global y;

global w;

% Definición de la Arquitectura de la aplicación

 Arquitectura =  [  21   31   99   0   0;
                    0    31   0    0   0;
                    0    0    0    0   0   ];

% Definición de los Servicios que implementa cada nodo de la aplicación.

 Operaciones=  [  'A'  'B'  'C'  0   0;
                  0    'D'  0    0   0;
                  0    0    0    0   0   ];

% Definición de las redes sobre las cuales manda el mensaje cada nodo.

 Op_redes =  [  'z'  'y'  0   0   0;
                0    'w'  0   0   0;
                0    0    0   0   0   ];

% Definición del número de nodos en serie que tiene la aplicación.

nodosSerie=1;

while (Arquitectura(1,nodosSerie) = 99)

        nodosSerie=nodosSerie+1;

end

% Definición de los deadlines del sistema. Se definen los deadlines de los

% nodos, de las redes y el deadline del conjunto de la aplicación.

deadline_nodo = round(20*(rand+1));% Deadlines entre 20 y 40

deadline_red=round(50*(rand)+nodosSerie*40);% Deadlines entre 50 y 90
```

```
deadline_aplicacion = round(200*(rand)+nodosSerie*40);

% Definición de los perfiles posibles en cada Servicio

 A =  [  1  2
         4  3
         2  1
         1  1  ];


 B =  [  1  3
         3  2
         5  1
         2  3  ];


 C =  [  2  2
         3  1
         3  3
         3  2  ];

 D =  [  1  1
         2  2
         4  3
         3  2  ];

% Definición de los redes asociadas a la aplicación y la red física sobre

% la que se ejecutan

 z =  [  [round(4*(rand+0.25))  1  ];


 y =  [  [round(4*(rand+0.25))  2  ];


 w =  [  [round(4*(rand+0.25))  3  ]
```

Bibliografía

[1] Iria Estévez-Ayres. *Técnicas de soporte a la flexibilidad funcional en sistemas embarcados distribuidos de tiempo real.* PhD thesis, Departamento de Ingeniería Telemática. Universidad Carlos III de Madrid, Septiembre 2007.

[2] John A. Stankovic. Misconceptions about real–time computing: A serious problem for next generation systems. *IEEE Computer*, 21(10):10–19, 1988.

[3] Alan Burns, Neil Hayes, and M.F. Richardson. Generating feasible cyclic schedules. *Control Engineering and Practice*, 3(2):151–162, 1995.

[4] 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.

[5] Paulo Veríssimo and Luís Rodrigues. *Distributed systems for system architects.* Kluwer Academic Publishers, Norwell, MA, USA, 2001.

[6] Steve Jones. Toward an acceptable definition of service. *IEEE Software*, 22(3):87–93, 2005.

[7] 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.

[8] Mark Weiser. The Computer for the 21st Century. *Scientific American*, September 1991.

[9] 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.

[10] Hermann Kopetz. *Real–Time Systems: Design Principles for Distributed Embedded Applications.* Kluwer Academic Publishers, 1997.

[11] 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.

[12] 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/.

[13] A. Tešanović. *Developing Reusable and Reconfigurable Real–Time Software Using Aspects and Components.* PhD thesis, Linköping University, 2006.

[14] 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.

[15] 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.

[16] OMG. Real Time Corba Specification Version 1.2. formal/05-01-04, 2005.

[17] OMG. Corba Component Model. CCM v.4.0, 2006.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Alan Burns and Andy Wellings. *Real–Time Systems and their programming languages.* Addison Wesley, second edition, 1996.

[23] Kang G. Shin and P. Ramanathan. Real–time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE*, 82(1), 1994.

[24] N. Audsley. Deadline Monotonic Scheduling. Technical Report YCS_146, Department of Computer Science, University of York, Septiembre 1991.

[25] 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.

[26] 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.

[27] Keneth William Tindell. *Fixed Priority Scheduling of Hard Real–Time Systems.* PhD thesis, University of York, 1993.

[28] R. Garey and S. Johnson. Complexity results for multiprocessor scheduling under resource constraints. *SIAM Journal of Computing*, 4(4):397–411, 1975.

[29] Jose Carlos Palencia. *Análisis de planificabilidad de sistemas distribuidos de tiempo real.* PhD thesis, Grupo de Computadores y Tiempo Real. Universidad de Cantabria, 1999.

[30] J. Zamorano-Flores. *Planificación estática de procesos en sistemas de tiempo real críticos.* PhD thesis, Universidad Politécnica de Madrid, 1995.

[31] H. S. Wilf. *Algorithms and Complexity.* Prentice Hall, 1986.

[32] C. Y. Castañeda Roldán. *Estudio comparativo de diversos métodos de solución del problema del agente viajero (PAV).* PhD thesis, Departamento de Ingeniería en Sistemas Computacionales, Escuela de Ingeniería, Universidad de las Américas Puebla, 2000.

[33] Narsingh Deo Edward M. Reingold, Jurg Nievergelt. *Combinatorial Algorithms Theory and Practice.* Prentice-Hall, 1977.

[34] D. R. Stinson. *An Introduction to the Design and Analysis of Algorithms.* Charles Babbage Research Centre, 2nd ed. rev. edition, 1987.

[35] 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.

[36] C. Locke. Software architecture for hard real–time applications: Cyclic executives versus fixed priority executives. *Real–Time Systems*, 4(1):37–53, 1992.

[37] C. L. Liu and James W. Layland. Scheduling algorithms for multiprogramming in a hard–real–time environment. *J. ACM*, 20(1):46–61, 1973.

[38] Paulo Pedreiras. *Supporting Flexible Real-Time Communication on Distributed Systems.* PhD thesis, Departamento de Electrónica e Telecomunicaões of the University of Aveiro, Aveiro, Portugal, 2003.

[39] 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.

[40] J. Leung and J. Withehead. On the complexity of fixed priority scheduling of periodic real–time tasks. *Performance Evaluation*, 2(4), 1982.

[41] M. Joseph and P. Pandya. Finding Response Times in a Real–Time System. *BCS Computer Journal*, 29(5):390–395, Octubre 1986.

[42] P. K. Harter. Response times in level structured systems. Technical Report CU–CS–269–94, Department of computer Science. University of Colorado, USA, 1984.

[43] P. K. Harter. Response times in level structured systems. *ACM Transactions on Computer Systems*, 5:232–248, 1997.

[44] 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.

[45] 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.

[46] M. L. Dertouzos. Control robotics: the procedural control of physical processses. *Information Processing*, page 74, 1974.

[47] 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.

[48] 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.

[49] Marco Spuri and Giorgio C. Buttazzo. Efficient aperiodic service under the earliest deadline scheduling. In *Proc. of the IEEE Real–Time Systems Symposium*, 1994.

[50] Marco Spuri and Giorgio C. Buttazzo. Scheduling Aperiodic Tasks in Dynamic Priority Systems. *Real–Time Systems*, 10(2):179–210, 1996.

[51] L. Abeni and G. Buttazzo. Resource reservation in dynamic real–time systems. *Real–Time Systems*, 27(2):123–165, 2004.

[52] 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.

[53] 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.

[54] 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.

[55] Giorgio Buttazzo and L. Abeni. Adaptive workload management through elastic scheduling. *Real Time–Systems*, 23(1–2):7–24, 2002.

[56] 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.

[57] 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.

[58] I. Ripoll. *Planificación con Prioridades Dinámicas en Sistemas de Tiempo Real Crítico*. PhD thesis, Valencia, 1996.

[59] 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.

[60] 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.

[61] 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.

[62] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessing and Microprogramming*, 50(2–3), 1994.

[63] Ken Tindell. An extendible approach for analysis fixed priority hard real-time tasks. *Real–Time Systems Journal*, 6(2), Marzo 1993.

[64] 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.

[65] J. Oscar Rojo. *Introducción a los Sistemas Distribuidos*, 2003. Disponible a 20 de Enero de 2010 en http://www.augcyl.org/?q=glol-intro-sistemas-distribuidos.

[66] IEC ISO and ITU. The Reference Model of Open Distributed Processing (RM-ODP). Disponible a 15 de Enero de 2010 en http://www.rm-odp.net/.

[67] International Organization for Standardization. International Organization for Standardization Web Page. Disponible en http://www.iso.org/iso/home.htm on.

[68] R. Garey and D. Johnson. Complexity Bounds for Multiprocessor Scheduling with Resource Constraints. *SIAM Journal of Computing*, 4(3):187–200, 1975.

[69] John A. Stankovic et al. Implications of Classical Scheduling Results for Real–Time Systems. *IEEE Computer*, 28(6), 1995.

[70] 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.

[71] 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.

[72] 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.

[73] Ken Tindell. Adding Time–Offsets to Schedulability Analysis. Technical Report YCS221, Department of Computer Science, University of York, England, 1994.

[74] 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.

[75] J. Sun and J. Liu. Synchronization protocols in distributed real-time systems. In *Proc. of the 16th International Conference on Distributed Systems*, Mayo 1996.

[76] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services. Concepts, Architectures and Applications.* Springer, 2004.

[77] World Wide Web Consortium. *Web Services Glossary*, February 2004. W3C Working Group Note.

[78] M. Satyanarayanan. Pervasive computing: vision and challenges. *IEEE Personal Communications*, 8(4):10–17, August 2001.

[79] M. H. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, January/February 2005.

[80] Mario Calha. *A Holistic Approach Towards Flexible Distributed Systems*. PhD thesis, DET / IEETA-LSE. Universidade de Aveiro, 2006.

[81] Robert Davis and Andy J. Wellings. Dual Priority Scheduling. In *IEEE Real-Time Systems Symposium*, pages 100–109, 1995.

[82] C.R.Reeves. Modern heuristic techniques for combinatorial problems. *Blackwell Scientific Press, Oxford*, 1993.

[83] Wikibooks contributors. *LaTeX*. Wikibooks, 2005. Disponible a 10 de Diciembre de 2009 en http://upload.wikimedia.org/wikibooks/en/2/2d/LaTeX.pdf.

[84] Michel Goossens Frank Mittelbach. *The LaTeX Companion (Tools and Techniques for Computer Typesetting)*. Addison–Wesley, 2nd edition, 2004.

[85] Leslie Lamport. *Latex: A Document Preparation System*. Addison–Wesley, 2nd edition, 1994.

[86] Nikos Drakos. *Manual de LaTeX*. Computer Based Learning Unit, University of Leeds, 1993. Disponible a 7 de Enero de 2010 en http://www.fceia.unr.edu.ar/lcc/cdrom/Instalaciones/LaTex/latex.html.

[87] Carlos Ivorra. *Preparación de textos con LATEX*. Departamento de Matemáticas para la Economía y la Empresa, Universitat de València. Disponible a 8 de Enero de 2010 en http://www.uv.es/ivorra/Latex/LaTeX.pdf.

[88] Hugo S. Salinas. *Apuntes de Latex*. Departamento de Matemática, Facultad de Ingeniería, Universidad de Atacama, CHILE. Disponible a 30 de Enero de 2010 en http://www.mat.uda.cl/hsalinas/latex.htm.