ESCUELA POLITÉCNICA SUPERIOR

UNIVERSIDAD CARLOS III DE MADRID

INGENIERÍA TÉCNICA DE TELECOMUNICACIONES:

SISTEMAS DE COMUNICACIONES

PROYECTO FIN DE CARRERA

IMPLEMENTACIÓN Y EVALUACIÓN DE ALGORITMOS DE COMPOSICIÓN Y RECONFIGURACIÓN DE APLICACIONES DISTRIBUIDAS DE TIEMPO REAL BASADAS EN SERVICIOS

Autor: **MARÍA RODRÍGUEZ ALIQUE**

Tutor: **DRA. IRIA MANUELA ESTÉVEZ AYRES**

OCTUBRE 2011

Vivir no es sólo existir,
sino existir y crear,
saber gozar y sufrir
y no dormir sin soñar.

Gregorio Marañón

Agradecimientos

Me gustaría mostrar mi más profundo agradecimiento a todas las personas que siempre me han mostrado su apoyo, y me han brindado la oportunidad de desarrollarme tanto intelectual como personalmente.

En primer lugar, agradecer el aguante y la perseverancia que ha tenido Iria, mi directora del proyecto, conmigo. Desde que comencé este proyecto, ella ha sufrido conmigo todos los altibajos por los que he pasado durante este período de tiempo. Siempre he tenido su apoyo, y gracias a él, ahora puedo decir que he cumplido mi sueño.

Además, agradecer a mis padres el enorme esfuerzo que han tenido que realizar a lo largo de su vida para darme la formación académica que ellos nunca pudieron tener. Por los buenos y malos momentos, por aguantar suspensos y aprobados, llantos y risas, y sobre todo, por apoyar todas las decisiones tomadas a lo largo de mi formación. Por todo ello, gracias.

A mis hermanos, por ayudarme a crecer como persona, por su apoyo incondicional ante cualquier circunstancia, y como no, por entenderme y estar siempre a mi lado.

A mis amigos y compañeros por facilitarme el día a día, por animarme en los malos momentos y por aguantarme en los peores, y sobre todo, por apoyarme en mis proyectos de futuro. En definitiva, por estar siempre ahí.

Por último, y no por ello menos importante, sino todo lo contrario, a Javier, sin tí esto no habría sido realidad, tu apoyo constante ha conseguido que no me hundiera en los peores momentos, y que siempre sacara fuerzas para continuar. Gracias por aguantar los momentos de nervios y de estrés.

A todo el mundo que ha contribuido a hacer realidad este sueño.

Gracias.

Resumen

Actualmente cada vez son más numerosos los sistemas con requisitos temporales presentes en numerosos entornos. Esto implica que el dominio de aplicación de sistemas con requisitos temporales se ha ampliado, incluyendo dominios de aplicación que, además, requieren que los sistemas en ellos desplegados sean lo suficientemente flexibles y dinámicos como para adaptarse a entornos cambiantes. Por ello, se han buscado sinergias con otros campos de investigación, como la orientación a servicios.

Los sistemas distribuidos de tiempo real basados en servicios pueden aportar dicha flexibilidad y dinamismo. Una aplicación basada en servicios se definirá a través de las interacciones entre las distintas funcionalidades (servicios) que la integran. A su vez, una funcionalidad (servicio) podrá ser implementado por distintas versiones de código (implementaciones) en distintos (o el mismo) nodo físico. La composición es el proceso de elegir el conjunto de implementaciones más adecuado que compondrán una aplicación, en un momento determinado y teniendo en cuenta el estado del sistema y los requisitos impuestos por el usuario. Por otra parte, cuando alguna implementación deja de estar disponible o falla, o cuando el estado del sistema cambia de tal forma que ya no es posible atender a los requisitos del usuario, es necesaria la reconfiguración del mismo.

En el presente proyecto, el objetivo principal es desarrollar un procedimiento para la reconfiguración de aplicaciones de tiempo real basadas en servicios teniendo en cuenta tanto la estructura de la aplicación como el estado actual del sistema. Para ello, basándonos en un algoritmo de reconfiguración previamente desarrollado, se han propuesto mejoras a dicho algoritmo, se ha implementado en *Matlab* y se ha analizado su comportamiento en diversas situaciones con distintas aplicaciones, llegando a una herramienta analítica que permite discernir cuándo aplicar cada algoritmo de reconfiguración.

Para conseguir dicho objetivo principal se han implementado tanto algoritmos de composición como de reconfiguración en *Matlab*. También en *Matlab*, se ha desarrollado una herramienta que permite la simulación de los distintos algoritmos, y el análisis de su comportamiento.

Índice General

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este capítulo se describe de forma general el contexto del presente proyecto fin de carrera. En primer lugar, se plantean las motivaciones que han dado lugar a la elaboración del proyecto, en segundo se enumeran los objetivos del mismo, y por último, se presenta la organización del resto de capítulos de este documento.

1.1. Visión general

Los sistemas de tiempo real (STR) son sistemas informáticos que se encuentran en multitud de aplicaciones, desde la electrónica de consumo hasta el control de complejos procesos industriales. Están presentes en prácticamente todos los aspectos de nuestra sociedad como, teléfonos móviles, automóviles, control de tráfico, ingenios espaciales, procesos automáticos de fabricación, producción de energía, aeronaves, etc. Además, el auge de los Sistemas de Tiempo Real está en constante aumento, ya que cada vez más máquinas se fabrican incluyendo un número mayor de sistemas controlados por computador. Un ejemplo cercano es la industria del automóvil, ya que un turismo actual de gama media incluye alrededor de una docena de estos automatismos (ABS, airbag, etc). Otro ejemplo cotidiano son los electrodomésticos de nueva generación, que incluyen Sistemas de Tiempo Real para su control y temporización. Hoy día son tantas las aplicaciones de estos sistemas que su número duplica actualmente al de los sistemas informáticos "convencionales" o de propósito general. Las previsiones son que esta

diferencia vaya en constante aumento, debido fundamentalmente al elevado crecimiento de la automatización en casi todas las facetas de la vida cotidiana [27] .

La característica diferenciadora de los Sistemas de Tiempo Real es que sus acciones deben producirse dentro de unos intervalos de tiempo determinados por la dinámica del sistema físico que supervisan o controlan. Por poner un ejemplo, el sistema de control de inyección de combustible en un motor alternativo (como los que están presentes en los automóviles) debe realizar la inyección de la mezcla dentro del intervalo de tiempo marcado por la rotación del motor, de otro modo el motor no funcionará correctamente. En este caso, se trata de un sistema de tiempo real empotrado, es decir, el sistema informático se encuentra físicamente incluido en un sistema de ingeniería más complejo. La mayoría de los sistemas de tiempo real son sistemas empotrados y suelen tener restricciones adicionales en cuanto al uso de recursos computacionales con respecto a otros tipos de sistemas informáticos. Además, suelen tener requisitos de seguridad y fiabilidad más severos, ya que si el sistema falla puede ocasionar pérdidas económicas (por ejemplo, avería del motor) o incluso humanas (por ejemplo, si el motor fuera de una aeronave).

Para adaptarse a este nuevo entorno, se está empezando a usar la utilización del paradigma de computación orientada a servicios (*Service Oriented Computing, SOC*)[25]. Este paradigma está basado en un modelo de computación distribuida fundamentado en la existencia de proveedores y consumidores de servicios, y registros para la publicación y búsqueda de los mismos. Entendiendo como servicio "un dominio de control de una envergadura acotada que contiene un conjunto de tareas que cooperan para alcanzar objetivos relacionados". A partir de ahora, nos referiremos a servicio como a una entidad software autocontenida que proporciona una determinada funcionalidad.

El ámbito de aplicación de la SOC son tanto entornos internos a una organización como entornos de computación colaborativa entre distintas organizaciones. Las principales características que debe cumplir una arquitectura que dé soporte a este paradigma son, según se expone en [25]:

- Acoplamiento débil entre proveedores y consumidores de servicios.
- Independencia con respecto a la implementación (lenguajes de programación, bibliotecas de código, plataformas de ejecución, etc.)
- Configuración flexible y dinámica.

- Robustez y fiabilidad.
- Modelado de interacciones a un nivel de abstracción alto.
- Colaboración.

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

Así, se puede observar que la utilización de la composición de servicios además de permitir decrementar el tiempo de desarrollo de un proyecto, a través del uso de piezas ya existentes para crear una mayor, permite una sencilla gestión de la flexibilidad de los servicios seleccionados, pudiendo elegir de entre un conjunto de proveedores aquél que, implementando la misma funcionalidad, sea el que más se ajuste a nuestras necesidades actuales, seleccionando otro proveedor cuando éstas cambien (selección dinámica de servicios). Esta flexibilidad puede ser utilizada para proporcionar tolerancia a fallos a nivel de aplicación, ya que servicios que ya no funcionan pueden ser cambiados por otros y, además, este tipo de arquitecturas suele dar facilidades [25] para poder capturar y manipular el estado de una transacción. Al permitir agregar recursos, también puede ser usado para superar las restricciones impuestas por las características físicas de determinados dispositivos.

1.2. Motivación

Muchos de los sistemas de tiempo real actuales son de planificación estática, es decir, están formados por un conjunto de servicios fijos, previamente seleccionados, que son asignados a los recursos disponibles y, para asegurar que todas las restricciones temporales son satisfechas, se realizan comprobaciones previas al tiempo de ejecución. En

contraste, los sistemas de tiempo real adaptativos del futuro necesitarán más flexibilidad durante la fase de ejecución, ajustando el funcionamiento del sistema a nueva información dinámica que procederá tanto del entorno en el que se ejecuta como del propio sistema. Bajo estas circunstancias, las demandas al sistema cambiarán dinámicamente y estos sistemas adaptativos deberán poder modificar el conjunto de servicios que soportan y extender sus funciones para poder satisfacerlas.

Como ya se comentó en el apartado anterior, el paradigma de computación orientada a servicios puede aportar esta flexibilidad a los sistemas, permitiendo la creación dinámica de aplicaciones a partir de servicios existentes remotos, pudiendo ser usada ventajosamente en el desarrollo de sistemas de tiempo real. Aplicándose en entornos tan dispares como sistemas de control de red, donde diferentes sensores y filtros pueden ser vistos como servicios que pueden ser compartidos y actualizados *on-line*, o sistemas multimedia distribuidos en los que los servicios pueden corresponder a filtros y codificadores/decodificadores que ofrezcan diferentes calidades de servicios. Así, la composición dinámica de servicios puede ser interesante no sólo desde el punto de vista del desarrollo de aplicaciones, a través de la composición automática de elementos *software*, sino también como un medio para dar soporte a la actualización dinámica y reconfiguración de servicios, por ejemplo para la gestión dinámica de *QoS* (calidad de servicio), equilibrado de carga o tolerancia a fallos.

Cuando Iria Estévez Ayres comenzó su tesis [1] todavía no se habían definido arquitecturas con características de tiempo real flexibles en las cuales se aplicaran de manera ventajosa conceptos del campo de la orientación a servicios, por lo que ante la idea de justificar la necesidad de la definición de métodos y metodologías, así como de una arquitectura de tiempo real genérica que aplicara este paradigma, surgió dicha tesis. En ella se realizaron aportaciones en el ámbito de sistemas de tiempo real distribuidos para flexibilizarlos con conceptos y aportaciones técnicas inspirados en la computación orientada a servicios.

1.3. Objetivos

Apoyándonos tanto en los estudios realizados por Iria Estévez Ayres [1], como en los realizados por Jorge Díez Sánchez [2], se fijaron los objetivos de este proyecto fin de carrera.

1. Estudio de los conceptos y propiedades de los sistemas distribuidos de tiempo real.

2. Estudio de algoritmos de composición de aplicaciones distribuidas de tiempo real basadas en servicios de [1] y [2].

3. Implementación y mejora de nuevos algoritmos de composición para aplicaciones distribuidas de tiempo real basadas en servicios. Éstos deberán proporcionar soluciones a la composición en un tiempo acotado, o al menos obtener cotas máximas de su tiempo de ejecución para un espacio acotado de búsqueda, de tal forma que sea posible su utilización en tiempo de ejecución en un entorno con restricciones temporales.

4. Validación de los algoritmos de composición implementados mediante simulación sobre una herramienta creada para el efecto. Así como la validación de la eficiencia de las figura de mérito relativas implementadas (*Véase Apartado 4.5.2*).

5. Implementación de algoritmos de reconfiguración de aplicaciones distribuidas de tiempo real basadas en servicios. Éstos deberán proporcionar la posibilidad de reconfigurar aplicaciones de sistemas de tiempo real en caso de que algún perfil de servicio (*Véase Apartado 3.3.1*) que esté siendo utilizado deje de estar disponible. Estos algoritmos deben proporcionar soluciones a la composición en un período limitado de tiempo, de tal forma que sea posible su uso *on-line* por aplicaciones de sistemas de tiempo real con restricciones temporales.

6. Validación de los algoritmos de reconfiguración implementados:

    a. Simulación sobre la herramienta el correcto funcionamiento de los algoritmos de reconfiguración actuando sobre diferentes aplicaciones.

    b. Procesamiento y clasificación de los resultados generados.

7. Propuesta de una herramienta analítica que permite realizar la reconfiguración de aplicaciones *on-line*, eligiendo además, en función de los parámetros de la aplicación, qué algoritmo de entre los implementados en el proyecto es el más eficiente para realizar la reconfiguración.

Resaltar que el desarrollo de todo el proyecto se ha realizado sobre el programa de software matemático *MATLAB*. Éste ofrece un entorno de desarrollo integrado con un lenguaje de programación propio (*lenguaje M*) [19]. Se ha elegido este programa por la

facilidad que presenta a la hora de operar con matrices y la claridad en la representación de datos y funciones, además de ser el programa que se utilizó para la implementación de los algoritmos de composición de [2].

1.4. Organización de la memoria

La memoria está distribuida en 8 capítulos:

Capítulo 1 (Introducción): en este capítulo se realiza una pequeña introducción acerca de la importancia de los sistemas de tiempo real. Además se explican las motivaciones y los objetivos de este proyecto. Así como la organización de la memoria.

Capítulo 2 (Estado del Arte): en este capítulo se presenta, de manera teórica, una visión general de los sistemas de tiempo real. Se hablará, de manera introductoria, acerca de la planificación de estos sistemas, así como la clasificación de los algoritmos en función de su dinamismo. Además se hará una breve descripción de los sistemas distribuidos.

Capítulo 3 (Modelos de Diseño): en este capítulo se introduce el modelo de sistema en el que se basó el trabajo desarrollado en este proyecto y el modelo de aplicación de tiempo real basada en servicios. Además, se presenta y caracteriza funcional y temporalmente el concepto de perfil de servicio (o implementación de servicio) propuesto en [1].

Capítulo 4 (Implementación de Algoritmos de Composición): en este capítulo se explican los algoritmos de composición implementados, así como una explicación de las figuras de mérito utilizadas. Se realiza una explicación de las funciones de las que se compone cada uno de los algoritmos. Además se explican las decisiones de diseño consideradas y el prototipo de diseño seguido para la creación de la herramienta de simulación.

Capítulo 5 (Implementación de Algoritmos de Reconfiguración): en este capítulo se presentan los dos algoritmos de reconfiguración implementados, presentando una explicación detallada de cada uno de ellos Además, se realiza una breve descripción de las decisiones de diseño consideradas.

Capítulo 6 (Evaluación y Pruebas de los Algoritmos de Composición): en este capítulo se presentan la evaluación y pruebas de los algoritmos de composición. Se validará el funcionamiento de los algoritmos de composición a través de la herramienta

diseñada. Se definirán pruebas con diferentes aplicaciones para la evaluación de los algoritmos, y se registrarán los resultados, presentándolos tanto en tablas como en gráficas.

Capítulo 7 (Evaluación y Pruebas de los Algoritmos de Reconfiguración): en este capítulo se presentan la evaluación y pruebas de los algoritmos de reconfiguración. Se validará el funcionamiento de los algoritmos de reconfiguración a través de la herramienta diseñada. Se definirán pruebas con diferentes aplicaciones para la evaluación de los algoritmos. Se registrarán los resultados, presentándolos tanto en tablas como en gráficas. Se procesarán dichos resultados, concluyendo en una serie de normas (o reglas) que permitan predecir qué algoritmo es más eficiente en función de la estructura de la aplicación que se quiera reconfigurar. Se realizará una herramienta analítica que permita, de forma transparente para el usuario, la reconfiguración de una aplicación en tiempos de ejecución utilizando el algoritmo más conveniente.

Capítulo 8 (Conclusiones y Líneas Futuras): en este capítulo se presentan a grandes rasgos las conclusiones generales extraídas a lo largo de la elaboración del proyecto, exponiéndose las principales contribuciones realizadas y las posibles sugerencias de líneas de trabajo futuras.

Capítulo 2

Estado del Arte

En este capítulo se realiza un resumen de los conceptos básicos de los Sistemas de Tiempo Real (STR), los cuales están ampliamente relacionados con este proyecto. Se incluyen definiciones, características, y clasificación de los mismos. En los últimos apartados se detalla, de una manera más exhaustiva, la planificabilidad de los Sistemas de Tiempo Real, ya que es la parte más ligada al ámbito de este proyecto.

2.1. Sistemas de Tiempo Real

Se entiende por sistema de tiempo real aquel en el que para que las operaciones computacionales sean correctas no sólo es necesario que la lógica e implementación de los programas computacionales sea correcto, sino también el tiempo en el que dicha operación entregó su resultado. Si las restricciones de tiempo no son respetadas el sistema se dice que ha fallado [4]. Por lo tanto, es esencial que las restricciones de tiempo se cumplan, es decir, que el sistema sea predecible. Por otra parte, en determinados entornos, es también deseable que el sistema obtenga un alto grado de utilización a la vez que cumple con los requerimientos de tiempo.

Las características deseables para un sistema de tiempo real son:

- **Puntualidad**. Los resultados deben ser correctos tanto en su valor como en el tiempo en el cual se producen.

- **Alta utilización de recursos**. Poder alcanzar utilizaciones altas de los recursos, en especial de la CPU, a la vez que se garantiza el cumplimiento de los plazos.

- **Diseñados para soportar picos de carga**. Los sistemas de tiempo real deben ser diseñados para que soporten picos de carga, es decir los sistemas de tiempo real no deben colapsar cuando están sujetos a condiciones de pico de carga y deben poder manejar todos los escenarios anticipadamente.

- **Fiabilidad**. Los sistemas de tiempo real deben proporcionar el servicio especificado.

- **Previsibilidad**. El sistema debe garantizar un mínimo nivel de rendimiento y ser capaz de predecir las consecuencias de alguna decisión de planificación. El sistema debe ser capaz de predecir la evolución de las tareas y garantizar anticipadamente que todas las restricciones de tiempo crítico sean cumplidas. Sin embargo esto depende de varios factores, los cuales tienen que ver con las características de la arquitectura de hardware (DMA, memoria cache, etc.), los mecanismos y políticas adoptadas por el kernel (semáforos, llamadas del sistema, interrupciones, etc.) y el lenguaje de programación utilizado en la implementación.

- **Tolerante a fallos**. Simples fallos en el hardware o en el software no deben causar que el sistema se colapse. Así, los componentes críticos de los sistemas de tiempo real tienen que ser diseñados para tolerar fallos.

- **Producir un bajo aumento de la sobrecarga** (overhead), es decir, un bajo aumento del trabajo extra que la CPU dedica para realizar la planificación.

- **Mantenibilidad**. La arquitectura del sistema de tiempo real debe ser modular para asegurar que posibles modificaciones del sistema sean realizadas fácilmente.

2.1.1. Tareas de un Sistema de Tiempo Real

Una de las más importantes entidades de software tratada por un sistema operativo es el proceso. Un proceso es un cálculo que es realizado por la CPU de forma secuencial. Una *tarea* es una ejecución secuencial de código que no se suspende así misma durante su ejecución, un *proceso* es una actividad computacional más compleja, que puede estar compuesta por varias tareas.

Los sistemas de tiempo real se caracterizan por tener actividades computacionales con estrictas restricciones que deben ser cumplidas en orden para alcanzar el comportamiento deseado. Una típica restricción tiempo sobre una tarea es el plazo *(deadline)*, el cual representa el tiempo antes del cual un proceso debe completar su ejecución. Dependiendo de las consecuencias de la pérdida de un plazo, las tareas de tiempo real son clasificadas en:

- **Críticas**. Una tarea se dice que es crítica si la pérdida de su plazo puede causar catastróficas consecuencias sobre el sistema. Sistemas de tiempo real críticos son sistemas capaces de manejar tareas de tiempo real críticas, por ejemplo sistemas de control de vuelo, sistemas de control de procesos químicos o ambientes críticos en tiempo tal como sistemas de control de robots y sistemas de conmutación telefónica.

- **Acríticas**. Una tarea se dice que es acrítica si la pérdida de su plazo decrementa el rendimiento del sistema pero no pone en peligro su correcto comportamiento. Sistemas de tiempo real acríticos son sistemas capaces de manejar tareas de tiempo real acríticas, por ejemplo, sistemas de adquisición de datos remotos.

A la hora de implementar un sistema, hay que conocer las características y exigencias de cada una de las tareas, por eso a continuación se muestran los principales parámetros por los que se caracteriza una tarea:

- **Tiempo de llegada** (a_i): tiempo en el cual la tarea esta lista para su ejecución, también es referido como el tiempo de liberación o tiempo de solicitud indicado por r_i.

- **Tiempo de computación en el peor caso** - *Worst Case Execution Time (WCET)* (C_i): Tiempo necesario de procesador para ejecutar la tarea sin interrupción.

– **Plazo** – *Deadline* – (D_i): tiempo antes del cual una tarea deberá completarse.

– **Tiempo de comienzo** (S_i): tiempo en el cual la tarea comienza su ejecución.

– **Tiempo de finalización** (f_i): tiempo en el cual la tarea finaliza su ejecución.

– **Tiempo de respuesta** - *Worst Case Response Time (WCRT)* (R_i): tiempo de respuesta de la tarea en el peor caso.

– **Criticidad** (K_i): parámetro relacionado a las consecuencias de perder el plazo. Típicamente puede ser crítica o acrítica.

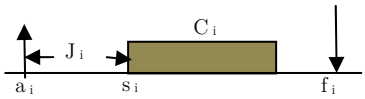
– **Prioridad** (p_i): importancia relativa de una tarea con respecto a otra en el sistema.

A cada tarea τ_i le es asignada una prioridad p_i que indica la importancia o urgencia que tiene τ_i con respecto a las otras tareas en el sistema. Las prioridades pueden ser asignadas a las tareas estáticamente o dinámicamente. Si en un tiempo t, $p_a > p_b$ significa que la ejecución de τ_a es más importante que la de τ_b; así, τ_b puede ser retardada a favor de τ_a.

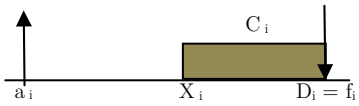
– **Retardo** (L_i): representa el retardo de una tarea completada con respecto a su plazo (*deadline*). $L_i = f_i - D_i$.

– **Tardanza o Tiempo excedente** (E_i): tiempo que una tarea activa después de su plazo. $E_i = max\ (0,\ L_i)$.

– **Variación** – *Jitter* (J_i): tiempo que una tarea gasta desde su liberación hasta su tiempo de comienzo. $J_i = a_i - s_i$.



– **Latencia o tiempo de holgura** (x_i): es el máximo tiempo que la activación de una tarea puede ser retardada tal que pueda completarse dentro de su plazo. $X_i = D_i - a_i - C_i$.



Otra característica importante que puede ser especificada sobre una tarea de tiempo real, se refiere a la regularidad de su activación. En particular las tareas pueden definirse en:

- **Tareas periódicas:** consisten de una secuencia infinita de actividades idénticas, llamadas instancias que son activadas regularmente a una razón constante. La activación de la primera instancia es llamada fase. Si ϕ_i es la fase de la tarea periódica τ_i, el tiempo de activación de la k-ésima instancia estará dado por la siguiente expresión: $\phi_i + (k\text{-}1)T_i$, donde T_i es el período de la tarea. La Figura 1, muestra un ejemplo de una secuencia de tareas periódicas.

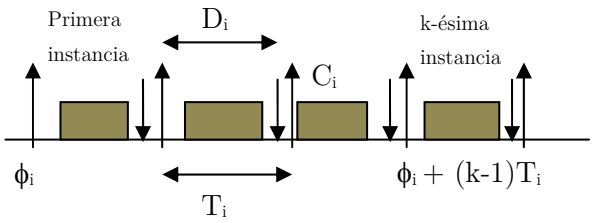


Figura 1. Secuencia de tareas periódicas

Así, considerando un conjunto de N tareas periódicas Π, éste se puede caracterizar como:

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

- **Tareas aperiódicas:** consisten de una secuencia infinita de instancias cuyas activaciones no son regulares. La Figura 2, muestra un ejemplo de una secuencia de tareas periódicas.

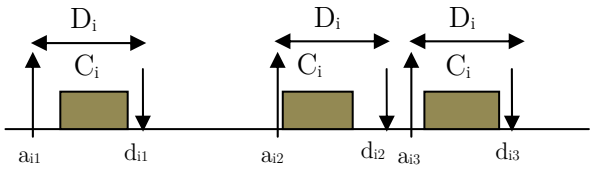


Figura 2. Secuencia de tareas aperiódicas

2.2. Planificación de Sistemas de Tiempo Real

Cuando un procesador tiene que ejecutar un conjunto de tareas concurrentes (que pueden solaparse en el tiempo), las tareas tienen que ser asignadas a la CPU de acuerdo a un criterio predefinido llamado política de planificación. El conjunto de reglas que en algún tiempo determina el orden en el cual las tareas son ejecutadas es llamado un *algoritmo de planificación.*

Siempre hay un método de análisis que permite comprobar el comportamiento temporal del sistema, y acompaña al algoritmo de planificación. En general éste estudia el peor comportamiento posible.

Una tarea que puede potencialmente ejecutarse sobre el procesador, independientemente de su disponibilidad, es llamada una tarea preparada o lista, cuando la tarea se está ejecutando es llamada una tarea en ejecución. Todas las tareas listas que esperan por el procesador son guardadas en una cola, llamada cola de preparados.

En muchos sistemas operativos que permiten activación de tareas dinámicas, las tareas en ejecución pueden ser interrumpidas en algún momento, tal que una tarea más importante que llegue al sistema pueda inmediatamente tomar el procesador y no necesite esperar en la cola de preparados. En este caso la tarea en ejecución es interrumpida e insertada en la cola de preparados, mientras que la CPU es asignada a la tarea lista más importante que acaba de llegar. La operación de suspensión de tareas que están ejecutándose y, de inserción de éstas en la cola de preparados, es llamada *desalojo* [6].

Un *planificador con desalojo* es un planificador en el cual las tareas que están ejecutándose pueden ser arbitrariamente interrumpidas en algún instante de tiempo, para asignar la CPU a otra tarea de acuerdo a una política de planificación predefinida.

Cuando una tarea está ejecutándose, ésta puede ser *suspendida* si la tarea requiere algún recurso que no esté disponible o simplemente porque la misma esté esperando a alguna señal del sistema. Todas las tareas suspendidas sobre un mismo recurso son almacenadas dentro de una cola asociada al recurso. La Figura 3, ilustra los estados y el proceso de planificación de las tareas.

Un planificador se dice que es *factible* si todas las tareas pueden ser completadas de acuerdo a un conjunto de restricciones específicas. Un conjunto de tareas se dice que es planificable si existe al menos un algoritmo que pueda producir un planificador factible.

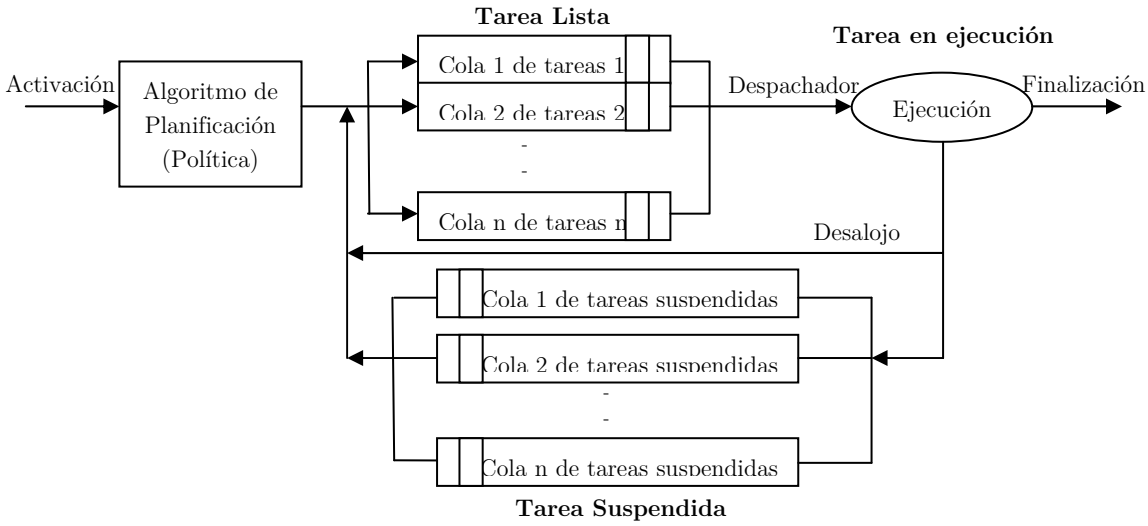


Figura 3. Estado y proceso de planificación de las tareas

Por lo tanto, el término de *planificabilidad* se referirá al proceso de seleccionar una tarea concreta para su ejecución en un determinado instante temporal. El análisis de planificabilidad para un determinado algoritmo se basa en tests. El resultado de éstos debe reflejar la capacidad de ese algoritmo para encontrar una planificación factible dado un conjunto de taras. Sin embargo, la precisión de estos test depende de su complejidad, siendo clasificados en exactos, suficientes y necesarios [10].

Los más complejos computacionalmente son los tests *exactos*, aquéllos que dan un resultado positivo siempre que exista una planificación posible para el sistema y negativo en caso contrario. Desgraciadamente, la mayor parte de este tipo de tests son computacionalmente intratables [23], o implican una carga computacional excesiva. Más sencillos de implementar son los tests denominados *suficientes*, aquéllos cuyo resultado positivo asegura la existencia de una planificación posible, pero si es negativo no aseguran que no la haya. Nótese que este tipo de tests puede, por lo tanto, rechazar conjuntos de tareas planificables. Los tests denominados *necesarios* sólo aseguran con un resultado negativo la imposibilidad de planificar el conjunto de tareas, mientras que si es positivo puede ocurrir que éstas no sean planificables.

Por otra parte, principalmente hay dos familias de tests aplicados a sistemas de tiempo real, basada cada una de ellas en conceptos diferentes [24]:

- **Límite de utilización**: entendiendo por utilización el porcentaje de ocupación del procesador. Cada tarea periódica ¿i carga el procesador con una utilización dada por:

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

- **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.

2.2.1. Algoritmos de planificación

Cada algoritmo presenta una solución para un problema de planificación particular, el cual es expresado a través de un conjunto de suposiciones sobre el conjunto de tareas y por el criterio de optimización que es usado por el planificador.

Existen principalmente dos paradigmas de algoritmos de planificación: *planificadores cíclicos* y *planificadores basados en prioridades*, tal y como se muestra en la Figura 4.

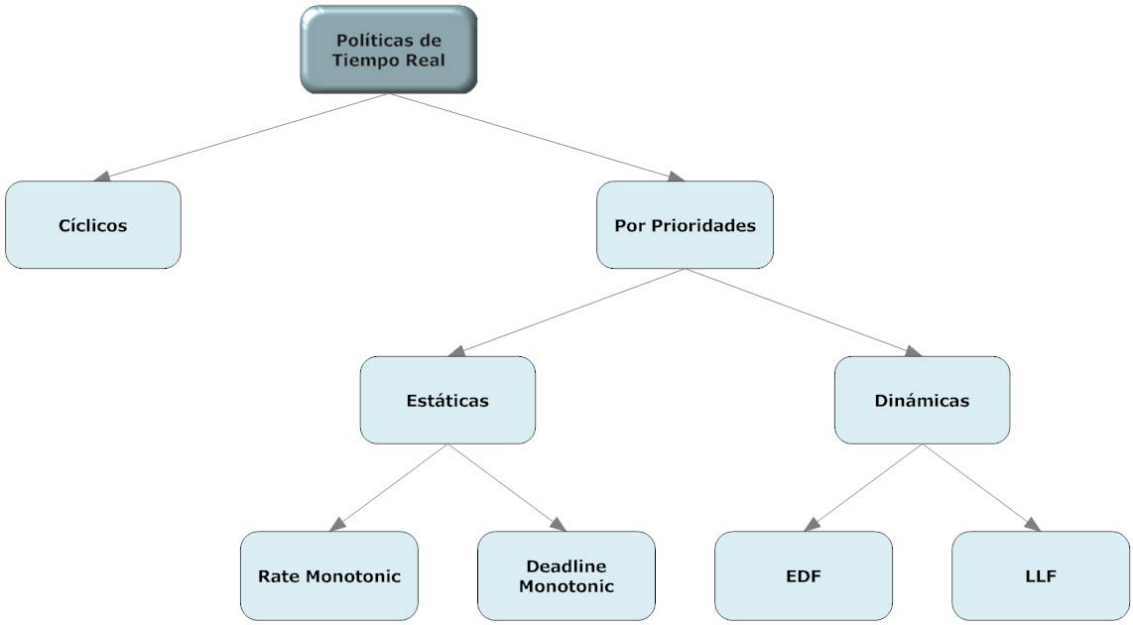


Figura 4. Algoritmos de planificación

El primero, es el que más se ha utilizado tradicionalmente, se conoce por el nombre de planificador cíclico y consiste en construir en la fase de diseño del sistema una tabla con todas las acciones que tenía que llevar a cabo el planificador durante la ejecución. Durante la ejecución, el planificador únicamente consulta la tabla y hace que el procesador ejecute las tareas siempre en el mismo orden. El principal problema que presentan este tipo de algoritmos es la poca flexibilidad a la hora de modificar alguno de

los parámetros de las tareas, pues ello conlleva la re-elaboración de todo el plan. A pesar de ello, aún hoy este tipo de planificación se utiliza en la industria.

Otro gran grupo de planificadores son los basados en prioridades, éstos se dividen en prioridades estáticas y prioridades dinámicas. Este grupo se explicará más detalladamente en los siguientes apartados.

Este proyecto se basará en algoritmos de planificación con prioridades estáticas. En los planificadores estáticos, durante la fase de diseño a cada tarea se le asigna una prioridad. Después, durante la ejecución, el algoritmo de planificación simplemente tiene que ordenar la ejecución de las tareas en función de la prioridad asignada.

2.2.2. Planificación con prioridades fijas

Los dos principales algoritmos de asignación de prioridades fijas son:

- Dar más prioridad a la tarea más frecuente o ***Rate Monotonic*** (RMS). Cuanto menor es el periodo de una tarea, mayor es su prioridad.
- Dar más prioridad a la tarea más urgente o ***Deadline Monotonic*** (DMS). Cuanto menor es el plazo de una tarea, mayor es su prioridad.

En cualquiera de los dos casos, la prioridad se mantiene fija durante la ejecución del sistema, de ahí que se denomine prioridad fija.

2.2.2.1. Rate Monotonic

El algoritmo de planificación Rate Monotonic es el que se ha utilizado en este proyecto. Este algoritmo asigna prioridades a las tareas de acuerdo a su tasa de solicitud, es decir tareas con periodos cortos tendrán una alta prioridad (esto es, como una función monótona de la tasa de solicitud). Como los periodos son constantes, entonces este tipo de algoritmos son asignadores de prioridades fijas. Además, una tarea que llega con un periodo más corto puede desalojar y adelantar una tarea que esté ejecutándose.

Liu y Layland [8] demostraron que Rate Monotonic es óptimo entre todas las asignaciones con prioridad fija, y además derivaron un límite mínimo superior para el factor de utilización del procesador para un conjunto de n tareas periódicas.

Así, para un conjunto de n tareas periódicas, el factor de utilización del procesador U es la fracción de tiempo de procesador gastada en la ejecución del conjunto de tareas.

Puesto que *Ci / Ti* es la fracción de tiempo de procesador gastada en la ejecución de la tarea τ_i, el factor de utilización para un conjunto de n tareas vendrá dado por:

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

Este factor provee una medida de la carga computacional sobre la CPU debido al conjunto de tareas periódicas, por lo tanto existe un máximo valor de *U* bajo el cual el conjunto de tareas es planificable y por encima del cual no es planificable. Tales límites dependen del conjunto de tareas y del algoritmo usado para la planificación.

Test de garantía (Factor de utilización)

Un conjunto de n tareas será planificable bajo el Rate-Monotonic si se cumple que el factor de utilización del conjunto de tareas es menor que la expresión:

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

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; y asumiendo valores más elevados de *n*:

$$\lim_{n\to\infty} n\left(2^{\frac{1}{n}} - 1\right) = \ln 2 \approx 0{,}69$$

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.

Test de garantía (tiempos de respuesta)

Un conjunto de n tareas será planificable bajo cualquier asignación de prioridades si y sólo si, cada tarea cumple su plazo de ejecución en el peor caso. Siendo el peor caso de cada tarea aquel en el que todas las tareas de prioridad superior se activan a la vez que ésta.

Si lo expresamos matemáticamente el test queda formulado como sigue:

$$\forall\, 1 \leq i \leq n, \quad W_i \leq D_i$$

Donde W_i representa el instante en el que finaliza la ejecución en el peor caso. Este valor se obtiene de la siguiente expresión:

$$W_i = C_i + \sum_{\forall j \in h_p(i)} \left\lceil \frac{W_i}{P_j} \right\rceil C_j$$

devuelve el entero por exceso de su argumento.

donde $h_p(i)$ representa el conjunto de tareas con prioridad mayor que la tarea i. Tal como vemos a ambos lados de la igualdad tenemos el valor W_i, que no se puede despejar. La solución de esta expresión se obtiene de forma iterativa.

La iteración se repite hasta encontrar dos valores consecutivos (W_i^k y W_i^{k+1}) iguales. El último valor de W obtenido es el que luego emplearemos en la expresión del test de garantía. Si en alguna iteración se obtiene un valor de W mayor que el plazo máximo de ejecución de la tarea, entonces esta tarea no será planificable y por tanto el sistema tampoco.

Por otro lado se puede asegurar que este test es necesario y suficiente para asegurar la planificabilidad del sistema.

2.2.3. Planificación con prioridades dinámicas

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

$$\forall \tau_i, \tau_j \in \Pi_R : d_i < d_j : p_i > p_j$$

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

Más tarde, Mok propuso otro algoritmo óptimo, el del método de prioridad a la tarea con la menor holgura (Least Laxity First o LLF). En él, la holgura de una tarea en un cierto instante de tiempo se define como la diferencia entre su plazo de finalización absoluto y el tiempo estimado de finalización en el peor caso. LLF presenta propiedades similares a las de EDF, pero introduce una mayor sobrecarga en tiempo de ejecución, debido a los

cambios de contexto provocados por cambios en la holgura de las tareas en tiempo de ejecución.

2.3. Sistemas de Tiempo Real Distribuidos

Según Coulouris et al [14] 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".*

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

Un sistema de tiempo real distribuido será, entonces, un sistema distribuido con restricciones temporales, es decir, un sistema distribuido donde existen actividades con requisitos de tiempo real. Para cumplir los requisitos temporales, estas actividades además de necesitar ejecutar sus tareas en una unidad de procesamiento, pueden, ocasionalmente, necesitar intercambiar datos con otras tareas ubicadas en otras unidades. Para que el sistema completo cumpla sus restricciones temporales, tendrá que ser capaz tanto de realizar el procesamiento de dichas tareas en cada procesador, como de intercambiar datos entre las distintas entidades, todo de manera determinista, ajustándose a las cotas temporales impuestas por los requisitos temporales de cada una de las actividades de tiempo real existentes en el sistema. Así, la planificación de sistemas distribuidos presenta frente a la de sistemas centralizados el problema de adicional de tener que realizar la planificación de los canales de comunicaciones y la asignación de tareas a los procesadores.

En general, un sistema de tiempo real distribuido está compuesto por uno o varios procesadores, conectados entre sí a través de un bus o de una red de comunicación, que interaccionan con el entorno. Dicha interacción se realiza mediante sensores que informan del estado del sistema y de actuadores que modifican alguna característica del mismo. Por ejemplo, en un sistema de frenado ABS, existen sensores ubicados en las ruedas que controlan permanentemente la velocidad de giro de las mismas. A partir de los datos que suministra cada uno de los sensores, la unidad de control calcula la velocidad media. Si se

compara la velocidad de una rueda concreta con la velocidad media de las cuatro ruedas, se puede detectar cuándo una rueda está a punto de bloquearse. En este caso, el sistema, por medio de actuadores, reduce la presión de frenado en dicha rueda hasta que alcance su velocidad un umbral fijado por debajo del límite de bloqueo. Éste es un sistema distribuido compuesto por la unidad de control y los sensores y actuadores, en él se pueden intuir las restricciones temporales: la eficacia del ABS está directamente relacionada con la diferencia temporal entre el momento en el cual detecta el problema y en el que actúa, no pudiendo ser éste arbitrario, sino que ha de ser acotado y determinista.

Por otra parte, existen dos enfoques principales para el desarrollo de sistemas distribuidos, que originan sendos paradigmas que analizaremos a continuación:

- **Sistemas gobernados por eventos:** las acciones son activadas por la ocurrencia de eventos, que caracterizan el flujo de control del programa. Se dice que el sistema está dirigido por eventos(event-triggered systems)

- **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. Se dice que el sistema está dirigido por tiempo (time-triggered systems).

2.3.1. Sistemas basados en tiempo

Una característica importante de una planta que opera en tiempo real es la constante de tiempo de la planta (una constante de tiempo es una medida del tiempo tomado por una planta para responder a un cambio en la entrada). La constante de tiempo puede ser medida en horas para algunos procesos químicos o en milisegundos para un sistema de avión. El control realimentado requiere una tasa de muestreo la cual depende de la constante de tiempo de los procesos a ser controlados. La constante de tiempo del proceso más corta, es la tasa de muestreo más rápida que se requiere. El computador que es usado para controlar la planta debe, por lo tanto, ser sincronizado a tiempo real y debe ser capaz de cumplir todas las operaciones requeridas - medición, control y actuación – dentro de cada intervalo de muestreo. La sincronización es usualmente obtenida, añadiendo un reloj al sistema del computador, normalmente referido como un reloj de tiempo.

Las tareas basadas en tiempo típicamente son referidas como tareas cíclicas o periódicas dónde el término puede implicar que la tarea se ejecute una vez por período T, o esta se ejecute en exactamente T intervalos.

2.3.2. Sistemas basados en eventos

Existen muchos sistemas donde las acciones no son realizadas en un tiempo particular o en un intervalo de tiempo pero si en respuesta a algún evento. Por ejemplo: el apagado de una bomba o el cierre de una válvula cuando el nivel de líquido en un tanque alcance un valor determinado, etc. Sistemas basados en eventos son también usados extensamente para indicar condiciones de alarma e iniciar acciones de emergencia. Las especificaciones de sistemas basados en eventos usualmente incluyen un requerimiento que el sistema debe responder dentro de un máximo tiempo dado para un evento particular.

Los sistemas basados en eventos normalmente emplean interrupciones para informar al sistema de computación que acción es requerida. Los eventos no ocurren en intervalos deterministas y las tareas basadas en eventos son frecuentemente referidos como una tarea aperiódica o esporádica. Tales tareas pueden tener plazos expresados en términos de tiempo de iniciación o tiempo de finalización (o ambos). Por ejemplo, una tarea puede ser requerida para que comience su ejecución dentro de 5 segundos de la ocurrencia de un evento, o alternativamente esta debe producir una salida a los 5 segundos del evento.

2.3.3. Arquitecturas gobernadas por eventos vs. gobernadas por tiempo.

Tal y como se ha comentado anteriormente, en un sistema gobernado por eventos, los mensajes son enviados por la aplicación cuando ocurre un determinado evento, como un cambio en el valor de alguna entrada o la activación de una alarma. Mientras que en un sistema gobernado por tiempo, los mensajes son enviados sólo en instantes temporales predefinidos.

Seguidamente se presenta una comparativa de ambos tipos de sistemas atendiendo a sus características más importantes:

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

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

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

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

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

En general, se considera que un sistema gobernado por tiempos es adecuado para sistemas de control, donde existen muchos mensajes periódicos que deben ser intercambiados en el sistema con una variación del retardo baja o acotada. Mientras que los sistemas gobernados por eventos se adaptan mejor a la monitorización de alarmas, ya

que, aunque ocurren ocasionalmente, han de ser tratadas dentro de una latencia mínima especificada y, en el caso de un período de sobrecarga, este tipo de sistemas pueden ser diseñados para exhibir lo que se denomina degradación suave (*graceful degradation*), por ejemplo seleccionando algunas tareas para que no cumplan sus plazos y así evitar un fallo completo del sistema.

Actualmente, existe una tendencia a combinar de forma eficiente tráfico basado en eventos y en tiempo, para conseguir beneficiarse de las ventajas de los dos entornos, principalmente de la flexibilidad y eficiencia de los sistemas gobernados por eventos y la predictibilidad, propiedad de composición y seguridad de los sistemas gobernados por tiempo.

Capítulo 3

Modelos de diseño

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, puesto que éstos son la base del presente proyecto.***

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 perseguía 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 [15]. 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

Los sistemas software distribuidos se han vuelto más dinámicos permitiendo distribución, auto-reconfiguración, portabilidad y migración de aplicaciones. 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 [21][22]. 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 Iria 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.

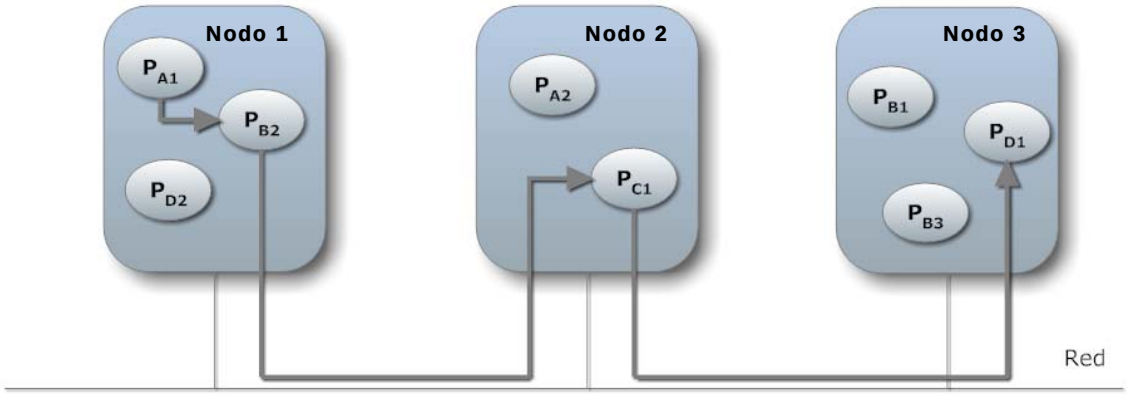


Figura 5. Ejemplo de aplicación distribuida

Como podemos ver en la Figura 5 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 $P^1{}_A$ en el nodo 1 y $P^2{}_A$ en el nodo 2. En el caso del servicio S_B se puede apreciar que está implementado en el perfil $P^2{}_B$ en el nodo 1 y por los perfiles $P^1{}_B$ y $P^3{}_B$ en el nodo 3. En el caso del servicio S_D tiene el perfil $P^1{}_D$ en el nodo 3 y $P^2{}_D$ 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 cada momento determinado para la composición de la aplicación. Como se puede observar en el ejemplo, el sistema en este caso elige los perfiles $P^1{}_A$, $P^2{}_B$, $P^1{}_C$, $P^1{}_D$ que resuelven la aplicación planteada.

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

Si, en un momento determinado el perfil $P^1{}_A$ falla, el sistema puede reconfigurar la aplicación evitando un fallo general de la misma, eligiendo como se muestra en la Figura 6, por ejemplo el perfil $P^2{}_A$. Asimismo pueden cambiar también los mensajes entre los diferentes nodos. Con lo que la aplicación resultante es:

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

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.

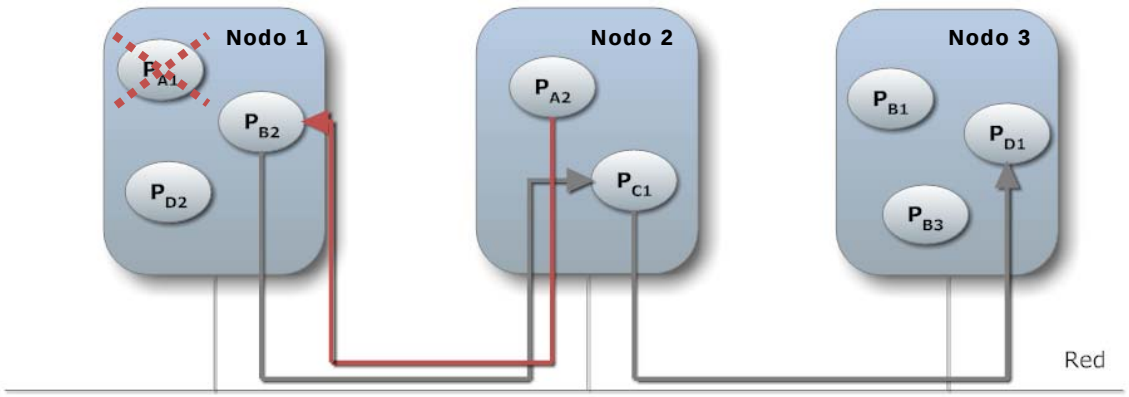


Figura 6. Ejemplo de reconfiguración de una aplicación distribuida.

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, definido 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:

$$S = \{S_1, S_2, S_3, \dots, S_n\}$$

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

$$P_i = \{P_i^1, P_i^2, P_i^3, \dots, P_i^m\}$$

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_j^i, el sistema detecta que, durante su ejecución, se produce un fallo, puede elegir otro perfil, P_k^i, que, implementando la misma funcionalidad, permita que el sistema sobreviva.

Asimismo, como las prestaciones de un perfil varían 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. Así, se caracterizará cada invocación periódica del perfil P_i como:

$$\tau_{i,j}^t = \left(C_{i,j}^t, D_{i,j}^t, T_{i,j}^t, \Phi_{i,j}^t, p_{i,j}^t\right)$$

y cada invocación esporádica como:

$$\tau_{i,j}^t = \left(C_{i,j}^t, D_{i,j}^t, T_{min\{i,j\}}^t, p_{i,j}^t\right)$$

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, en la tesis de Iria Estévez Ayres [1][12] se consideran tres tipos de tareas: productoras, consumidoras y productoras/consumidoras.

Tareas productoras. Como se muestra en la Figura 7 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:

$$\Phi_{mensP} = \Phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC}$$

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.

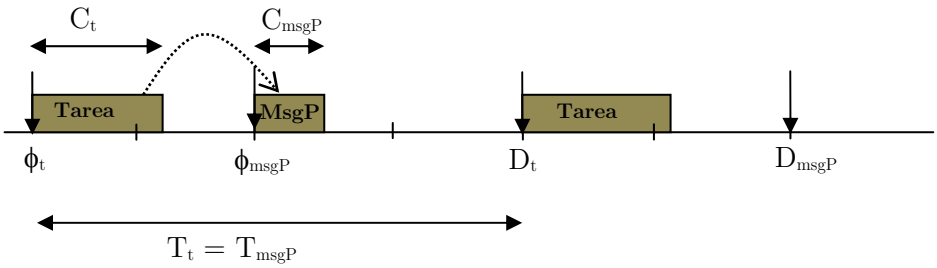


Figura 7. Tarea productora.

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 8 serán:

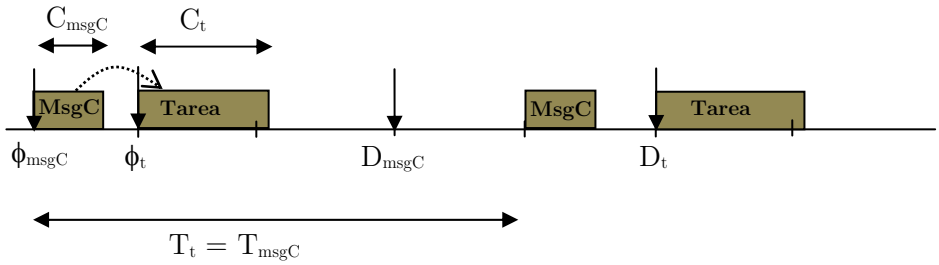


Figura 8. Tarea consumidora.

el *deadline* del mensaje consumido:

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

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

$$\Phi_t = \Phi_{mensC} + \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC}$$

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:

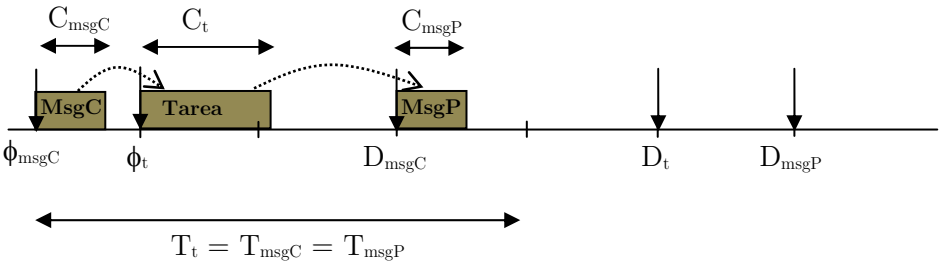


Figura 9. Tarea productora-consumidora.

el *deadline* del mensaje consumido:

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

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

$$\Phi_t = \Phi_{mensC} + \left\lceil \frac{C_{mensC}}{T_{EC}} \right\rceil T_{EC}$$

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

$$\Phi_{mensP} = \Phi_t + \left\lceil \frac{C_t}{T_{EC}} \right\rceil T_{EC}$$

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 10, 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 (PS_2) es 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 ecuaciones anteriores quedarán modificadas de la siguiente manera:

$$\Phi_t = \max_{\forall=[1\ldots n]} \left\{ \Phi^i_{mensC} + \left\lceil \frac{C^i_{mensC}}{T_{EC}} \right\rceil T_{EC} \right\}$$

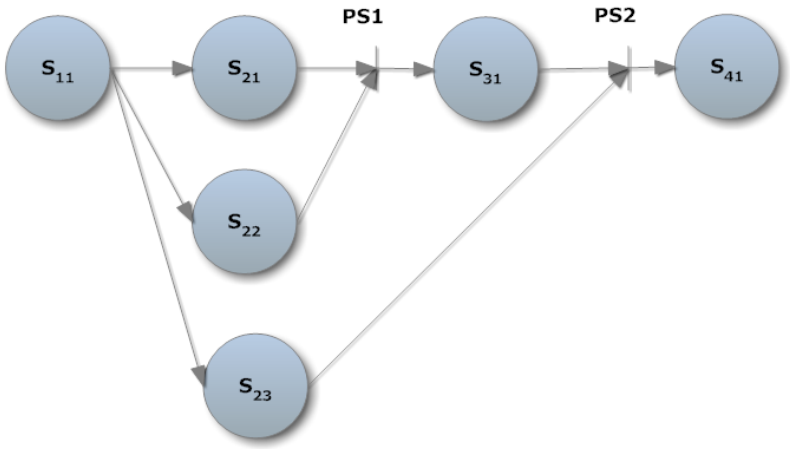


Figura 10. Aplicación con dos puntos de sincronización.

3.4. Algoritmos implementados

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 *sub-óptima* 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 [16], o mediante el uso de heurísticos desarrollados a tal efecto [17][18].

Para ver la explicación detallada del mismo ir al Capítulo 5 de la Tesis de Iria Estévez Ayres [1].

3.4.2. Algoritmo mejorado

Tal y como se dijo anteriormente, 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 evalúa 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 evalúan, 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.

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.

Para ver la explicación detallada del mismo ir al Capítulo 5 de la Tesis de Iria Estévez Ayres [1].

3.4.2.1. Heurísticos desarrollados

Los siguientes heurísticos son los que han sido desarrollados en 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 evalúan 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.

Dispersión 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.

3.4.2.2. Figuras de Mérito desarrolladas

De la misma manera se presentan a continuación las figuras de mérito, tanto globales como relativas, desarrolladas en la tesis. El algoritmo de composición escogerá como mejor combinación aquélla cuya figura de mérito sea la de menor valor. Las figuras de mérito globales propuestas pueden combinarse entre ellas, ponderándolas y ajustando la figura de mérito relativa a esa ponderación, en el caso de aplicar el algoritmo mejorado.

Minimización del tiempo de respuesta extremo a extremo. Es la relación entre el tiempo de ejecución en el peor caso de cada perfil y el plazo deseado por la aplicación. Esta figura de mérito también será implementada para los algoritmos del presente proyecto.

Maximización del tiempo de respuesta extremo a extremo. En algunos entornos se puede buscar la maximización del tiempo de respuesta para dar tiempo a un servicio en concreto a ejecutarse previamente. Por lo que la figura de mérito será 1 menos la figura de mérito anterior.

Minimización/Maximización del número de nodos físicos involucrados. Consiste en una sencilla suma contando el número de nodos físicos involucrados.

Como no se puede saber a priori el número de nodos involucrados en toda la aplicación, esta figura de mérito global no tiene figura de mérito relativa.

Minimización de la utilización media. Esta figura de mérito tiene en cuenta la utilización media en los nodos físicos para seleccionar una combinación determinada.

Capítulo 4

Implementación de Algoritmos de Composición

En este capítulo se explicarán las decisiones de diseño tomadas para la implementación de los algoritmos de composición de este proyecto. Además de explicarse los algoritmos implementados, se explicarán las funciones más relevantes que acompañan el funcionamiento de los mismos. Para dar una visión esquemática y gráfica se acompañarán de diagramas de bloques tanto las explicaciones de los algoritmos como las explicaciones de las funciones descritas.

4.1. Introducción

En este capítulo se estudia cómo realizar composición de aplicaciones de tiempo real distribuidas basadas en servicios. En primer lugar, se plantea la problemática existente en la composición de aplicaciones de tiempo real a partir de servicios previamente existentes. Ésta tiene que tener en cuenta de forma *co-dependiente* la elección de los servicios adecuados, en función de unas determinadas restricciones, que compondrán la aplicación distribuida y, la asignación de parámetros temporales a dichos servicios, asegurando que no se ponen en peligro ni la planificación ni las prestaciones del sistema completo.

Con ese fin se propusieron los algoritmos de composición de [1]. En primer lugar, se propuso un algoritmo de composición exhaustivo para la composición de aplicaciones basado en la búsqueda de una combinación de perfiles que minimizara una determinada figura de mérito global que reflejara los deseos del usuario y las restricciones impuestas por el propio sistema. Sin embargo, el uso de algoritmos exhaustivos cuando el número de combinaciones a explorar crecía, era excesivamente costoso para poder ser realizado en tiempo de ejecución. Para poder realizar la composición en tiempo de ejecución del sistema, era necesario un algoritmo mejorado [1] que, ofreciendo una combinación *sub-óptima* aunque lo suficientemente cercana a la del exhaustivo pudiera ser buena, y además explorar un menor número de combinaciones. Este algoritmo mejorado se basaba en el uso de una figura de mérito relativa que reflejara la figura de mérito global que se empleara en el exhaustivo, y, además, en el uso de heurísticos que acotaran el número de combinaciones a explorar.

Destacar que, en este capítulo, únicamente se plantea la búsqueda de una solución a la composición inicial de aplicaciones, tratando el tema de la reconfiguración de dichas aplicaciones en el Capítulo 5. Por ello, el objetivo principal en este capítulo es el estudio e implementación de algoritmos de composición de aplicaciones distribuidas basados en servicios.

A la vez que los algoritmos, se ha implementado una herramienta que permite comprobar que su implementación y funcionamiento ha sido el correcto. Dicha herramienta se ha diseñado con el fin de conseguir que los algoritmos se ejecuten bajo las mismas condiciones del sistema (es decir, las mismas condiciones en las redes, y en los nodos), de forma que la comparación que se realice entre ellos sea lo más cercana posible a la realidad.

Además, se han utilizado dos figuras de mérito relativas para el desarrollo de los algoritmos heurísticos, *minimización del tiempo de ejecución en el peor caso* y *maximización del tiempo restante antes del deadline*. Esta última se ha desarrollado e implementado en este proyecto, y por lo que se hará una explicación más exhaustiva de ella en el *Apartado 4.5.2*.

Por otro lado significar la reutilización de parte del código de los algoritmos de composición que se implementaron en el proyecto "*Implementación y evaluación de algoritmos de composición de aplicaciones distribuidas de tiempo real basadas en servicios*" ([2]).

Este proyecto es una continuación del comentado anteriormente ([2]), y aunque en un principio, el objetivo inicial era el desarrollo de una herramienta analítica que permitiera la reconfiguración de aplicaciones distribuidas en tiempo de ejecución, por motivos de definición de variables y definición de funciones poco flexibles en dicho proyecto, se decidió implementar de nuevo los algoritmos de composición, con el fin de basar los algoritmos de reconfiguración (los cuales se verán en el *Capítulo 5*) en ellos.

4.2. Decisiones de diseño

El modelo de sistema elegido para este proyecto mantiene las principales características seguidas por Iria Estévez Ayres en su tesis [1], las cuales que fueron detalladas en el *Capítulo 3*. En dicha tesis, tal y como ya se vio, uno de los objetivos que se perseguía era que el sistema tuviera la posibilidad de cambiar en tiempo de ejecución las implementaciones de los servicios que empleaba una determinada aplicación, sin que tuviese efectos en las demás aplicaciones existentes en el sistema. Este objetivo no estará presente en este capítulo, sin embargo si lo estará cuando hablemos del algoritmo de reconfiguración en el *Capítulo 5*. Por lo cual, este apartado se centrará en la implementación de algoritmos de composición inicial de aplicaciones basadas en servicios.

El modelo se basó en la abstracción que es comúnmente utilizada en el diseño de sistemas distribuidos de tiempo real: *"una aplicación como un conjunto de tareas que se ejecutan en distintos procesadores que intercambian mensajes sobre una red"*.

Las características en las que se basa el modelo de aplicaciones utilizado en este proyecto son las que se enumeran a continuación:

1. Aproximaciones en un sistema gobernado por tiempo, ya que, en un principio, no era necesario contar con la flexibilidad y eficiencia que puede brindar un sistema gobernado por eventos. Por lo tanto, las aplicaciones presentes en el sistema están gobernadas por tiempo, y proporcionan la predictibilidad y seguridad característica de estas aproximaciones.

2. Aplicaciones distribuidas, es decir, como se vio en el *Apartado 3.2*, los perfiles están dispersos en la red en diferentes nodos físicos, pudiendo varios perfiles residir en el mismo nodo y varias aplicaciones compartir el mismo perfil. Además se asumirá que la comunicación entre perfiles se realizará siempre mediante paso de mensajes regulados por los protocolos de red subyacentes. Este modelo de aplicaciones permite al sistema elegir, de entre el conjunto de combinaciones

posibles, aquella que más se adecúe a las especificaciones requeridas previamente. Así, se puede caracterizar una aplicación, como un conjunto de servicios, y cada servicio como un conjunto de perfiles.

3. Las tareas serán periódicas, y siguen las mismas características que las explicadas en el *Apartado 3.3.1.* Así se puede caracterizar un conjunto de tareas como:

$$\tau_{i,j}^{t} = \left(C_{i,j}^{t}, D_{i,j}^{t}, T_{i,j}^{t}, p_{i,j}^{t}, \Phi_{i,j}^{t}\right)$$

   Siendo caracterizadas las mismas por los siguientes parámetros:

   - *(C)*: Tiempo de ejecución en el peor caso de la tarea.
   - *(D)*: Plazo relativo o tiempo límite de finalización de la tarea (*deadline*). Éste se considerará que es igual al *deadline* del servicio.
   - *(T)*: Período de repetición de la tarea.
   - *(p)*: Prioridad de la tarea.
   - *(**Φ**)*: Desplazamiento de la tarea. En este caso se considerará que ninguna de las tareas tienen desplazamiento (*offset*), por lo que esté valor no será necesario definirlo.

4. Los mensajes se considerarán como entidades independientes, cuya transmisión será gobernada por la red. Así se puede caracterizar un mensaje según la siguiente expresión:

$$\tau_{i,j}^{m} = \left(C_{i,j}^{m}, D_{i,j}^{m}, num_{mens}, t_{resp}\right)$$

   Siendo caracterizados los mismos por los siguientes parámetros:

   - *(C)*: Tiempo de computación del mensaje o tiempo de ejecución en el peor caso.
   - *(D)*: Plazo relativo o tiempo límite de finalización (*deadline*), éste se considerará que es igual al *deadline* de la aplicación.
   - *(num_{mens})*: Posición del mensaje.
   - *(t_{resp})*: Tiempo de respuesta del mensaje.

De este modo, planteando una aplicación sencilla como la de la **Figura 11**, para evaluarla, primeramente se tendría que tener en cuenta los servicios que entran en juego, y a continuación evaluar cada uno de los perfiles de los servicios indicados, para encontrar cuales serían más óptimos para la aplicación genérica.

Como se puede ver, en este ejemplo, se ha definido que para todos los servicios (S_1, S_2 y S_3) el mejor perfil evaluado es el primero.

Figura 11. Aplicación sencilla.

También es importante analizar con qué tipo de tarea se está trabajando (tareas productoras y tareas consumidoras - *véase Apartado 3.3.1*). Una forma ilustrativa de verlo es con la Figura 11, en la cual se tendría que el servicio S_{11} estará implementado por una tarea productora, ya que se encarga de producir el mensaje $mens_1$; La tarea S_{21} estará implementando una tarea productora/consumidora, ya que por un lado consume el mensaje generado por la tarea anterior, y a su vez genera el mensaje $mens_2$; Finalmente, la tarea S_{31} estará implementando una tarea consumidora, encargada de consumir el $mens_2$.

4.3. Esquema del prototipo desarrollado

Para realizar la comprobación y validación de los algoritmos de composición implementados en este proyecto, se ha creado una herramienta que permite analizar su comportamiento. Esta herramienta realiza la llamada a los algoritmos dándoles a conocer los parámetros que son necesarios para que cumplan con la función para la que han sido implementados.

Asimismo, en el diseño del prototipo se ha decidido que los parámetros necesarios para realizar el análisis del comportamiento de los algoritmos (estructura de la aplicación, servicios de la misma, perfiles de servicios, *deadlines*, etc) fueran fijados antes de que se produjera la llamada a los mismos. Por ello, todos estos parámetros se concentraron en un fichero, el cual se pasa por parámetro a la función que ejecuta el algoritmo (Figura 13).

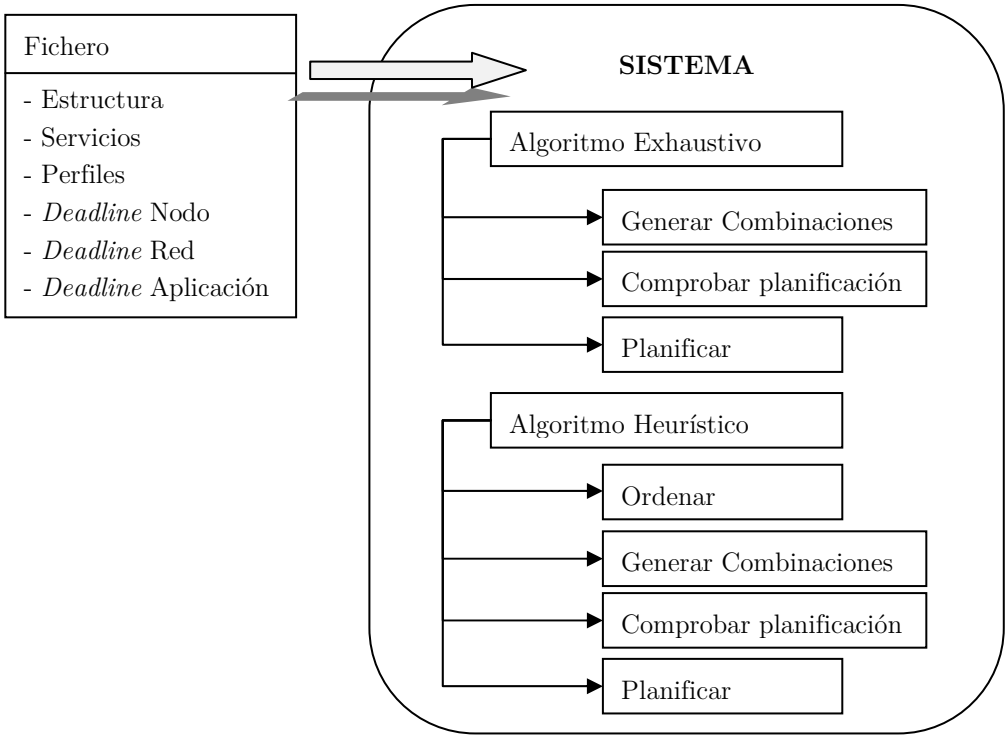


Figura 13. Estructura genérica del sistema

A continuación se listan los parámetros que se declaran en dicho fichero así como la forma en la que han sido definidos:

- **Servicios y Perfiles:** los servicios han sido definidos como una estructura. En esta estructura se puede ver el nombre del servicio y la definición de los perfiles de los que consta. Asimimo, en el ejemplo mostrado a continuación, se puede ver un servicio 'A', compuesto por cuatro perfiles.

  Los perfiles están formados por un vector de dos posiciones, el valor de la primera posición será el que defina el tiempo de computación del perfile, y será un número entre el 1 y el 5. Mientras que el valor de la segunda posición será el que defina el nodo físico al que pertenece dicho perfil.

  *servicios(i)= **struct** ('nombre_serv', 'A', 'perfil', ((ceil(5*rand) ceil(3*rand)); (ceil(5*rand) ceil(3*rand); (ceil(5*rand) ceil(3*rand); (ceil(5*rand) ceil(3*rand)));*

- **Estructura de la aplicación:** la estructura de la aplicación se definirá como una matriz, en la que se mostrará la posición de los servicios a través de números, Por ejemplo para una estructura como la mostrada en la Figura 14.

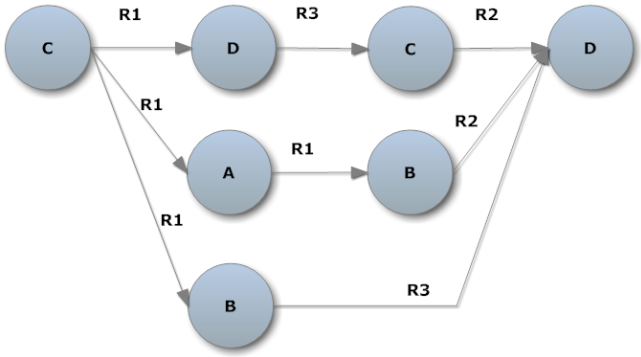


Figura 14. Ejemplo de Estructura con 3 paralelos

La matriz correspondiente a la estructura anterior sería como la que se muestra a continuación:

$$\begin{pmatrix} 11 & 21 & 31 & 99 \\ 0 & 22 & 32 & 0 \\ 0 & 23 & 0 & 0 \end{pmatrix}$$

- **Estructura de los servicios:** la estructura de los servicios se definirá como una matriz, y seguirá el mismo planteamiento que el explicado en la estructura anterior. Por lo tanto, la matriz de servicios correspondiente a la aplicación de la Figura 14:

$$\begin{pmatrix} C & D & C & D \\ 0 & A & B & 0 \\ 0 & B & 0 & 0 \end{pmatrix}$$

- **Estructura de la red:** la estructura de la red, de forma similar a las dos anteriores, se definirá como una matriz siguiendo un planteamiento similar, siendo la matriz de la estructura de red para una aplicación como la mostrada en la Figura 14 la siguiente:

$$\begin{pmatrix} R1 & R3 & R2 & 0 \\ 0 & R1 & R2 & 0 \\ 0 & R3 & 0 & 0 \end{pmatrix}$$

- ***Deadline* Nodo:** se decidió que los valores sobre los que debía oscilar el *deadline* del nodo debían ser entre 30 y 60, por lo que se utilizó la siguiente fórmula:

$$deadline\_nodo = round(30*(rand+1));$$

- ***Deadline* Red:** se decidió que el *deadline* de la red debe tener una relación directa con el número de servicios que haya en la aplicación, por lo que, a pesar de que los valores de este *deadline* son aleatorios, son dependientes de este factor. Por lo que se utilizó la siguiente fórmula:

  deadline_red=round(70(rand) + num_Servicios*30);*

- ***Deadline* Aplicación:** igual que pasa en el anterior caso, el *deadline* de la aplicación debe tener una relación directa con el número de servicios que haya en la aplicación, por lo que, a pesar de que los valores de este *deadline* son aleatorios, son dependientes de este factor. Por lo que se utilizó la siguiente fórmula:

  deadline_aplicacion = round(600(rand) + num_Servicios*30);*

Aparte de los parámetros de la aplicación, los algoritmos necesitan estar acompañados de una serie de funciones que permiten que su objetivo pueda ser cumplido, por lo que, en los siguientes apartados, se realizará una explicación tanto de los algoritmos de composición implementados (*Algoritmo Exhaustivo* y *Algoritmo Heurístico*) como de cada una de las funciones que forman parte de los mismos.

4.4. Algoritmos implementados

En este apartado se explicará con detalle el desarrollo de los dos algoritmos de composición que se han implementado en este proyecto (*algoritmo exhaustivo* y *algoritmo heurístico*).

Primeramente se hablará del *algoritmo exhaustivo*, el cual realiza una búsqueda entre todas las combinaciones posibles hasta encontrar con la que obtiene el mejor tiempo de respuesta (es importante tener en cuenta que la figura de mérito global que se ha considerado es el tiempo de respuesta).

Seguidamente se hablará de un algoritmo que mejora las prestaciones del algoritmo anterior, y el cual será llamado a partir de ahora *algoritmo heurístico*. Este algoritmo incorpora una serie de cambios sobre el exhaustivo que permiten obtener una solución *sub-óptima* lo suficientemente cercana a la óptima como para poder ser considerada como atractiva. El principio que sigue este algoritmo es la limitación del número de combinaciones que estudia, por lo que el tiempo que tarda en ejecutarse es mucho menor. Como no podía ser de otro modo, toda mejora tiene un coste, y al coste de este algoritmo se le llama error, error por no conseguir la solución más óptima, pero, como ya se

explicará en el *Capítulo 6*, el error será mínimo en comparación con la mejora en tiempos de ejecución que produce.

Antes de comenzar con las explicaciones de cada uno de los algoritmos, es importante destacar que los algoritmos de composición necesitan escoger primeramente todo el camino de la aplicación, luego planificarlo y por último, calcular el tiempo de respuesta que se obtendría para dicha aplicación.

4.4.1. Algoritmo Exhaustivo

Este algoritmo, tal y como se ha comentado anteriormente, realiza una búsqueda entre todas las combinaciones posibles hasta encontrar la que mejor tiempo de respuesta le proporciona. Por este motivo el tiempo que tarda en ejecutarse es excesivamente elevado para poder ser considerado como un algoritmo atractivo dentro de una aplicación de tiempo real. A pesar de ello, con él se asegura que la combinación elegida es la más óptima entre todas las posibles, por lo que para sistemas con pocos servicios sí puede ser un algoritmo interesante.

A continuación se explica el código del algoritmo paso a paso. Además, para una explicación más ilustrativa, se ha realizado el diagrama de bloques que se muestra en la Figura 15 y Figura 16.

1) Primeramente, se crean e inicializan debidamente las variables que van a ser usadas en el algoritmo.

2) A continuación, se realiza la generación de combinaciones que van a ser analizadas con el fin de conseguir la más óptima para la aplicación elegida. En el caso de este algoritmo de composición, y siendo fiel a su nombre, se realiza una búsqueda exhaustiva entre todas las combinaciones existentes.

3) Seguidamente se comienza con el análisis de todas las combinaciones posibles, la cual se realizará a través de diversos bucles anidados.

    a. Comienza con un bucle en el cual se analizarán todas las combinaciones posibles. Es importante tener en cuenta que se comprobará el estado en que quedarían los nodos y las redes tras la introducción de las tareas y mensajes correspondientes.

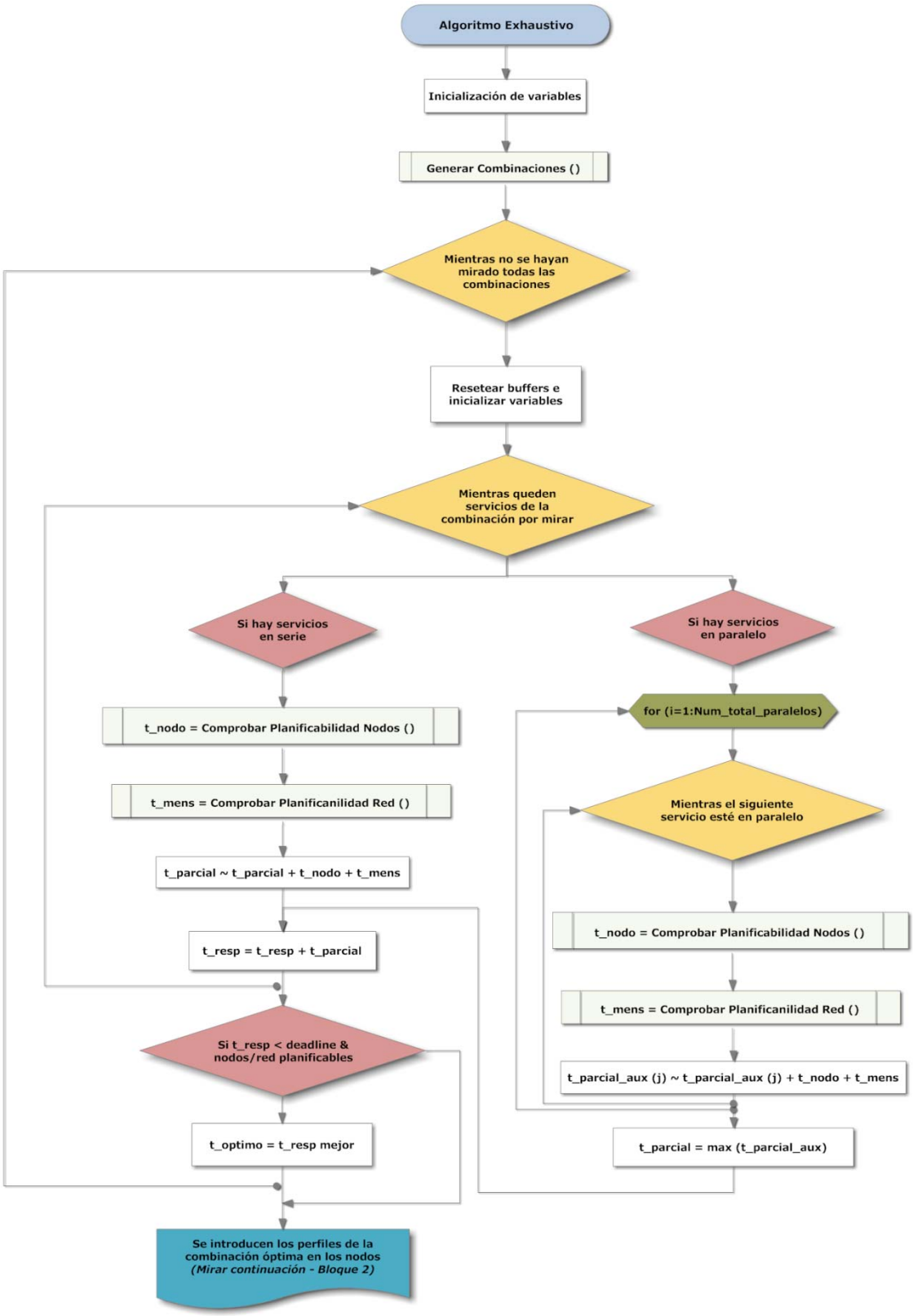


Figura 15. Algoritmo de Composición Exhaustivo – Bloque 1

b. Se resetean los *buffers*, y se inicializan algunas variables, para así conseguir que los nodos y redes estén en las mismas condiciones de utilización en la comprobación de cada combinación.

c. Comienza otro bucle, el cual se seguirá ejecutando mientras exista algún perfil de servicio de la combinación sin analizar.

d. A continuación se crea una diferencia entre las estructuras que contengan servicios en paralelo y serie. Dependiendo de si el perfil que se está analizando es parte de un paralelo o no, se realizarán un paso u otro.

    i. En caso de que el perfil no pertenezca a ningún paralelo se comprobará la planificabilidad de los nodos y de la red.

    ii. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

    iii. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

    iv. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

    v. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

e. El tiempo de respuesta de la combinación que se esté analizando será la suma de los tiempos parciales.

f. Si todavía no se han visto todos los perfiles de la combinación se vuelve al *paso c)*.

g. Si el tiempo de respuesta final que se obtiene es menor que el último tiempo de respuesta mejor encontrado (el cual en la primera iteración será el *deadline*), y además tanto los nodos como las redes son planificables, entonces se guarda el tiempo de respuesta como el más óptimo. También se guarda la combinación que lo genera.

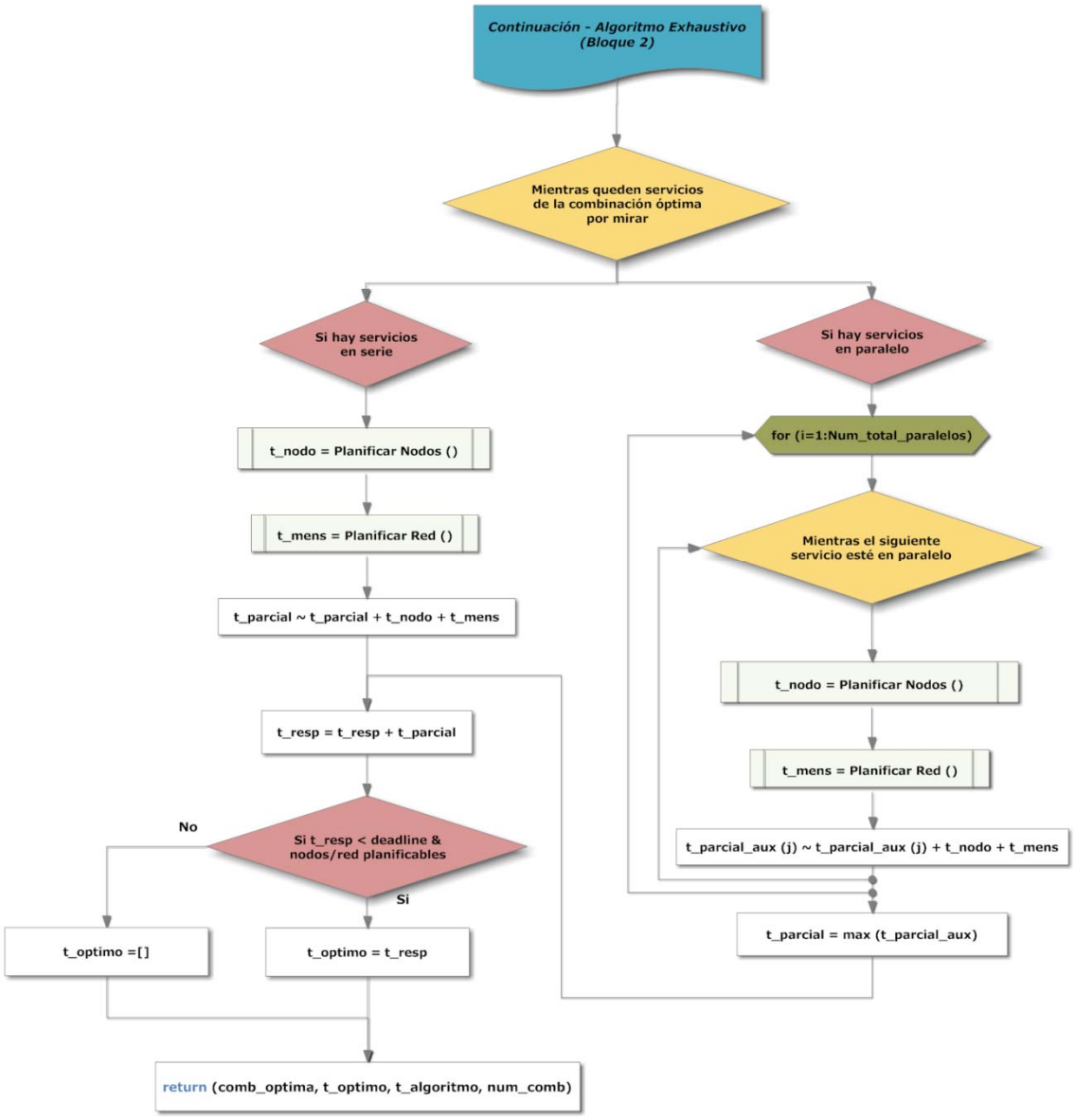


Figura 16. Algoritmo de Composición Exhaustivo – Bloque 2

h. En caso de que no se obtenga ningún tiempo de respuesta mejor, o las redes no sean planificables, entonces se devuelve el valor del tiempo de respuesta igual a 5000, y la combinación óptima como una cadena vacía.

i. Si no se han visto todas las combinaciones se vuelve al *paso a)*.

4) Una vez que se han visto todas las combinaciones, se comprueba primeramente que exista una combinación óptima. Si no existe se va al *paso 6)*.

5) Si existe alguna combinación óptima, entonces se procede a insertar la tarea en los nodos físicos y los mensajes correspondientes en las redes, para ello se realizarán los siguientes pasos, los cuales se pueden ver en la Figura 16:

    a. Se comienza con un bucle el cual se ejecutará mientras queden perfiles sin introducir en los nodos físicos.

    b. En caso de que el perfil no pertenezca a ningún paralelo se realizará la inserción de la tarea en el nodo físico correspondiente, y se introducirá también el mensaje en la red asociada, comprobando así si son planificables el nodo y la red.

    c. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

    d. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

    e. Se insertarán las tareas del paralelo en los nodos físicos correspondientes, y también, se introducirán los mensajes en las redes asociadas.

    f. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

    g. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

    h. Se comprueba que tanto los nodos como las redes son planificables. Si no es así, se pasa al *paso 6*).

    i. Una vez introducida la combinación en el sistema, y siendo planificables tanto los nodos como las redes, se devuelven los tiempos de respuesta mínimos, el tiempo de ejecución del algoritmo, la combinación óptima y el número de combinaciones totales que se han examinado durante el análisis del algoritmo.

6) En caso de no planificabilidad, se devuelve la cadena vacía como combinación óptima y su tiempo de respuesta con una valor de *Inf* (infinito), para indicar que la aplicación no es planificable.

Puede resultar evidente que si en la comprobación de las combinaciones se obtenía que con la combinación óptima indicada la aplicación era planificable, a la hora de introducir las tareas en los nodos físicos, también debería serlo. Pero debido a que estamos evaluando sistemas de tiempo real, puede que en el periodo de tiempo que se ha estado realizando la comprobación del resto de las combinaciones, se introduzcan nuevas tareas en los nodos físicos, o nuevos mensajes en las redes, produciendo que la aplicación con la combinación óptima deje de ser planificable.

4.4.2. Algoritmo Heurístico.

Como ya 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 nodos de la aplicación. Como el número de nodos de una aplicación no es posible que sea modificado, en este algoritmo se intentará acotar de diferentes maneras el número de combinaciones que se tienen que evaluar.

Por lo tanto este algoritmo se basa en la disminución del número de combinaciones a evaluar mediante el uso de heurísticos de poda (como pueden ser el hecho de quedarse con la primera combinación válida, o introducir un tamaño de bloque fijo para limitar el número de combinaciones que se deben analizar). El algoritmo heurístico aplica una figura de mérito relativa a los perfiles de ese nivel, colocándolos así de mejor a peor. Hay que tener en cuenta que la figura de mérito relativa empleada ha de estar fuertemente vinculada a la figura de mérito global.

Asimismo, en este algoritmo se ha utilizado un heurístico de poda que permite escoger de cada servicio un número determinado de perfiles, a ese número se le denominará tamaño de bloque. Por lo que el algoritmo genera combinaciones en función de dicho tamaño de bloque, y en caso de no encontrar ninguna combinación planificable, entonces amplia la selección de perfiles, con el fin de poder encontrar alguna otra que sí sea planificable. En caso de que necesite evaluar todas las combinaciones, el algoritmo será igual o más lento que el algoritmo exhaustivo, pero en el caso de que encuentre la combinación óptima en la primera iteración el tiempo de ejecución es rebajado de forma considerable.

Con el fin de conseguir que este algoritmo siempre sea más rápido que el exhaustivo, se ha introducido otro parámetro que limita el número de combinaciones totales que puede analizar el mismo. Por lo que se considerará que el umbral de combinaciones para este algoritmo será un 30% de las combinaciones totales que puede analizar el algoritmo exhaustivo:

$$Num\_Comb_{Heurístico} = 0{,}3 * Num\_Comb_{Totales}$$

A continuación se realiza una explicación detallada del algoritmo heurístico implementado en este proyecto. Además, igual que en el algoritmo anterior, se muestra el diagrama de bloques de dicho algoritmo en la Figura 17 y Figura 18.

1) Primeramente, se crean e inicializan debidamente las variables que van a ser usadas en el algoritmo.

2) Se ordenan los perfiles de cada servicio en función de su figura de mérito relativa, es decir un criterio parcial. Esta parte es primordial, porque dependiendo de si la figura de mérito es buena o mala, así se obtendrán mejores o peores resultados.

3) A continuación, y atendiendo al tamaño de bloque que haya sido seleccionado, se realiza la generación de combinaciones que van a ser analizadas, con el fin de conseguir una combinación *sub-óptima* lo más cercana a la conseguida en el algoritmo exhaustivo.

4) Seguidamente se comienza con el análisis de todas las combinaciones posibles, la cual se realizará a través de diversos bucles anidados.

    a. Primeramente se entra en un bucle, el cual se seguirá ejecutando siempre que haya bloques de combinaciones por evaluar y no se haya encontrado ninguna solución óptima, o no se haya sobrepasado el número de combinaciones totales que se pueden evaluar.

    b. Sigue con un bucle en el cual se analizarán todas las combinaciones posibles. Es importante tener en cuenta que se comprobará el estado en que quedarían los nodos y las redes tras la introducción de las tareas y mensajes correspondientes.

    c. Se resetean los *buffers*, y se inicializan algunas variables, para así conseguir que los nodos y redes estén en las mismas condiciones de utilización en la comprobación de cada combinación.

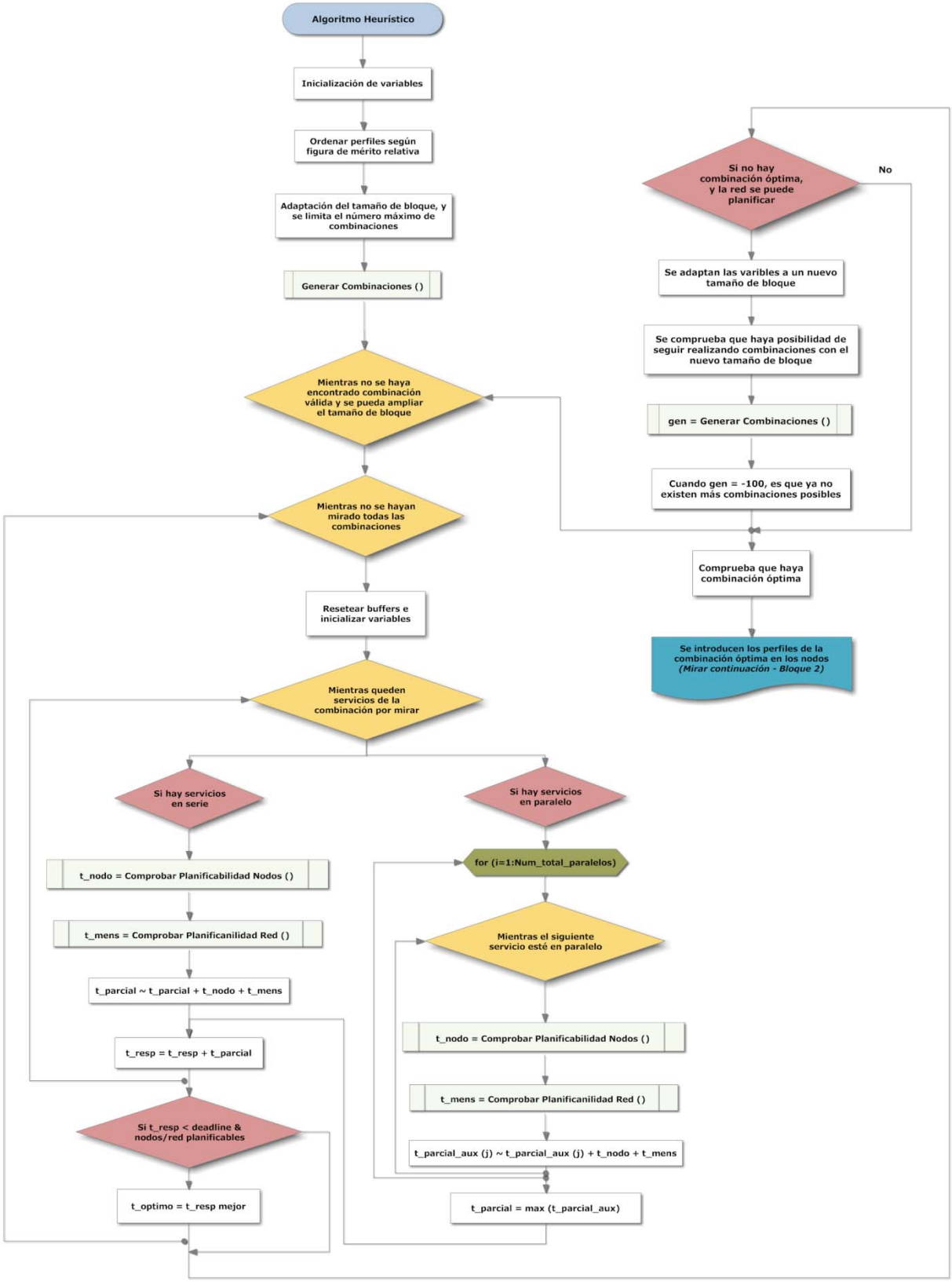


Figura 17. Algoritmo de Composición Heurístico – Bloque 1

d. Comienza otro bucle, el cual se seguirá ejecutando mientras exista algún perfil de servicio de la combinación sin analizar.

e. A continuación se crea una diferencia entre las estructuras que contengan servicios en paralelo y serie. Dependiendo de si el perfil que se está analizando es parte de un paralelo o no, se realizarán un paso u otro.

   i. En caso de que el perfil no pertenezca a ningún paralelo se comprobará la planificabilidad de los nodos y de la red.

   ii. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

   iii. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

   iv. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

   v. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

f. El tiempo de respuesta de la combinación que se esté analizando será la suma de los tiempos parciales.

g. Si todavía no se han visto todos los perfiles de la combinación se vuelve al *paso d).*

h. Si el tiempo de respuesta final que se obtiene es menor que el último tiempo de respuesta mejor encontrado (el cual en la primera iteración será el *deadline*), y además tanto los nodos como las redes son planificables, entonces se guarda el tiempo de respuesta como el más óptimo. También se guarda la combinación que lo genera.

i. En caso de que no se obtenga ningún tiempo de respuesta mejor, o las redes no sean planificables, entonces se devuelve el valor del tiempo de respuesta igual a 5000, y la combinación óptima como una cadena vacía.

j. Si no se han visto todas las combinaciones se vuelve al *paso b)*.

k. Después de haber mirado todos los caminos posibles, si no se ha encontrado ninguna combinación óptima que consiga que el sistema sea planificable, y además, los nodos y las redes del sistema sí son planificables, entonces se procederá a aumentar el tamaño del bloque del primer nodo que se marcó como "no planificable".

l. Se comprobará que el tamaño del bloque no sobrepase el número total de los perfiles de servicio del nodo. En caso de que lo haga, se igualará el tamaño del bloque al valor máximo. Si el último valor cogido ya era el número máximo de perfiles de servicio posible, entonces querrá decir que no existe ninguna combinación planificable. finalizará el algoritmo, vaciando la matriz generadora de combinaciones.

m. Si la matriz generadora de combinaciones no está vacía, es decir, todavía es posible encontrar alguna combinación posible para la planificación del sistema, se procederá a generar nuevas combinaciones posibles con los tamaños de bloque actualizados.

n. Se actualizan las variables, y si todavía hay combinaciones que se pueden estudiar, se volverá al paso *a)*.

5) Una vez que se han visto todas las combinaciones, se comprueba primeramente que exista una combinación óptima. Si no existe se va al *paso 7)*.

6) Si existe alguna combinación óptima, entonces se procede a insertar la tarea en los nodos físicos y los mensajes correspondientes en las redes, para ello se realizarán los siguientes pasos, los cuales se pueden ver en la Figura 18.

a. Se comienza con un bucle el cual se ejecutará mientras queden perfiles sin introducir en los nodos físicos.

b. En caso de que el perfil no pertenezca a ningún paralelo se realizará la inserción de la tarea en el nodo físico correspondiente, y se introducirá también el mensaje en la red asociada, comprobando así si son planificables el nodo y la red.

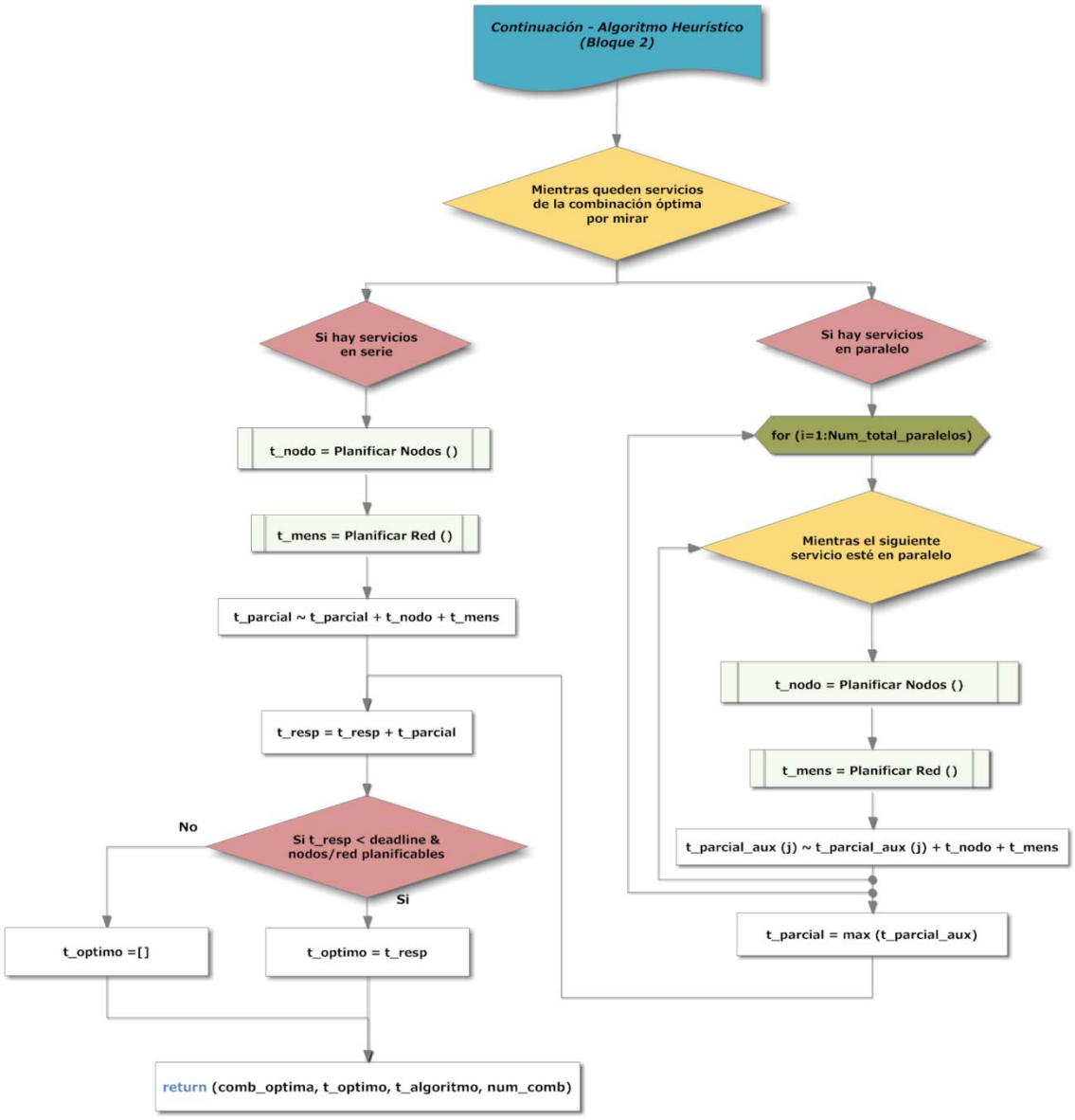


Figura 18. Algoritmo de Composición Heurístico – Bloque 2

c. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

d. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

e. Se insertarán las tareas del paralelo en los nodos físicos correspondientes, y también, se introducirán los mensajes en las redes asociadas.

f. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

g. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

h. Se comprueba que tanto los nodos como las redes son planificables. Si no es así, se pasa al *paso 6*).

i. Una vez introducida la combinación en el sistema, y siendo planificables tanto los nodos como las redes, se devuelven los tiempos de respuesta mínimos, el tiempo de ejecución del algoritmo, la combinación óptima y el número de combinaciones totales que se han examinado durante el análisis del algoritmo.

7) En caso de no planificabilidad, se devuelve la cadena vacía como combinación óptima y su tiempo de respuesta con una valor de *Inf* (infinito), para indicar que la aplicación no es planificable.

4.5. Figuras de mérito.

Según se comentó en [1], las figuras de mérito aplicadas a la composición de una aplicación deseada pueden reflejar:

- Deseos del cliente sobre el comportamiento de la aplicación.
- Requisitos impuestos por parte del administrador del sistema sobre el comportamiento general de éste y de las aplicaciones.
- Ser una combinación de los dos anteriores.

Por ejemplo, pueden preferirse aquellas combinaciones que minimicen o maximicen el número de nodos involucrados, aquéllas que minimicen la utilización global del sistema o que equilibren la carga.

Cuando existe una determinada política con respecto al sistema, impuesta por el administrador del mismo, ésta ha de estar contemplada en todas las figuras de mérito que se apliquen. Por ejemplo, una política de sistema puede ser el equilibrado y la minimización de la utilización en todos los nodos. Las figuras de mérito que se utilicen para realizar la composición, aunque incluyan otros términos como pueden ser la

minimización o maximización del plazo deseado, o la calidad de servicio medida por un determinado parámetro ofrecida por los perfiles (por ejemplo, la calidad de imagen ofrecida o el número de filtros aplicados por un determinado servicio) deben reflejar siempre este requisito del administrador, pues la instanciación de una aplicación cuya composición que no lo contemple puede provocar que el sistema se desequilibre, es decir, que ya no responda a la política del sistema.

Nótese que las replanificaciones contempladas en el algoritmo de composición, tanto en la versión exhaustiva como en la mejorada, al poder modificar los tiempos de respuesta de las acciones, pueden provocar que las aplicaciones dejen de ajustarse a la política empleada en el sistema. Si la política responde a un deseo de minimización de la utilización en los nodos y/o a un equilibrado de dicha utilización, como la utilización depende del tiempo de ejecución en el peor caso y no del tiempo de respuesta, no se vería modificada. Sin embargo, si lo que se pretende es que todas las aplicaciones minimicen el tiempo de respuesta extremo a extremo, una replanificación de una determinada aplicación puede provocar que ya no se cumpla esta restricción. El administrador del sistema será el encargado de determinar la tolerancia del sistema frente a las replanificaciones.

Los deseos del cliente deberán ajustarse a la política preestablecida del sistema y, si entran en conflicto con ésta, deberá ponderarse si es permisible una relajación en la política del sistema para aceptar a dicho cliente, o si se utiliza una figura de mérito que, ofreciendo una solución *sub-óptima* para el cliente, sí refleje dicha política.

Por otra parte, si se emplea el algoritmo de composición heurístico, deberá escogerse una figura de mérito relativa adecuada a la figura de mérito global, de tal forma que la figura de mérito global de la combinación ofrecida por el algoritmo mejorado esté aceptablemente próxima a la figura de mérito de la combinación óptima, que sería la que calcularía el algoritmo exhaustivo.

El algoritmo de composición escogerá como mejor combinación aquélla cuya figura de mérito sea la de menor valor. Las figuras de mérito globales propuestas pueden combinarse entre ellas, ponderándolas y ajustando la figura de mérito relativa a esa ponderación, en el caso de aplicar el algoritmo heurístico.

A continuación se explican las figuras de mérito, tanto globales como relativas, con las que se ha trabajado en la elaboración del presente proyecto para la composición de servicios compuestos.

4.5.1. Figura de Mérito Global

Para el desarrollo de los algoritmos de este proyecto se ha escogido como figura de mérito global la que se explica a continuación: *minimización del tiempo de respuesta extremo a extremo.*

Minimización del tiempo de respuesta extremo a extremo.

La figura de mérito desarrollada para la minimización del tiempo de respuesta se ha relacionado con el plazo deseado de la aplicación. Así, para una combinación C determinada de perfiles:

$$\mathcal{F}_1(C) = \frac{R_{e2e}}{D_{deseado}}$$

4.5.2. Figura de Mérito Relativa

Por otro lado, se ha trabajado con dos figuras de mérito relativas. Éstas han sido necesarias a la hora de realizar la implementación del Algoritmo Heurístico, puesto que el paso fundamental de dicho algoritmo es la ordenación de los perfiles de las tareas en función de una figura de mérito relativa.

La primera de ellas ya había sido desarrollada en otros estudios, mientras que la segunda forma parte de la implementación de código de este proyecto, por ello, a continuación, se detallará exhaustivamente su funcionalidad.

Minimización del tiempo de ejecución en el peor caso

La primera figura de mérito, sigue las mismas bases de la figura de mérito global, pero como no se pueden relacionar los plazos extremo a extremo, y todavía no se conocen los tiempos de respuesta (por desconocer cuál será exactamente la distribución de las tareas entre los nodos), la mejor opción será la relación entre el tiempo de ejecución en el peor caso de cada perfil y el plazo deseado por la aplicación:

$$\mathcal{F}_1(p) = \frac{C_p}{D_{aplicación}}$$

Maximización del tiempo restante antes del deadline

Como segunda opción, se ha desarrollado una figura de mérito que maximiza el tiempo restante antes del *deadline*. Esta figura de mérito se ha desarrollado de forma específica

para este proyecto, en el cual se analizará la eficiencia de la misma, aplicándola como figura de mérito relativa en los algoritmos heurísticos implementados.

Los parámetros necesarios para el desarrollo de la misma son, el tiempo de computación mínimo de los perfiles de cada servicio, el *deadline* de la aplicación, el tiempo de computación de los mensajes, el número de tareas que formarán parte de la aplicación, y por último, el tiempo de computación del perfil sobre el que se está aplicando la figura de mérito.

$$\mathcal{F}_2(p) = \frac{D_{aplicación} - \left(\sum_{i=1}^{n} C_i^{min}\right) - \left(\sum_{i=1}^{n-1} C_i^{mensajes}\right)}{n} - C_p$$

Por lo tanto el tiempo de computación de cada tarea designará el criterio parcial a aplicar para la realización de esta figura de mérito.

Esta figura de mérito lo que hace es intentar maximizar el tiempo que queda entre el tiempo de computación mínimo total de una aplicación y su *deadline*, es decir maximizar la parte en negrilla de la Figura 19.

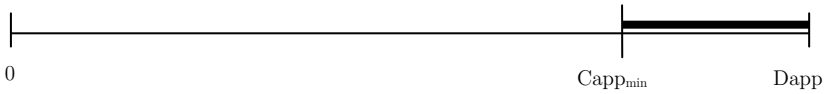


Figura 19. Explicación básica - Figura de Mérito 2.

Para conseguir maximizar la diferencia de tiempos indicada se ha seguido el siguiente procedimiento:

1. Primeramente se halla el tiempo de computación mínimo de la aplicación cogiendo los valores menores de los tiempos de computación de cada uno de los servicios implicados. A la vez se considera que todos los mensajes necesarios para dicha planificación tienen el mismo tiempo de computación. Dando la suma de todo el valor $Capp_{min}$ mostrado en la Figura 19.

2. Se ha considerado que todas las tareas tienen el mismo derecho a pasarse del tiempo restante que queda hasta el *deadline* de la aplicación, por lo que se ha dividido dicho tiempo entre la cantidad de tareas que se tengan. De forma que cada una de ellas sabe qué cantidad de tiempo puede sobrepasar. Por ejemplo si tenemos una aplicación en la que se necesitan 3 tareas, haciendo los cálculos que

se han comentado, la tarea 1 podría llegar a durar hasta T_1, la tarea 2 hasta T_2, y la tarea 3 hasta T_3 (Figura 20).

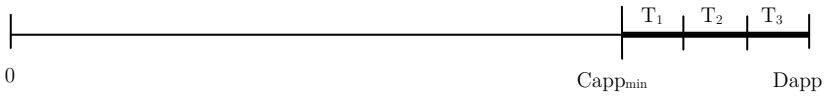


Figura 20. Explicación Básica 2 - Figura de Mérito 2

Evidentemente habrá tareas que superen su tiempo T_X (siendo T_X igual a T_1, T_2 o T_3, tal como se ve en la anterior figura), pero también habrá tareas que no, por lo que se compensan unos tiempos con otros.

3. Por último se resta el tiempo T_X del tiempo de computación de la tarea sobre la que se esté aplicando la figura de mérito. De forma que las tareas con mayor tiempo de computación harán que la diferencia de tiempo a maximizar sea menor.

Por lo tanto, para ordenar las tareas en función de esta figura de mérito, se seguiría una ordenación de mayor a menor, justo al contrario de lo que se haría si se ordenara por tiempo de computación, de forma que la que genere mayor figura de mérito será la que minimice el tiempo de respuesta de la aplicación y por lo tanto, la que maximice la diferencia de tiempos entre el tiempo de respuesta de la aplicación y su *deadline*.

4.6. Implementación

A continuación se realizará una explicación detallada de las funciones que componen y hacen posible el funcionamiento de los algoritmos de composición implementados y explicados anteriormente.

4.6.1. Ordenar

Esta función se encarga de ordenar el conjunto de tareas/mensajes que recibe por parámetro con respecto a la figura de mérito relativa con la que se esté analizando el algoritmo.

En el caso de las tareas, irá comparando unas con otras y colocándolas en orden en función de la figura de mérito a utilizar. En el caso de los mensajes, los colocará en orden decreciente de tiempos de computación/*deadlines*, obteniendo finalmente un conjunto de mensajes ordenados de menor a mayor tiempo de computación/*deadline*.

Se han implementado dos funciones diferentes: una función válida para los mensajes y tareas que se ordenan en función de la figura de mérito 1 (*ordenar()*), y otra función para las tareas que se ordenan en función de la figura de mérito 2 (*ordenar_2()*).

1. Ordenar (Figura de Mérito 1)

function *[ordenados] =* ***ordenar*** *(ConjuntoMensajes, funcion)*

Como ya se ha comentado anteriormente, se utilizará esta función cuando se quieran ordenar mensajes o tareas en función de la figura de mérito 1. En la Figura 21 se muestra el diagrama de bloques de esta función.

Los parámetros a tener en cuenta son:

- *ConjuntoMensajes*: Matriz que contendrá las distintas tareas o mensajes que se quieren ordenar.
- *funcion*: Parámetro que ayuda a distinguir entre tareas y mensajes, ya que tendrán diferente algoritmo dependiendo si es uno u otro.
- *ordenados*: matriz con los mensajes/tareas recibidos por parámetro ya ordenados.

A continuación se realizará una explicación detallada de esta función:

1) Primeramente se discriminará entre los parámetros que sean "tareas" o "mensajes", ya que como se ha dicho, seguirán algoritmos diferentes.
2) Mensajes: Se creará una variable auxiliar en la que se irán almacenando los mensajes ya ordenados.
    a. Seguidamente nos encontramos con dos bucles anidados, los cuales tendrán la función de ordenar de menor a mayor los mensajes según el *deadline* de cada uno.
    b. En caso de que existan dos mensajes con el mismo *deadline*, se ordenarán según su tiempo de computación.
    c. Una vez ordenados todos los mensajes, se realiza una copia de la variable de entrada y se sustituyen las filas debidamente ordenadas.
3) Tareas: Siguiendo un planteamiento similar al anterior, se creará una variable auxiliar en las que se irán almacenando las tareas ya ordenadas.

a. Seguidamente nos encontramos con dos bucles anidados, lo cuales realizarán la función de ordenación de las tareas en función de su tiempo de computación.

b. En caso de que existan dos tareas con el mismo tiempo de computación, se ordenarán según su *deadline*.

c. Una vez ordenados todas las tareas, se realiza una copia de la variable de entrada y se sustituyen las filas debidamente ordenadas.

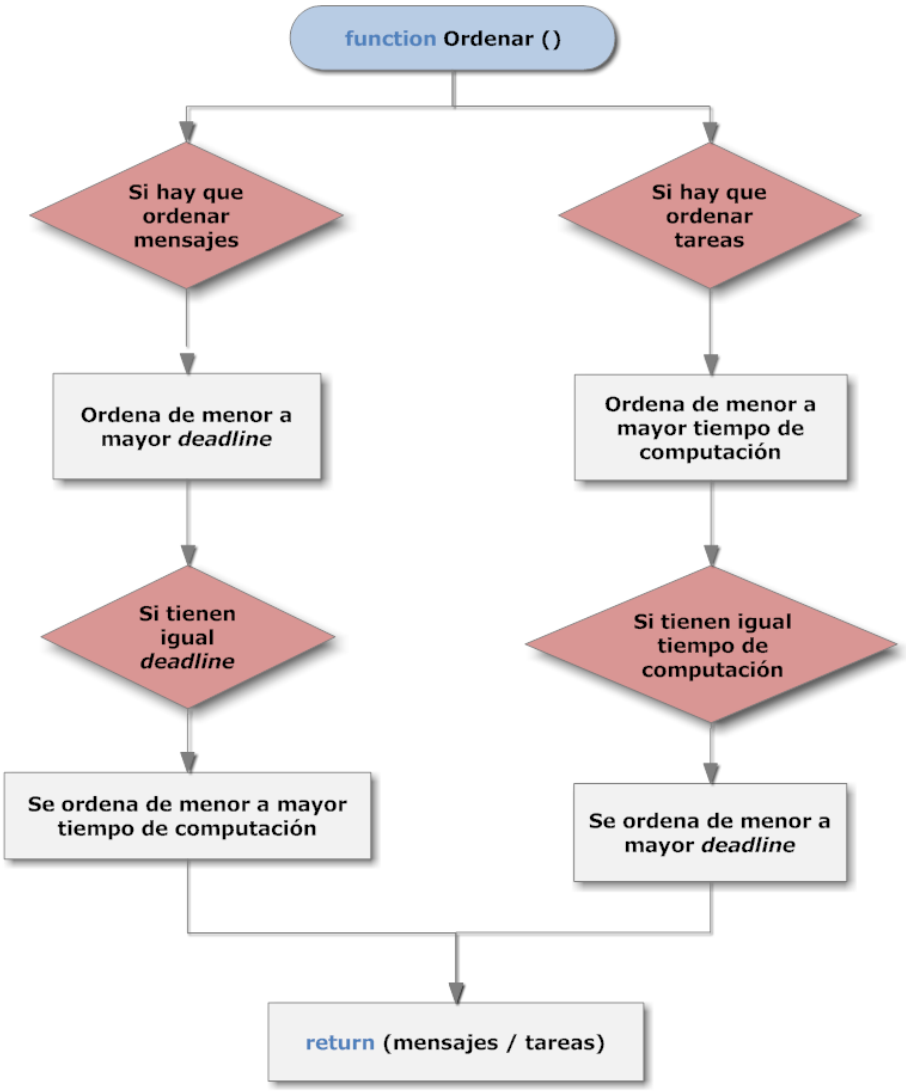


Figura 21. Función Ordenar ()

2. Ordenar_2 (Figura de Mérito 2)

function *[ordenado] =* ***ordenar2*** *(Perfil, alfa, n)*

Esta función, sin embargo, sólo se utilizará cuando haya que ordenar las tareas en función de la figura de mérito 2. En la Figura 22 se muestra el diagrama de bloques de dicha función.

Los parámetros a tener en cuenta son:

- *Perfil*: Perfil que va ser ordenado en función de la figura de mérito 2.
- *alfa*: Parámetro que será igual a la suma de los menores tiempos de respuesta de cada servicio más los tiempos de respuesta de los mensajes que se transmiten por la red.

$$alfa = \sum C_{min} + \sum mens$$

- *n*: Número total de tareas que se desean ordenar.
- *ordenado*: Perfil debidamente ordenado en función de la figura de mérito solicitada.

A continuación se realizará una explicación detallada de esta función:

1) Primeramente se creará una matriz auxiliar en la cual se introducirán los parámetros del perfil que se quiere ordenar.
2) Además se incluirá una nueva columna en la cual se irán introduciendo los nuevos valores relacionados con la figura de mérito 2, a partir de la cual se realizará la ordenación.
3) Seguidamente nos encontramos con dos bucles anidados, al igual que en la función anterior. Estos bucles serán los encargados de ordenar las tareas en función de la figura de mérito 2.
4) En caso de que existan dos tareas con el mismo parámetro obtenido a través de la figura de mérito, se ordenarán según su *deadline*.
5) Una vez ordenadas todas las tareas, se realiza una copia de la variable de entrada y se sustituyen las filas debidamente ordenadas.

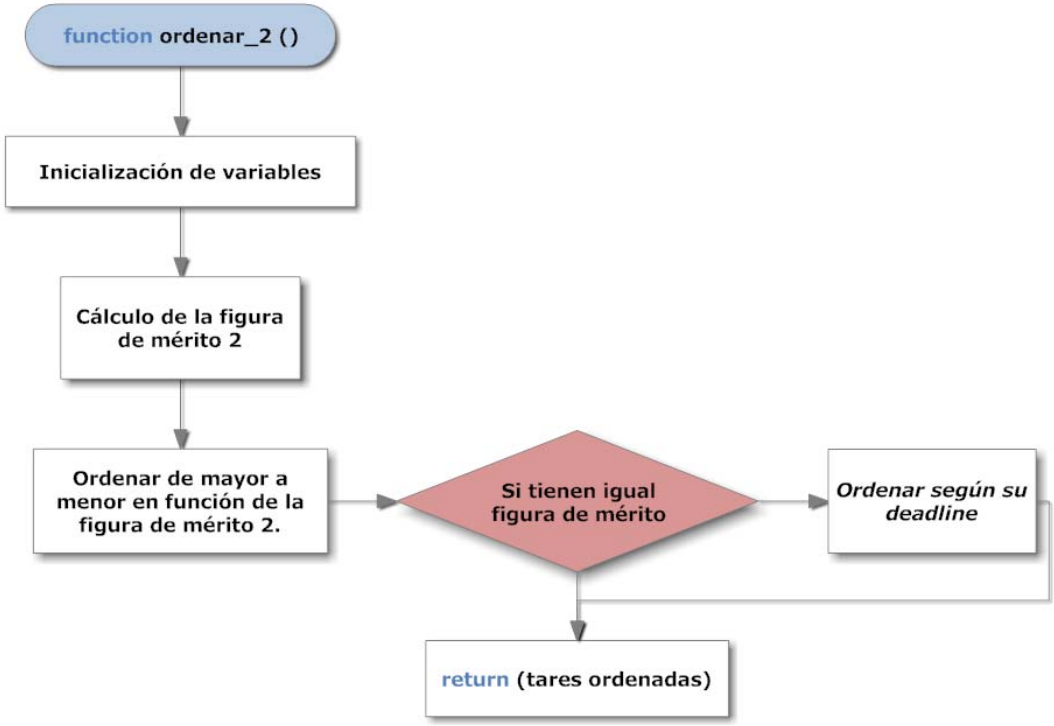


Figura 22. Función Ordenar 2 ()

4.6.2. Generar Combinaciones

Para poder generar todas las combinaciones posibles de una aplicación, se ha desarrollado la función *Generar Combinaciones*.

En esta función hay que tener en cuenta que el número de combinaciones aumenta de forma exponencial con el número de servicios que haya en la aplicación, dando igual si el servicio pertenece a servicio en serie, o si pertenece a alguna rama del paralelo, puesto que a la hora de decidir si una aplicación es planificable o no, se necesita que toda la aplicación lo sea.

Por ello la fórmula que decidiría el número de combinaciones de una aplicación sería la siguiente:

$$Num\ combinaciones = \prod_{i=1}^{n} P_i$$

Donde n define el número de servicios de una aplicación, y P el número de perfiles por servicio. Por lo que se irá mirando la cantidad de perfiles que tiene cada servicio, y multiplicándolos entre sí, de ahí que las combinaciones crezcan de forma exponencial.

Como se podrá apreciar a lo largo de la explicación detallada, está función ha sido implementada de forma que sea válida tanto para el algoritmo exhaustivo, como para el algoritmo heurístico.

La definición de la función sería la siguiente:

function *[combinaciones]=* ***generar_combinaciones*** *(long, nodos, algoritmo, tam_bloque, ultimo_long, modificado)*

Los parámetros de entrada a tener en cuenta son:

- *long*: vector que muestra la cantidad total de implementaciones de servicio de cada uno de los servicios que se encuentren en la aplicación.
- *nodos*: número total de servicios que forman la aplicación.
- *algoritmo*: indica el algoritmo que se está utilizando.
- *tam_bloque* (sólo para el Alg. Heurístico): tamaño del bloque elegido para el algoritmo heurístico.
- *ultimo_long* (sólo para el Alg. Heurístico): vector que muestra la cantidad de implementaciones de servicio estudiadas.
- *modificado* (sólo para el Alg. Heurístico): indica el servicio en el cual la aplicación dejó de ser planificable.

El parámetro de salida a tener en cuenta es:

- *combinaciones*: se trata de una matriz en la que están reflejadas todas las combinaciones posibles a analizar por el algoritmo que se esté ejecutando.

En la Figura 23 se puede ver la forma que tiene de trabajar esta función, las combinaciones se hallan forma iterativa (no se mostrará la forma de generar las combinaciones puesto que ya están implementadas en [2]). Además, no existe mucha diferencia de trabajar con el algoritmo exhaustivo o trabajar con el heurístico, puesto que la base es la misma, lo único que con el heurístico sólo se generan las combinaciones en función del tamaño de bloque. De esta forma, si se tiene un tamaño de bloque igual a 2, primero se generarían combinaciones con los perfiles 1 y 2, y si se vuelve a llamar a la función dentro de la misma ejecución, entonces se generarían combinaciones con los perfiles 3 y 4, y así sucesivamente.

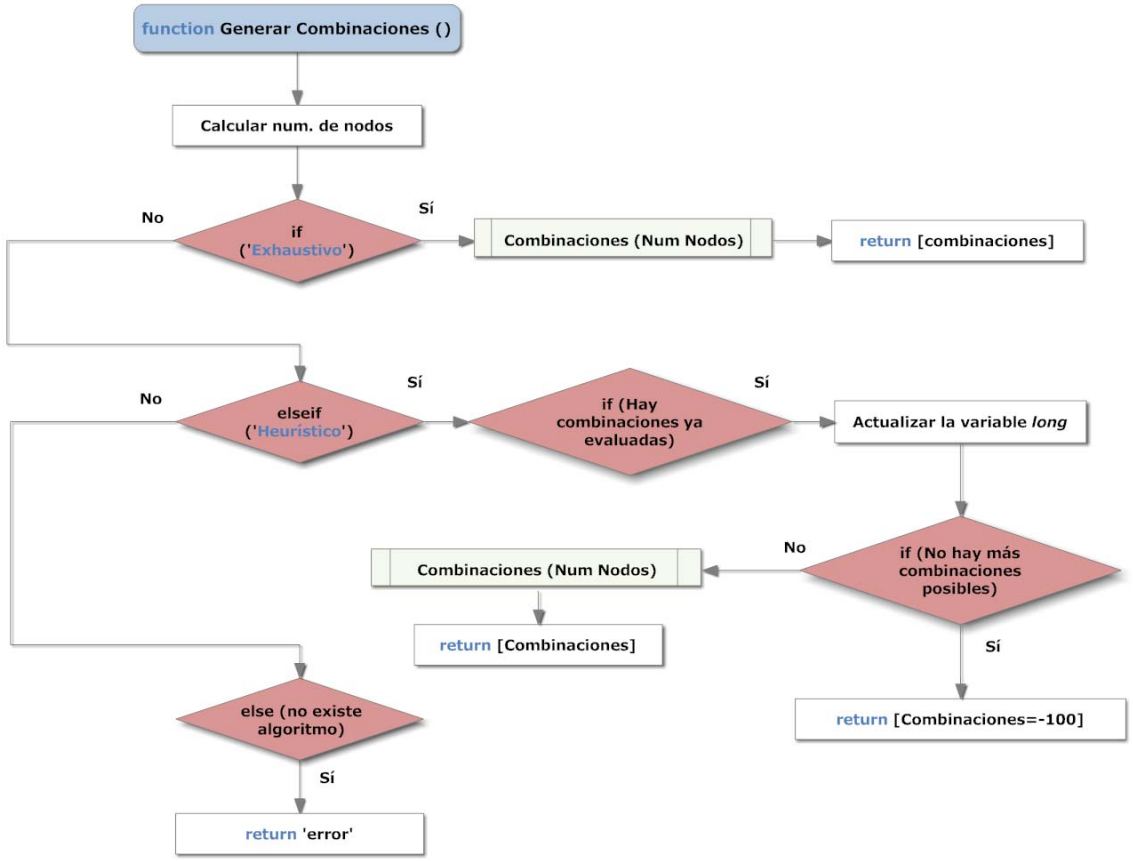


Figura 23. Función Generar Combinaciones ()

A continuación se realizará una explicación del diagrama de bloques de esta función.

1) Primeramente se inicializan las variables que se van a utilizar en la función.

2) Se calcula el número de nodos existente en la aplicación, para que a raíz de ellos, se pueda calcular el número de combinaciones posibles.

3) A continuación hay dos caminos diferentes, dependiendo del algoritmo que se esté utilizando:

    a. Exhaustivo: Se van calculando las combinaciones posibles, con bucles anidados, en función del número de perfiles que tenga cada servicio incluido en la estructura que se esté evaluando.

    b. Heurístico: Se van calculando las combinaciones posibles, con bucles anidados, en función del número de perfiles que tenga cada servicio incluido en la estructura que se esté evaluando.

Pero además, a la hora de calcular las combinaciones, tiene en cuenta el tamaño de bloque que se ha fijado, y si el nodo que se ha pasado para incrementar el número de combinaciones tiene más perfiles, o si por el contrario, ya se han evaluado todas las posibles, en cuyo caso se devolverá un error.

4.6.3. Comprobar Planificación

Este apartado engloba dos de las funciones implementadas en [2] y las cuales han podido ser reutilizadas en este proyecto, por lo que también se realizará un resumen de ambas. Una realiza la comprobación de la planificabilidad de los nodos físicos en caso que se insertara una nueva tarea y otra realiza la comprobación de la planificabilidad de la red.

1. Comprobar Planificación Nodos Físicos

function *[plan,t_resp] =* ***comprobar_planificabilidad_Nodos*** *(tarea, periodo)*

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.

Hay que tener en cuenta que esta función nunca introducirá las tareas en los nodos físicos, sino que las introducirá en unos búferes auxiliares que simularán los nodos, y con los resultados obtenidos decidir si sería planificable o no.

Los parámetros a tener en cuenta son:

- *tarea*: vector que incluye el tiempo de computación de la tarea, así como el nodo físico en el que se debe incluir el mensaje.
- *periodo*: al tratarse de una tarea periódica y trabajar sobre RMS, el período y *deadline* coincidirán, por lo que es el tiempo máximo de respuesta del nodo, y será tomado como el tiempo máximo de respuesta de la nueva tarea que pretende ser introducida en la red.
- *plan*: es el parámetro que nos indica si la tarea introducida ha sido planificable o no. En caso afirmativo se devuelve un '1'.

- *t_resp*: es el tiempo que necesita la tarea para ser ejecutada en el nodo. Este tiempo dependerá de lo cargado que esté el nodo con las tareas que ya tenía cargadas previamente.

A continuación se realizará una explicación detallada de esta función, mostrando además un diagrama de bloques de la función en la Figura 24.

1) Se inicializan todas las variables necesarias para la ejecución de la función, y se comprueba que el número de parámetros pasados es el correcto.

2) Si el mensaje introducido es '-1000', entonces hay que resetear los búferes auxiliares. Para ello se igualan las variables auxiliares al estado de las variables de los nodos físicos, y se finaliza la función.

3) A continuación se calcula la utilización del nodo físico al que se quiere introducir la tarea. El cálculo de la utilización del nodo viene definida como el sumatorio de los tiempos de computación de cada tarea partido por el periodo de las mismas.

4) Se comprueba que la utilización calculada no supere el máximo fijado en un principio. Si se supera, entonces se devolverá un error diciendo que la tarea no es planificable.

5) Se añade la tarea al *buffer*, y se colocan dependiendo de su periodo para seleccionar su prioridad. Para ello se calculan las prioridades siguiendo la premisa: "a menor periodo, mayor prioridad", y en caso de igual periodo, mayor prioridad a la que haya entrado antes en el sistema.

6) Finalmente se calcula el tiempo de respuesta aplicando la siguiente ecuación, que como ya se dijo en el Capítulo 2, únicamente se puede resolver de forma iterativa.

$$W_i = C_i + \sum_{\forall j \in h_p(i)} \left\lceil \frac{W_i}{P_j} \right\rceil C_j$$

7) En caso de que se evalúen un máximo de iteraciones y no se encuentre un resultado óptimo, se puede considerar que el tiempo de respuesta es infinito, y por lo tanto, la tarea no planificable.

8) Si se ha conseguido un tiempo de respuesta óptimo, se comprobará que no supere el *deadline* del nodo. En caso de que lo superará, la tarea tampoco sería planificable.

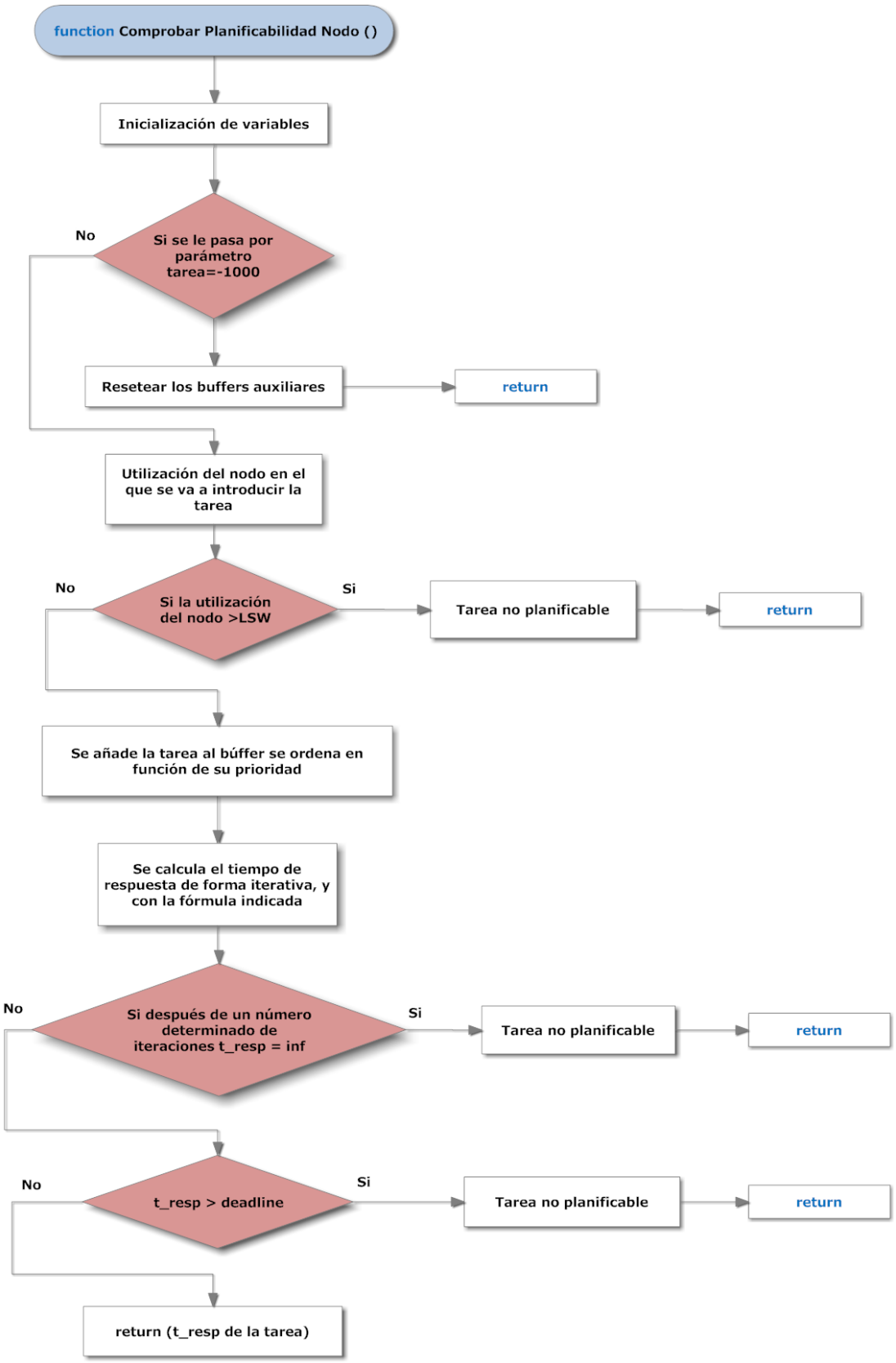


Figura 24. Función Comprobar Planificabilidad Nodos ()

9) Si el tiempo de respuesta es válido, entonces la tarea es planificable, y la función devuelve el tiempo de respuesta de la misma.

2. Comprobar Planificación Red

function *[plan,t_resp] =* ***comprobar_planificabilidad_Red*** *(mensaje, deadline)*

Esta función se encarga de realizar una comprobación sobre las redes existentes con el fin de dar como resultado si se podría añadir un nuevo mensaje con un tiempo de computación determinado, sobre la red señalada.

Igual que pasaba con la función anterior, el objetivo de esta función no es el de introducir los mensajes recibidos en las redes físicas, sino hacer un análisis sobre si sería posible la planificabilidad de la red una vez introducido el mensaje.

Los parámetros a tener en cuenta son:

- *mensaje*: vector que incluye el tiempo de computación de la tarea, así como el nodo físico en el que se debe incluir el mensaje.
- *deadline*: tiempo máximo de respuesta de la aplicación, el cual será considerado como el tiempo máximo de respuesta del nuevo mensaje.
- *plan*: es el parámetro que nos indica si la tarea introducida ha sido planificable o no. En caso afirmativo se devuelve un '1'.
- *t_resp*: es el tiempo que necesita el mensaje para ser ejecutado en la red. Este tiempo dependerá de lo cargado que esté la red con los mensajes que ya tenía cargados previamente. Este tiempo es función de los tiempos EC que sean necesarios usar.

A continuación se realizará una explicación detallada de esta función, mostrando además un diagrama de bloques de la función en la Figura 25.

1) Se inicializan todas las variables necesarias para la ejecución de la función, y se comprueba que el número de parámetros pasados es el correcto.

2) Si el mensaje introducido es '-1000', entonces hay que resetear los búferes auxiliares. Para ello se igualan las variables auxiliares al estado de las variables de las redes físicas, y se finaliza la función.

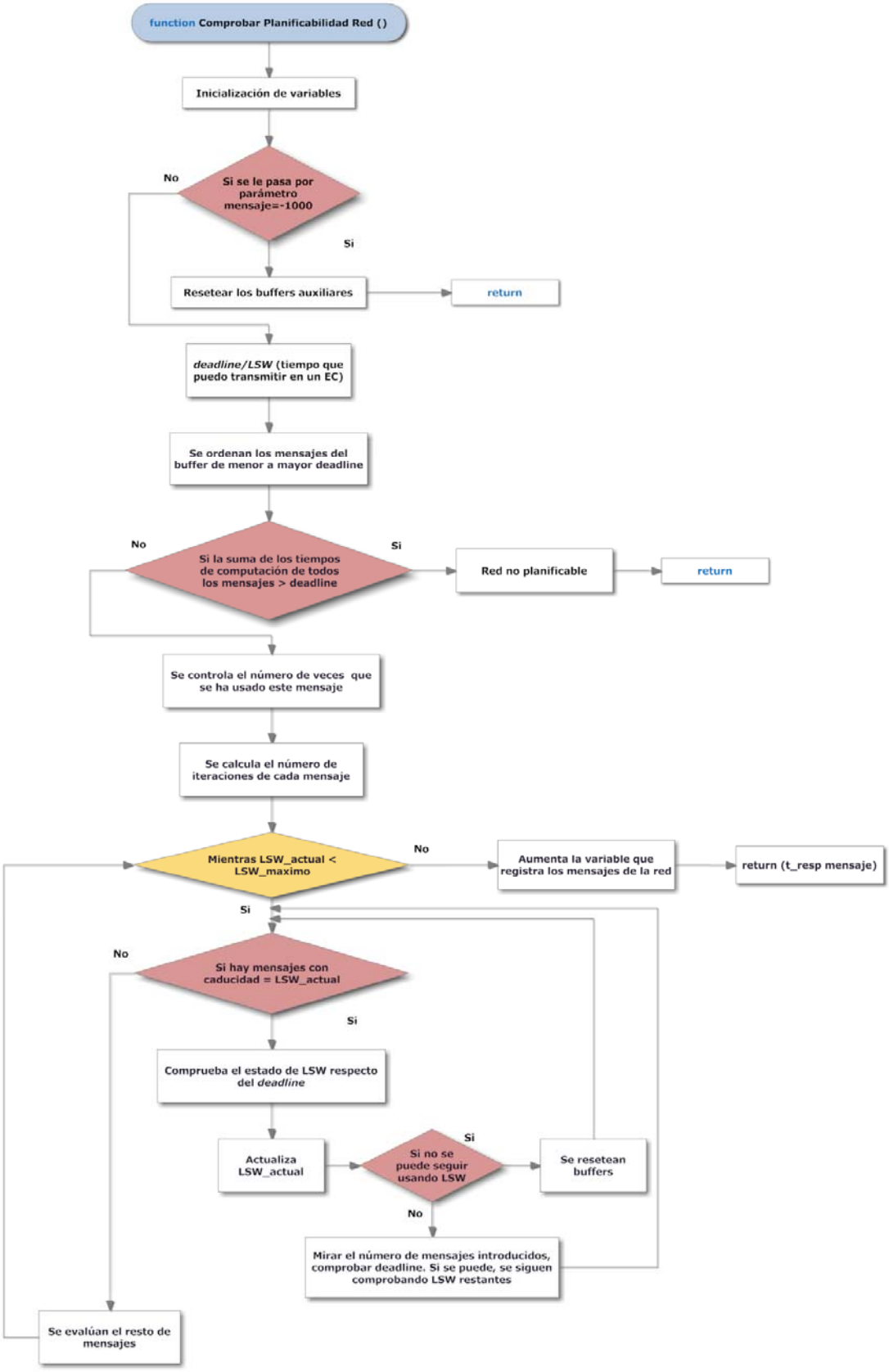


Figura 25. Función Comprobar Planificabilidad Red ()

3) Primeramente se adapta el *deadline*, es decir se dividirá este valor por LSW (tiempo que puedo transmitir en un EC), para pasarlo al formato EC, puesto que es el formato en el que trabajan las redes.

4) Se ordenan los mensajes que puedan haber en el *buffer* de menor a mayor, tal y como se explicó en la función ordenar, en caso de que alguno tenga el mismo *deadline* se ordenará en función de la carga computacional.

5) Se crea un vector con los datos del mensaje (C_i, *deadline*, número de mensaje, tiempo de respuesta).

6) Seguidamente se comprueba que la suma de los tiempos de computación de todos los mensajes que tenemos que transmitir es posible transmitirlos. En caso contrario, la red resultaría no planificable.

7) Se crea un vector que nos indique el número de veces que se ha usado el 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*.

8) A continuación se calcula el número de iteraciones de cada mensaje, lo cual nos indicará el número de veces que ha de transmitirse ese mensaje con respecto al *deadline* máximo.

9) Seguidamente, se realiza un bucle para evaluar cuáles son los mensajes cuya caducidad corresponden con la del primer LSW. Estos mensajes han sido considerados de criticidad alta, puesto que, en caso de superar su *deadline* será crítico para la planificación.

10) A partir de aquí, comienza el algoritmo que realizará el procesamiento de los mensajes que hay en la red:

    a. Se comienza con un bucle que se parará cuando el LSW actual sea superior al máximo de LSWs totales.

    b. Se realizará la comprobación de si existen mensajes cuya caducidad se corresponda con el LSW actual, los cuales serán prioritarios respecto a los demás. En caso de que no existan se pasará al *paso c)*.

A continuación, se coge como próximo mensaje a evaluar uno de los que caducan en el LSW actual. Y realiza la comprobación para ver si con las variables que se tienen se puede seguir rellenando el LSW actual o no.

i. Si todavía se puede seguir usando el LSW actual, se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no superemos el *deadline*.

   Si nos lo hemos pasado, la red no será planificable, y su tiempo de respuesta será infinito.

   Si se cumplen todas las restricciones, se procederá a introducir los datos del mensaje sobre el LSW actual. Habrá que tener especial cuidado de que no se vean incrementados los tiempos de respuesta de los mensajes antes almacenados, si es éste el caso, entonces la red no será planificable.

ii. Si no se puede seguir usando el actual LSW se resetearán los valores de carga del LSW con el que se está trabajando. Calculando los nuevos mensajes de caducidad del nuevo LSW. Pasando de nuevo al *paso b)*.

iii. Una vez introducidos los mensajes más prioritarios 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*, entonces se puede proceder a declarar la red como planificable. Si no se ha alcanzado ese nivel, se procederá a seguir comprobando nuevos mensajes para los LSW restantes volviendo al *paso b)*.

c. En caso de que no haya más mensajes con caducidad en el LSW actual, se procederá a evaluar el resto en el orden que se había impuesto. Para ello se realizará un bucle para evaluar todos los mensajes. En caso de que se produzca una salida del bucle se volverá al *paso a)*.

   i. Si todavía se puede seguir usando el LSW actual, se comprueba que solo se ha introducido el mensaje una vez por *deadline* y que no superemos el *deadline*.

Si nos lo hemos pasado, la red no será planificable, y su tiempo de respuesta será infinito.

Si se cumplen todas las restricciones, se procederá a introducir los datos del mensaje sobre el LSW actual. Habrá que tener especial cuidado de que no se vean incrementados los tiempos de respuesta de los mensajes antes almacenados, si es éste el caso, entonces la red no será planificable.

ii. Si no se puede seguir usando el actual LSW, se pasará al siguiente, reseteando los valores de carga del LSW con el que se está trabajando. Y calculando los nuevos mensajes de caducidad del nuevo LSW. Pasando de nuevo al *paso c).*

iii. Una vez introducidos los mensajes 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*, entonces se puede proceder a declarar la red como planificable. Si no se ha alcanzado ese nivel, se procederá a seguir comprobando nuevos mensajes para los LSW restantes volviendo al *apartado b).*

d. Por último, se procederá a actualizar la variable que controla todos los mensajes de la red aumentándola en una unidad.

11) Finaliza la función devolviendo el tiempo que necesita el mensaje para ser mandado por la red, y el *booleano* correspondiente a si el mensaje introducido sobre la red física que se pedía, es planificable o no.

4.6.4. Planificar

Este apartado engloba otras dos de las funciones implementadas en [2], y que junto a las otras se han podido reutilizar en el código implementado en este proyecto. Una de ellas nos dice la planificabilidad de los nodos físicos en caso que se insertara una nueva tarea, y la otra nos dice la planificabilidad de la red en función de los mensajes que la recorren.

1. Planificabilidad Nodos Físicos.

function *[plan, t_resp] =* ***planificar_Nodos*** *(tarea, periodo)*

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. Esta función repite el estudio de la planificabilidad a la hora de insertar la tarea, puesto que puede existir la posibilidad de que se haya producido algún cambio en las características de los nodos físicos.

Los parámetros a tener en cuenta son:

- *tarea*: vector que incluye el tiempo de computación de la tarea, así como el nodo físico en el que se debe incluir el mensaje.
- *periodo*: al tratarse de una tarea periódica y trabajar sobre RMS, el período y *deadline* coincidirán, por lo que es el tiempo máximo de respuesta del nodo, y será tomado como el tiempo máximo de respuesta de la nueva tarea que pretende ser introducida en la red.
- *plan*: es el parámetro que nos indica si la tarea introducida ha sido planificable o no. En caso afirmativo se devuelve un '1'.
- *t_resp*: es el tiempo que necesita la tarea para ser ejecutada en el nodo. Este tiempo dependerá de lo cargado que esté el nodo con las tareas que ya tenía cargadas previamente.

En la Figura 26 se puede ver el diagrama de bloques de esta función, que como era obvio pensar es prácticamente igual que el de la función *comprobar_Nodos*, por lo que no se volverá a describir paso a paso todo el procedimiento de la misma.

2. Planificabilidad Red

function *[plan, t_resp_mens] =* ***planificacion_Red*** *(mensaje, deadline)*

Esta función se encarga de añadir un mensaje en la red seleccionada, comprobando si éste sería planificable o no. Al igual que pasaba en la función anterior, esta función repite el estudio de la planificabilidad a la hora de insertar un nuevo mensaje, puesto que puede existir la posibilidad de que se haya producido algún cambio en las características de las redes.

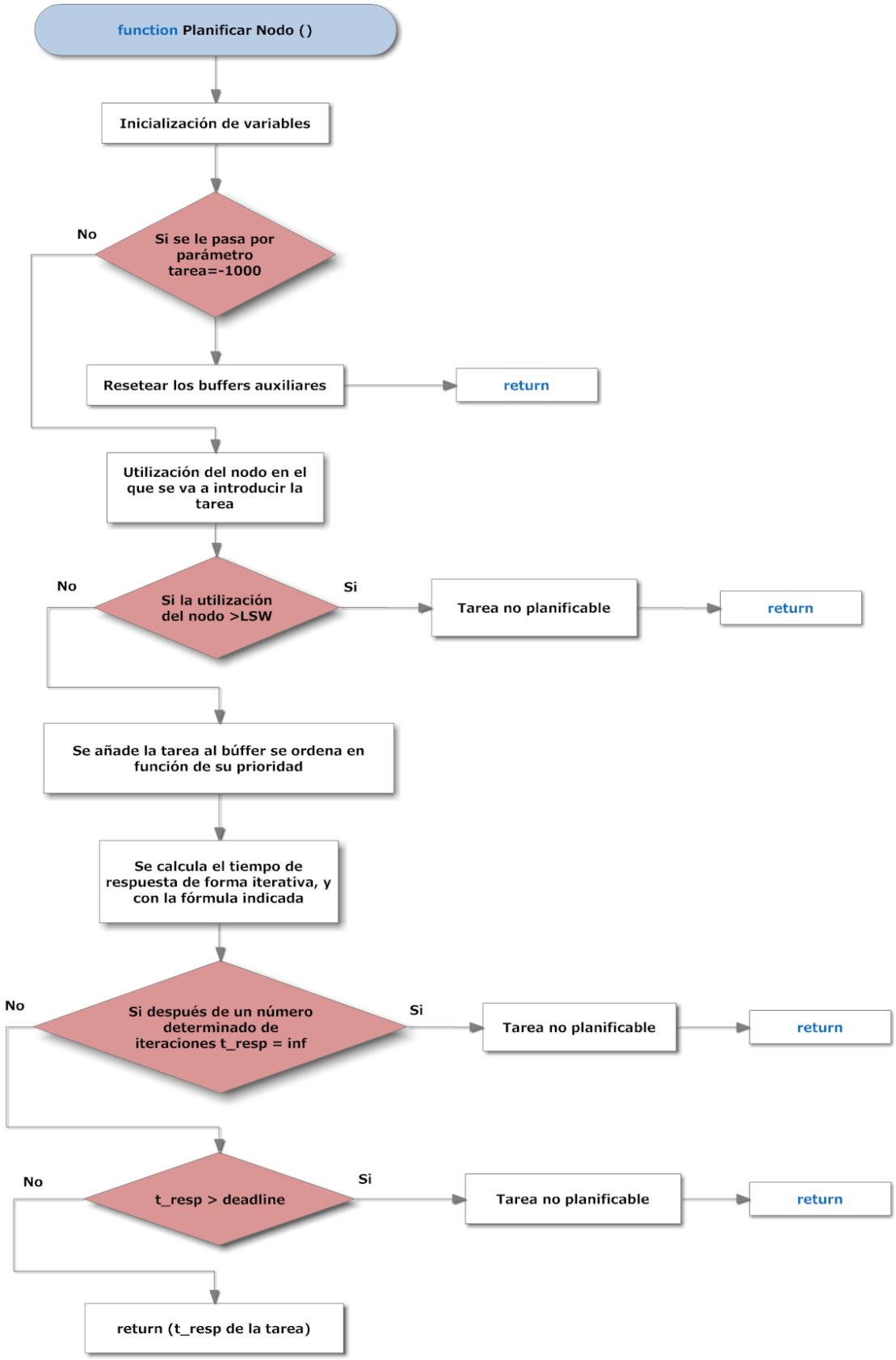


Figura 26. Función Planificar Nodos ()

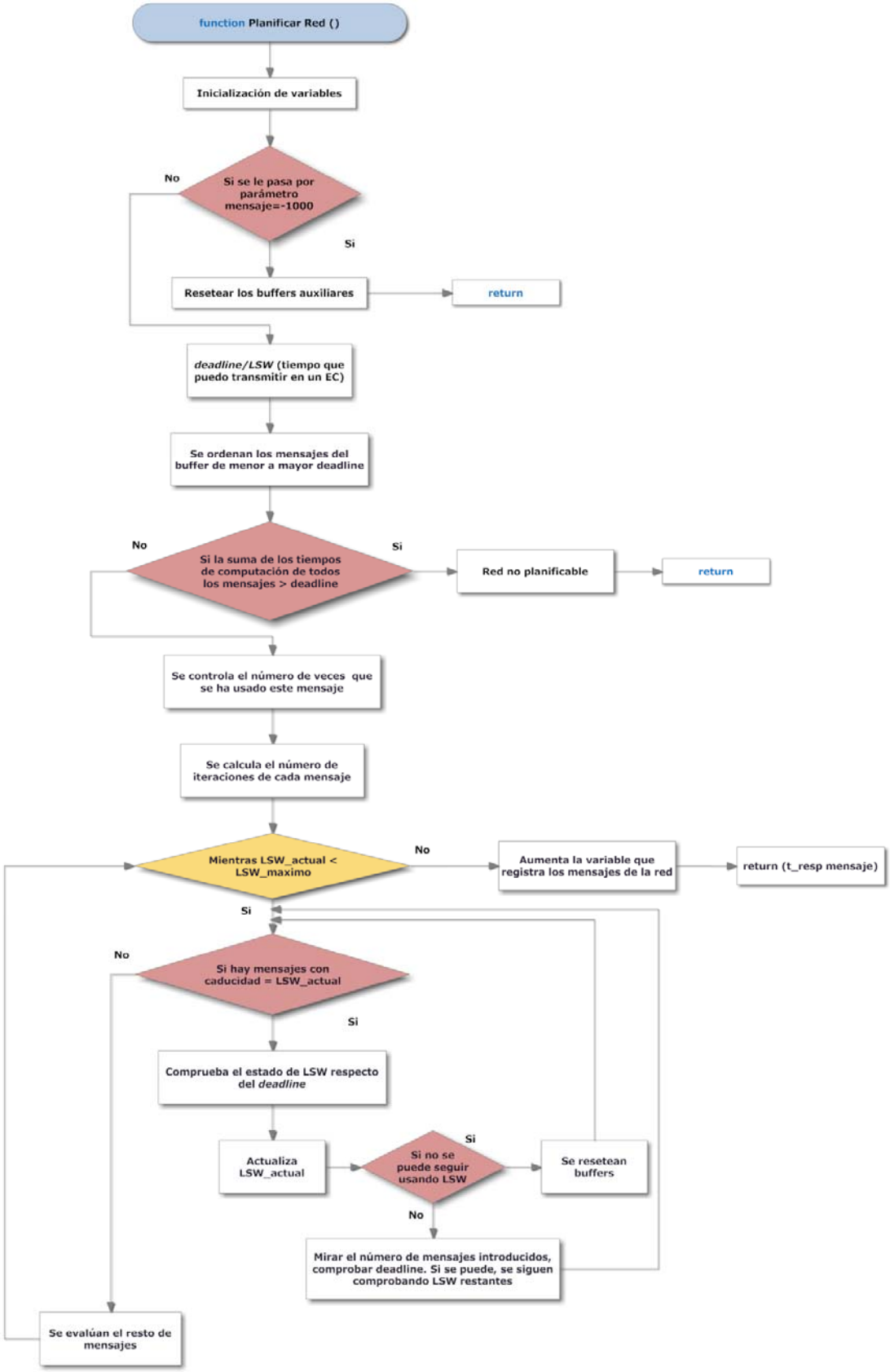


Figura 27. Función Planificar Red ()

Además, se ha considerado 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.

Los parámetros a tener en cuenta son:

- *mensaje*: vector que incluye el tiempo de computación de la tarea, así como el nodo físico en el que se debe incluir el mensaje.
- *deadline*: tiempo máximo de respuesta de la aplicación, el cual será considerado como el tiempo máximo de respuesta del nuevo mensaje.
- *plan*: es el parámetro que nos indica si la tarea introducida ha sido planificable o no. En caso afirmativo se devuelve un '1'.

- *t_resp_mens*: es el tiempo que necesita el mensaje para ser ejecutado en la red. Este tiempo dependerá de lo cargado que esté la red con los mensajes que ya tenía cargados previamente. Este tiempo es función de los tiempos EC que sean necesarios usar.

En la Figura 27 se puede ver el diagrama de bloques de esta función, que como era obvio pensar es prácticamente igual que el de la función *comprobar_Red*, por lo que no se volverá a describir paso a paso todo el procedimiento de la misma.

4.7. Conclusiones

En este capítulo se ha realizado una descripción de los algoritmos de composición implementados en este proyecto, así como las funciones más importantes que necesitan los mismos para funcionar.

No se ha realizado una descripción minuciosa de cada parte, puesto que al ser una continuación del proyecto [2], muchas de las implementaciones son similares (o incluso iguales si han podido ser aprovechadas íntegramente) a las ya explicadas en dicho proyecto.

Por lo que, este capítulo en sí, da forma a un manual básico para el usuario, el cual podría orientar de forma genérica el entendimiento de los algoritmos implementados, puesto que los algoritmos de reconfiguración deben utilizar los algoritmos de composición para realizar su objetivo, y por lo tanto, se necesitan unas ideas elementales de los mismos.

No hay que olvidar que el objetivo principal de dicho proyecto no es la implementación de algoritmos de composición, sino la implementación de algoritmos de reconfiguración.

Aún así, se han realizado una serie de pruebas para comprobar el buen funcionamiento de estos algoritmos, así como la eficiencia de la nueva figura de mérito implementada, las cuales se pueden ver en el *Capítulo 6*.

Capítulo 5

Implementación de Algoritmos de Reconfiguración

En este capítulo se explica la implementación de los algoritmos de reconfiguración diseñados en el presente proyecto. Comienza con una breve introducción acerca de las características que se incorporan al sistema gracias al uso de este tipo de algoritmos, luego se centra en las decisiones de diseño seguidas para realizar la implementación, y por último desarrolla paso por paso cada uno de los algoritmos implementados.

5.1. Introducción

Una de las características más importantes que pueden ser consideradas en los sistemas de tiempo real es el hecho de que sean dotados de una flexibilidad que permita al sistema un mejor uso de todas sus posibilidades. Esta flexibilidad se puede desarrollar de forma que el sistema tenga la posibilidad de cambiar en tiempo de ejecución las implementaciones de los servicios que emplea una determinada aplicación, sin que tuviese efectos en las demás aplicaciones existentes en el sistema. Por lo tanto, éste será el objetivo que estará presente en este capítulo, y que se desarrollará gracias a la implementación de dos *algoritmos de reconfiguración*.

La flexibilidad, puede mejorar por definición, un modelo y una arquitectura que soporta composición de servicios dinámicos, permitiendo así a las aplicaciones cambiar dinámicamente, por ejemplo, reconfigurando el conjunto de servicios del que está compuesto y soportando diferentes implementaciones para cada servicio. Las implementaciones de servicio disponibles pueden compartirse por aplicaciones diferentes y pueden ser cambiadas *on-line*, proporcionando un gran nivel de adaptabilidad en condiciones funcionales variantes, aumentando la eficiencia en el uso de los recursos del sistema.

Con objetivo de dotar a las aplicaciones de tiempo real basadas en servicios dicha flexibilidad, se propuso un esquema ([10]) de cómo realizar la reconfiguración de aplicaciones (sin entrar en detalle de qué algoritmo de composición debía utilizarse), y siguiendo este camino, se implementaron los algoritmos de reconfiguración de este proyecto.

Basándonos en lo indicado anteriormente, se propone usar como algoritmos de composición los implementados en el *Capítulo 4* (exhaustivo y heurístico), obteniendo por lo tanto un *algoritmo de reconfiguración exhaustivo* (el cual genera combinaciones en función de todos los perfiles que tenga el servicio caído) y un *algoritmo de reconfiguración heurístico* (el cual genera combinaciones de los perfiles de los servicios caídos únicamente teniendo en cuenta un tamaño de bloque que se defina previamente)

Como ya hemos explicado en el capítulo anterior, los algoritmos de composición implementados muestran soluciones bastantes válidas en cuanto a la búsqueda de la combinación más óptima, aunque cuando se habla de sistemas de tiempo real hay que tener en cuenta que estos algoritmos dejan de ser los más eficientes puesto que los tiempos de ejecución de los mismos en ocasiones no lo son.

Para ello, uno de los principales conceptos que se tuvieron en cuenta, fue el hecho de que los tiempos de ejecución pudieran ser disminuidos de forma considerable, sin descuidar el aumento del error respecto de la figura de mérito que esto conllevaba. Por lo que, otro requisito a tener en cuenta a la hora de realizar el desarrollo de los algoritmos de reconfiguración es el hecho de conseguir una buena relación entre tiempo de ejecución y error de la combinación, o lo que es lo mismo, error respecto a la figura de mérito global (en este caso, dicha figura de mérito es el tiempo de respuesta de la aplicación).

Por lo tanto, en este capítulo se estudiará la forma de reconfigurar un sistema de tiempo de real, de forma que, en función de la elección de los servicios adecuados con los parámetros temporales oportunos, seamos capaces de no dañar ni la planificación ni las

prestaciones del sistema completo, consiguiendo el buen funcionamiento de la aplicación deseada. Pudiendo así realizar aplicaciones de tiempo real distribuidas más flexibles y dinámicas.

5.2. Algoritmos de reconfiguración

Tal como se comenta en [10], cuando se componen o se reconfiguran aplicaciones de tiempo real, la selección de candidatos convenientes de implementaciones de servicio tienen que estar basados en las características de tiempo real de la aplicación y del sistema. Además, si es necesaria una reconfiguración, el algoritmo debe estar limitado en tiempo, por lo que, como es de suponer, todos los algoritmos desarrollados introducen un *time-out* que reduce el número de combinaciones a ser exploradas.

Este *time-out* lo que hace es limitar el tiempo de ejecución del algoritmo de reconfiguración. Es un valor que se crea en el sistema y es variable, puesto que es altamente dependiente del tipo de estructura que tenga la aplicación a ejecutar, ya que para estructuras en serie y con pocos nodos se considerará un *time-out* muy bajo, mientras que para estructuras con varios nodos en serie y paralelo éste se verá incrementado.

Cuando un perfil deja de estar disponible en un sistema, y se aplican los algoritmos de composición que se han visto hasta ahora, se vuelven a hallar todas las combinaciones posibles que les permite el mismo, sin embargo, los algoritmos de reconfiguración que se han diseñado en esta parte del proyecto no siguen las mismas bases. Estos algoritmos, cuando detectan un perfil no disponible, estudian una nueva configuración para la aplicación, y lo hacen siguiendo los pasos mostrados en la Figura 28 y que se explican a continuación:

a. Primeramente cambian la implementación de servicio del perfil no disponible, y se vuelve a obtener el tiempo de respuesta de la aplicación, si ésta es planificable con la nueva composición (*Paso 1*), deja buscar más combinaciones y sale del algoritmo, devolviendo el tiempo de respuesta nuevo y la combinación de perfiles seleccionada (se pasa a *e)*).

b. Si la aplicación no es planificable y además el *time-out* no ha expirado, se buscan nuevas implementaciones de servicio para el perfil no disponible más los nodos adyacentes al mismo (*Paso 2*), se vuelve a recalcular mejor tiempo de respuesta y

se mira si la aplicación es planificable. Si ahora sí lo es, se devuelve el tiempo de respuesta nuevo y la combinación de perfiles seleccionada

c. Si la aplicación sigue sin ser planificable y el *time-out* todavía no ha expirado, se repite el proceso con los nodos adyacentes a los anteriores (Repetir *Paso 2* para nodos adyacentes). De forma que si el *time-out* lo permite se puede llegar a reconfigurar la aplicación entera (aunque esto no es lo que se persigue, ya que para eso ya está diseñado el algoritmo de composición exhaustivo).

d. Finalmente, y si se ha encontrado alguna combinación *sub-óptima*, la aplicación queda reconfigurada (*Paso 3*), devolviendo el tiempo de respuesta nuevo y la combinación de perfiles seleccionada (se pasa a *e)*).

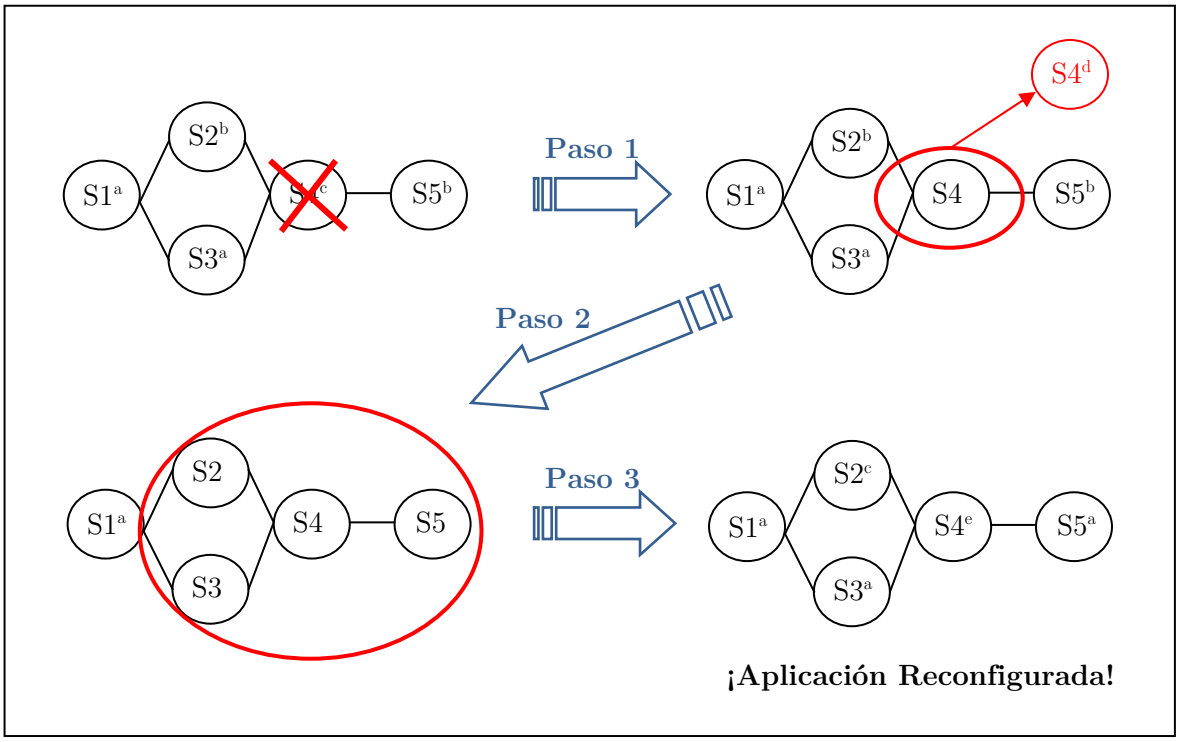


Figura 28. Pasos del Algoritmo de Reconfiguración.

Para el diseño de estos algoritmos, además del parámetro que ya se ha explicado anteriormente (*time-out*), se introduce otro nuevo término, el de *vecindario*. *Vecindario* se denomina a los nodos adyacentes al nodo o nodos que están siendo reconfigurados siguiendo las bases explicadas en los puntos anteriores.

Del modo explicado se consigue reconfigurar un sistema examinando el menor número de nodos posibles, explorando un menor número de combinaciones y, por lo tanto, consiguiendo unos tiempos de reconfiguración bastante bajos y atractivos como para ser considerados unos algoritmos bastante eficientes para sistemas de tiempo real.

Como ya se puede prever, la nueva combinación de perfiles encontrada por el algoritmo de reconfiguración es probable que no sea la óptima dado que no se vuelve a componer toda la aplicación sino sólo una parte de la misma, por lo que podemos apuntar ya que el error será, en muchos casos, mayor que el que sería introducido si se volviese a componer la aplicación completa, usando los algoritmos detallados en el *Capítulo 4*. Sin embargo, dado que el número de combinaciones que se analizan es mucho menor, también lo será el tiempo de ejecución de los algoritmos de reconfiguración, lo cual resulta beneficioso si deseamos realizar una reconfiguración en tiempo de ejecución del sistema.

Nótese que a medida que crecen el número de perfiles a reconfigurar de la aplicación, el número de combinaciones a analizar también crece (de forma exponencial con el número de perfiles), acercándose los tiempos de ejecución a los del algoritmo de composición que use internamente.

5.2.1. Decisiones de diseño

A la hora de hacer el desarrollo del código se decidió hacerlo de forma que cualquier algoritmo que consiguiera adaptar sus parámetros a los utilizados por los algoritmos de reconfiguración diseñados en este proyecto pudiera utilizarse, facilitando así posibles líneas futuras sobre el código del mismo.

Por lo tanto, y siendo una de las principales consideraciones que se tomaron al inicio, el código de los algoritmos de reconfiguración se creó de forma que fuera totalmente transparente al algoritmo de composición que se quisiera utilizar.

Los algoritmos implementados siguen el diagrama de bloques que se muestra en la Figura 29, en la cual se puede ver que cómo los bloques amarillos, que son los pertenecientes a los algoritmos de reconfiguración, son totalmente independientes del bloque rojo, el cual pertenece al algoritmo de composición que se utilice, funcionando de forma completamente aislada uno de otro.

Por otro lado, se ha considerado una variable auxiliar en el código de estos algoritmos, la cual nos indica qué servicios han dejado de estar disponibles en la aplicación que se esté analizando. Esta variable define con un 1 el servicio de la aplicación que no esté

disponible, y con un 0 el servicio de la aplicación que sí lo esté. Por lo que, se crea una matriz con la estructura de la aplicación, y sobre la misma se irá marcando el estado de los servicios, de forma que, analizando esa variable se puede saber cuántos servicios no están disponibles en la aplicación, y por lo tanto, sobre los que se debe aplicar la reconfiguración.

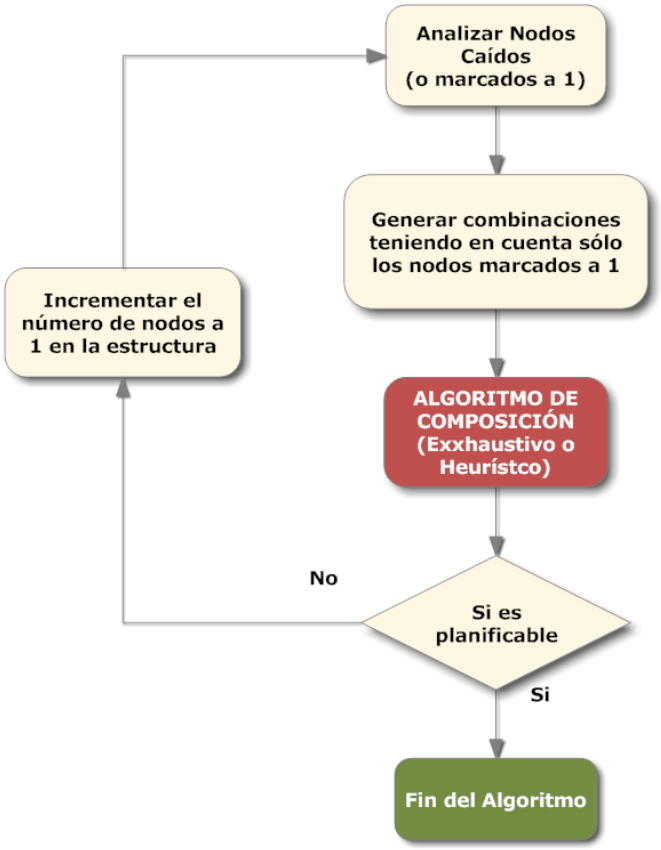


Figura 29. Diagrama básico del Algoritmo de Reconfiguración

Es importante comentar que en este capítulo no se hablarán de las funciones internas de los algoritmos, puesto que al poder utilizar los algoritmos de composición que se detallaron en el *Capítulo 4* en su totalidad, no ha sido necesaria la creación de unas nuevas.

5.2.1.1. *Time-out*

El valor del *time-out* que se ha supuesto en el diseño de los algoritmos de reconfiguración se ha basado en la relación directa que existe entre el tiempo de ejecución del algoritmo y el número de combinaciones que debe analizar el mismo, puesto que si se decrementa el

número de combinaciones, el tiempo de ejecución disminuye, y si crece el número de combinación, el tiempo de ejecución aumenta.

Por lo tanto, la fórmula que se ha propuesto para calcular el tiempo de ejecución máximo de un algoritmo es la siguiente:

$$time\ out = \frac{Num\ comb}{600}$$

Puesto que sabiendo el número de combinaciones que debe analizar el algoritmo de composición que se esté utilizando, se puede estimar el tiempo que tardará el mismo en ejecutarse, y así estimar un *time-out* óptimo para cada aplicación.

El valor del denominador no se ha elegido al azar, puesto que se ha realizado un estudio con los resultados hallados en el *Capítulo 6* (*Evaluación y pruebas de los algoritmos de composición*) y se ha concluido que ese valor consigue unos valores de *time-out* que resultan lo suficientemente óptimos como para considerar los tiempos de reconfiguración válidos. Hay que tener en cuenta que este valor se ha obtenido para aplicaciones como las mostradas tanto en el *Capítulo 6* como en el *Capítulo 7*, pudiendo no ser válidas para aplicaciones mucho más complejas.

Por otro lado, el uso de este *time-out* en el diseño que se ha realizado no es todo lo eficiente que debería, puesto que los algoritmos de reconfiguración implementados realizan la comparación entre el tiempo que lleva ejecutándose el mismo y el *time-out* cada vez que termina de analizar un *vecindario*, mientras que lo óptimo sería que el algoritmo finalizara una vez superado el *time-out* correspondiente, provocando la salida a través de una interrupción o algún agente externo.

En los siguientes apartados se explicará paso por paso los dos algoritmos de reconfiguración que se han implementado en este proyecto, tanto el *Algoritmo de Reconfiguración Exhaustivo* como el *Algoritmo de Reconfiguración Heurístico.*

5.2.2. Algoritmo de Reconfiguración Exhaustivo

En este apartado se explica el código paso a paso del algoritmo de reconfiguración que genera las combinaciones de forma parecida al algoritmo exhaustivo, puesto que aunque no se analizan todas las combinaciones posibles, porque no se varían los perfiles de todos los nodos, sí que se analizan todas las combinaciones referentes a los nodos que dejan de estar disponibles.

Además, para una explicación más ilustrativa, se ha realizado el diagrama de flujo que se muestra en la Figura 30.

1) Primeramente, se crean e inicializan debidamente las variables que van a ser usadas en el algoritmo.

2) Se entra en el primer bucle, el cual se seguirá ejecutando mientras no se encuentre ninguna combinación óptima o el *time-out* no haya expirado. Con el uso del *time-out* lo que se consigue es limitar el número de combinaciones que se tienen que analizar.

3) Se busca cuáles son los perfiles que no están disponibles. Esto, como se ha comentado antes, se conoce gracias a una variable la cual marca a 1 los servicios que no estén disponibles. Los perfiles de servicio marcados con un 0 no cambia. Por lo tanto únicamente se generarán combinaciones diferentes variando el/los perfil/s marcados con un 1.

4) Ahora se analizará combinación tras combinación hasta haber analizado cada una de las combinaciones generadas anteriormente.

    a. Comienza con un bucle en el cual se analizarán todas las combinaciones posibles. Es importante tener en cuenta que se comprobará el estado en que quedarían los nodos y las redes tras la introducción de las tareas y mensajes correspondientes.

    b. Se resetean los *buffers*, y se inicializan algunas variables, para así conseguir que los nodos y redes estén en las mismas condiciones de utilización en la comprobación de cada combinación.

    c. Comienza otro bucle, el cual se seguirá ejecutando mientras exista algún perfil de servicio de la combinación sin analizar.

    d. A continuación se crea una diferencia entre las estructuras que contengan servicios en paralelo y serie. Dependiendo de si el perfil que se está analizando es parte de un paralelo o no, se realizarán un paso u otro.

        i. En caso de que el perfil no pertenezca a ningún paralelo se comprobará la planificabilidad de los nodos y de la red.

ii. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

iii. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

iv. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

v. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

e. El tiempo de respuesta de la combinación que se esté analizando será la suma de los tiempos parciales.

f. Si todavía no se han visto todos los perfiles de la combinación se vuelve al *paso c)*.

g. Si el tiempo de respuesta final que se obtiene es menor que el último tiempo de respuesta mejor encontrado (el cual en la primera iteración será el *deadline*), y además tanto los nodos como las redes son planificables, entonces se guarda el tiempo de respuesta como el más óptimo. También se guarda la combinación que lo genera.

h. En caso de que no se obtenga ningún tiempo de respuesta mejor, o las redes no sean planificables, entonces se devuelve el valor del tiempo de respuesta igual a 5000, y la combinación óptima como una cadena vacía.

i. Si no se han visto todas las combinaciones se vuelve al *paso a)*.

5) Si después de analizar todas las combinaciones generadas todavía no se ha obtenido ninguna combinación planificable, entonces:

a. En caso de que el nodo puesto a 1 sea uno de los del paralelo, poner el resto de nodos del mismo paralelo a 1.

b. En caso de que haya un paralelo adyacente al nodo o nodos que se acaban de analizar, entonces se pondrán a 1 todos los nodos del paralelo.

c. En caso de que sólo haya nodos en serie, se pone a 1 los nodos adyacentes a los que se acaban de analizar.

Ampliando así el número de combinaciones posibles para encontrar una que sea planificable. Si el *time-out* no ha expirado, se vuelve a repetir todo el algoritmo desde *2)*.

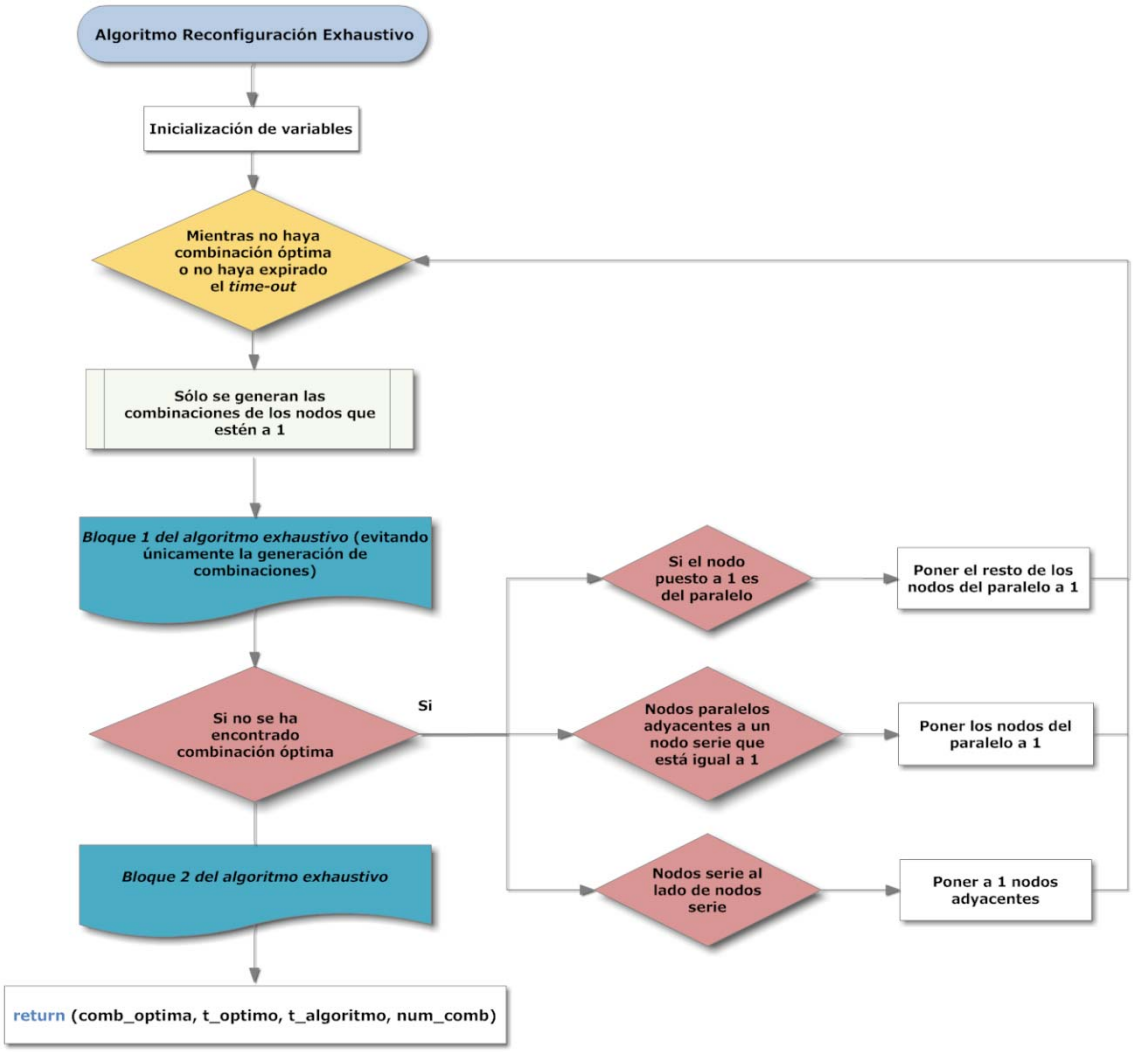


Figura 30. Algoritmo de Reconfiguración Exhaustivo

6) Si por el contrario hay una solución *sub-óptima* que sí que es planificable entonces se sale del bucle que comenzaba en *2)*.

7) Una vez que se han visto todas las combinaciones o ha expirado el *time-out*, se comprueba si existe una combinación óptima. Si no existe se va al *paso 9).*

8) Si existe alguna combinación óptima, entonces se procede a insertar la tarea en los nodos físicos y los mensajes correspondientes en las redes, para ello se realizarán los siguientes pasos:

    a. Se comienza con un bucle el cual se ejecutará mientras queden perfiles sin introducir en los nodos físicos.

    b. En caso de que el perfil no pertenezca a ningún paralelo se realizará la inserción de la tarea en el nodo físico correspondiente, y se introducirá también el mensaje en la red asociada, comprobando así si son planificables el nodo y la red.

    c. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

    d. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

    e. Se insertarán las tareas del paralelo en los nodos físicos correspondientes, y también, se introducirán los mensajes en las redes asociadas.

    f. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

    g. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

    h. Se comprueba que tanto los nodos como las redes son planificables. Si no es así, se pasa al *paso 9*).

    i. Una vez introducida la combinación en el sistema, y siendo planificables tanto los nodos como las redes, se devuelven los tiempos de respuesta mínimos, el tiempo de ejecución del algoritmo, la combinación óptima y el número de combinaciones totales que se han examinado durante el análisis del algoritmo.

9) En caso de no planificabilidad, se devuelve la cadena vacía como combinación óptima y su tiempo de respuesta con una valor de 5000, para indicar que la aplicación no es planificable.

10) Para finalizar el algoritmo de reconfiguración mejorado exhaustivo, se devuelven: el tiempo de ejecución del mismo, el tiempo de respuesta mínimo, la primera combinación *sub-óptima* planificable que ha encontrado y el número de combinaciones totales que se han analizado.

Como ya se ha comentado anteriormente, dicho algoritmo es muy parecido al algoritmo de composición exhaustivo, puesto que utiliza gran parte de su código, aunque reduciendo el número de exploraciones que se realizan.

5.2.3. Algoritmo de Reconfiguración Heurístico

A continuación se explica el código paso a paso del algoritmo de reconfiguración que genera las combinaciones de forma parecida al algoritmo heurístico, puesto que, aunque no se analicen las combinaciones con todos los nodos de la aplicación, sí que se analizan las combinaciones referentes a los nodos que cambian, y además, las implementaciones de servicio que se cogen se hacen de la misma forma que el algoritmo heurístico, con un tamaño de bloque determinado.

Además, para una explicación más ilustrativa, se ha realizado el diagrama de flujo que se muestra en la Figura 31.

1) Primeramente, se crean e inicializan debidamente las variables que van a ser usadas en el algoritmo.

2) Se ordenan los perfiles de cada servicio en función de su figura de mérito relativa, es decir un criterio parcial. Esta parte es primordial, porque dependiendo de si la figura de mérito es buena o mala, así se obtendrán mejores o peores resultados.

3) Se entra en el primer bucle, el cual se seguirá ejecutando mientras no se encuentre ninguna combinación óptima o el *time-out* no haya expirado. Con el uso del *time-out* lo que se consigue es limitar el número de combinaciones que se tienen que analizar.

4) Se busca cuáles son los perfiles que no están disponibles. Esto, como se ha comentado antes, se conoce gracias a una variable la cual marca a 1 los servicios

que no estén disponibles. Los perfiles de servicio marcados con un 0 no cambian. Por lo tanto únicamente se generarán combinaciones diferentes variando el/los perfiles marcados con un 1. Tener en cuenta que las combinaciones se generan, no sólo atendiendo a los perfiles que estén marcados a 1, sino también atendiendo al tamaño de bloque que se haya sido seleccionado, tal y como se hacía en el algoritmo de composición heurístico.

5) Seguidamente se comienza con el análisis de todas las combinaciones posibles, la cual se realizará a través de diversos bucles anidados.

    a. Primeramente se entra en un bucle, el cual se seguirá ejecutando siempre que haya bloques de combinaciones por evaluar y no se haya encontrado ninguna solución óptima, o no se haya sobrepasado el número de combinaciones totales que se pueden evaluar.

    b. Sigue con un bucle en el cual se analizarán todas las combinaciones posibles. Es importante tener en cuenta que se comprobará el estado en que quedarían los nodos y las redes tras la introducción de las tareas y mensajes correspondientes.

    c. Se resetean los *buffers*, y se inicializan algunas variables, para así conseguir que los nodos y redes estén en las mismas condiciones de utilización en la comprobación de cada combinación.

    d. Comienza otro bucle, el cual se seguirá ejecutando mientras exista algún perfil de servicio de la combinación sin analizar.

    e. A continuación se crea una diferencia entre las estructuras que contengan servicios en paralelo y serie. Dependiendo de si el perfil que se está analizando es parte de un paralelo o no, se realizarán un paso u otro.

        i. En caso de que el perfil no pertenezca a ningún paralelo se comprobará la planificabilidad de los nodos y de la red.

        ii. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

iii. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

iv. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

v. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

f. El tiempo de respuesta de la combinación que se esté analizando será la suma de los tiempos parciales.

g. Si todavía no se han visto todos los perfiles de la combinación se vuelve al *paso d)*.

h. Si el tiempo de respuesta final que se obtiene es menor que el último tiempo de respuesta mejor encontrado (el cual en la primera iteración será el *deadline*), y además tanto los nodos como las redes son planificables, entonces se guarda el tiempo de respuesta como el más óptimo. También se guarda la combinación que lo genera.

i. En caso de que no se obtenga ningún tiempo de respuesta mejor, o las redes no sean planificables, entonces se devuelve el valor del tiempo de respuesta igual a 5000, y la combinación óptima como una cadena vacía.

j. Si no se han visto todas las combinaciones se vuelve al *paso b)*.

k. Después de haber mirado todos los caminos posibles, si no se ha encontrado ninguna combinación óptima que consiga que el sistema sea planificable, y además, los nodos y las redes del sistema sí son planificables, entonces se procederá a aumentar el tamaño del bloque del primer nodo que se marcó como "no planificable".

l. Se comprobará que el tamaño del bloque no sobrepase el número total de los perfiles de servicio del nodo. En caso de que lo haga, se igualará el tamaño del bloque al valor máximo. Si el último valor cogido ya era el número máximo de perfiles de servicio posible, entonces querrá decir que

no existe ninguna combinación planificable. finalizará el algoritmo, vaciando la matriz generadora de combinaciones.

m. Si la matriz generadora de combinaciones no está vacía, es decir, todavía es posible encontrar alguna combinación posible para la planificación del sistema, se procederá a generar nuevas combinaciones posibles con los tamaños de bloque actualizados.

n. Se actualizan las variables, y si todavía hay combinaciones que se pueden estudiar, se volverá al paso *a)*.

6) Si después de analizar todas las combinaciones generadas todavía no se ha obtenido ninguna combinación planificable, entonces:

a. En caso de que el nodo puesto a 1 sea uno de los del paralelo, poner el resto de nodos del mismo paralelo a 1.

b. En caso de que haya un paralelo adyacente al nodo o nodos que se acaban de analizar, entonces se pondrán a 1 todos los nodos del paralelo.

c. En caso de que sólo haya nodos en serie, se pone a 1 los nodos adyacentes a los que se acaban de analizar.

Ampliando así el número de combinaciones posibles para encontrar una que sea planificable. Si el *time-out* no ha expirado, se vuelve a repetir todo el algoritmo desde *3)*.

7) Si por el contrario hay una solución *sub-óptima* que sí que es planificable entonces se sale del bucle que comenzaba en *3)*.

8) Una vez que se han visto todas las combinaciones o ha expirado el *time-out*, se comprueba si existe una combinación óptima. Si no existe se va al *paso 10)*.

9) Si existe alguna combinación óptima, entonces se procede a insertar la tarea en los nodos físicos y los mensajes correspondientes en las redes, para ello se realizarán los siguientes pasos:

a. Se comienza con un bucle el cual se ejecutará mientras queden perfiles sin introducir en los nodos físicos.

b. En caso de que el perfil no pertenezca a ningún paralelo se realizará la inserción de la tarea en el nodo físico correspondiente, y se introducirá también el mensaje en la red asociada, comprobando así si son planificables el nodo y la red.

c. Se actualizarán los tiempos parciales, teniendo en cuenta el tiempo de respuesta de las tareas y el tiempo que tardan en transmitirse los mensajes por la red.

d. En caso de que el perfil pertenezca a un paralelo, se comprobará la cantidad de ramas que contiene el mismo y se analizarán los tiempos parciales para cada perfil en una variable auxiliar.

e. Se insertarán las tareas del paralelo en los nodos físicos correspondientes, y también, se introducirán los mensajes en las redes asociadas.

f. Si las ramas del paralelo contienen más de un perfil, entonces se sigue analizando cada rama hasta que se llegue al siguiente servicio en serie.

g. El tiempo parcial del paralelo se corresponderá con la rama del mismo que obtenga un tiempo mayor.

h. Se comprueba que tanto los nodos como las redes son planificables. Si no es así, se pasa al *paso 10*).

i. Una vez introducida la combinación en el sistema, y siendo planificables tanto los nodos como las redes, se devuelven los tiempos de respuesta mínimos, el tiempo de ejecución del algoritmo, la combinación óptima y el número de combinaciones totales que se han examinado durante el análisis del algoritmo.

10) En caso de no planificabilidad, se devuelve la cadena vacía como combinación óptima y su tiempo de respuesta con una valor de 5000, para indicar que la aplicación no es planificable.

11) Para finalizar el algoritmo de reconfiguración mejorado heurístico, se devuelven: el tiempo de ejecución del mismo, el tiempo de respuesta mínimo, la primera combinación *sub-óptima* planificable que ha encontrado y el número de combinaciones totales que se han analizado.

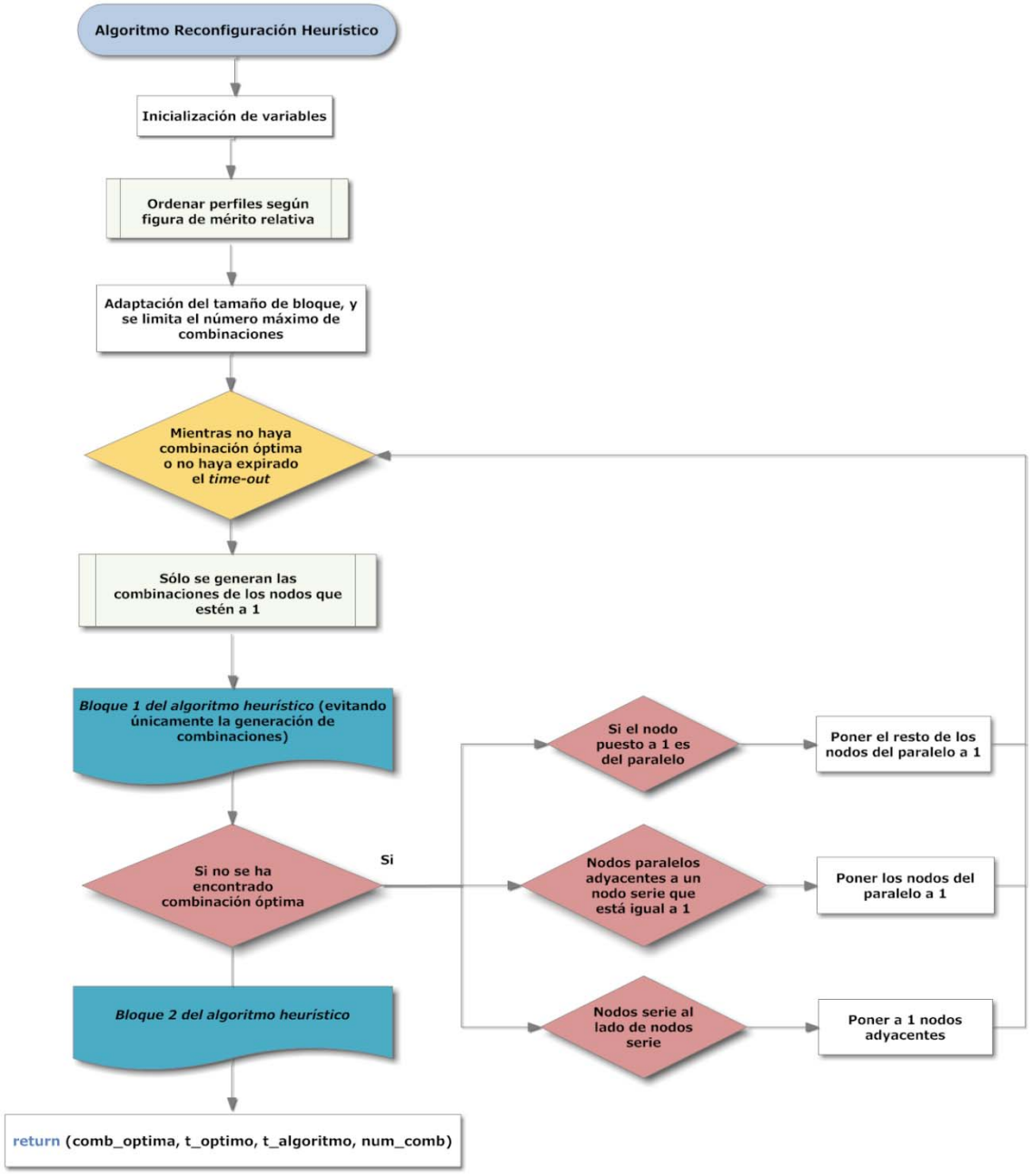


Figura 31. Algoritmo de Reconfiguración Heurístico

Como ya se ha comentado anteriormente, dicho algoritmo es muy parecido al algoritmo de composición heurístico, puesto que utiliza gran parte de su código, introduciendo las partes pertenecientes al algoritmo de reconfiguración de forma totalmente externa.

5.3. Conclusiones

Para dotar de flexibilidad a un sistema, éste tiene que tener la posibilidad de cambiar su configuración en tiempo real, para ello es necesario la implementación y desarrollo de algoritmos de reconfiguración que permitan dicha característica.

Para ello se han llevado a cabo dos desarrollos, ambos basados en los algoritmos de composición vistos en el capítulo anterior (algoritmo exhaustivo y algoritmo heurístico), pero que introducen la mejora de poder reconfigurar la aplicación una vez detectado que uno o varios perfiles que componen la aplicación han dejado de estar disponibles en el sistema.

Además, los algoritmos de reconfiguración se han diseñado de forma que sean totalmente transparentes al algoritmo de composición que se quiera utilizar, de forma que, por un lado, se analizan los perfiles no disponibles y se crean las combinaciones a analizar, y por otro, se aplica el algoritmo de composición (exhaustivo o heurístico) para comprobar si existe alguna combinación planificable o no. Son dos partes individuales.

En el *Capítulo 7* se analizarán los cuatro algoritmos implementados a lo largo de este proyecto (los de composición y los de reconfiguración) haciendo la función de reconfiguración del sistema. Como es de esperar, en principio los algoritmos de reconfiguración serán más rápidos, pero éste no es el único parámetro a tener en cuenta, puesto que la eficacia de un algoritmo de composición de aplicaciones de sistemas de tiempo real no se mide sólo por el tiempo que tarda en realizarse, sino que hay otros parámetros a tener en cuenta (como puede ser el error respecto de la figura de mérito que se esté asumiendo en el sistema).

Capítulo 6

Evaluación y Pruebas de los Algoritmos de Composición

En este capítulo se van a mostrar los resultados obtenidos tras la realización de una serie de pruebas que se han definido para analizar los algoritmos de composición implementados en este proyecto. En un principio, se describirán brevemente las consideraciones tomadas en cuanto a decisiones de diseño y que son comunes a todos los algoritmos, después se realizará una pequeña introducción de las pruebas que se han diseñado. Por último se documentará de forma exhaustiva los resultados obtenidos ayudándonos con la comparación entre las dos figuras de mérito implementadas para este proyecto. Todos los datos se reflejarán tanto en tablas como en gráficas.

6.1. Introducción

En este capítulo se ha diseñado una herramienta que permite obtener y comparar los resultados de los algoritmos de composición implementados en este proyecto, tanto el exhaustivo como el heurístico. Los cálculos de las pruebas se centrarán en la obtención de los tiempos de ejecución, junto con el número de combinaciones examinadas por los algoritmos y los errores respecto de la figura de mérito encontrados por los algoritmos de composición heurísticos implementados.

Tal como se dijo en la introducción, la implementación de los algoritmos, así como de la herramienta diseñada para la evaluación y pruebas de los mismos, han sido realizados utilizando el programa de software matemático *MATLAB*. Se ha elegido este programa por la facilidad que presenta a la hora de operar con matrices y la claridad en la representación de datos y funciones. Asimismo, y pensando en líneas futuras, también se eligió este programa debido a que ofrece la posibilidad de creación de interfaces gráficas de usuario, las cuales podrían formar parte de una posible ampliación del presente proyecto (*Apartado 8.2*).

6.1.1. Definición de los parámetros del sistema

En primer lugar, cabe destacar que el conjunto de pruebas realizadas para la evaluación de los algoritmos implementados en este proyecto quedará dividido en dos partes: una primera parte en la que se documentarán los resultados obtenidos para el algoritmo de composición (*Capítulo 6*), y una segunda parte en la que se documentarán los resultados obtenidos para los algoritmos de reconfiguración (*Capítulo 7*).

Las consideraciones que se han tomado en cuanto a los parámetros del sistema a la hora de diseñar el prototipo, o herramienta, han sido las mismas para todos los algoritmos, por este motivo la explicación se realizará en un apartado genérico.

Para todas las pruebas se ha considerado la utilización de tres nodos físicos (o tres procesadores) para incluir los servicios necesarios, y tres redes físicas. Éstas últimas serán denominadas como r1, r2 y r3.

Los servicios del sistema han sido definidos de forma aleatoria de la siguiente manera:

- El número de servicios incluidos en el sistema será definido de forma aleatoria[1], y tendrá un valor total entre 3 y 6 servicios.

- Cada servicio quedará definido por un número aleatorio de perfiles[2], los cuales irán entre 2 y 5 perfiles.

- Y, por último, cada perfil estará definido por dos valores que también se hallarán de forma aleatoria, el tiempo de computación de la tarea correspondiente (que

[1] En alguna prueba se forzará la cantidad de servicios, dejando así de ser aleatorio, aunque la programación se ha realizado para que sea posible trabajar de esta manera.
[2] En alguna prueba se forzará la cantidad de servicios, dejando así de ser aleatorio, aunque la programación se ha realizado para que sea posible trabajar de esta manera

tendrá un valor entre 1 y 5), y el nodo físico al que pertenece cada perfil (que tendrá un valor entre 1 y 3, correspondientes a los 3 nodos físicos fijados en un principio).

Asimismo la programación que se seguiría en *Matlab* para la definición de los servicios explicados antes sería parecida a lo siguiente:

*servicios(i)= struct ('nombre_serv', 'A', 'perfil', ((ceil(5*rand) ceil(3*rand)); (ceil(5*rand) ceil(3*rand); (ceil(5*rand) ceil(3*rand); (ceil(5*rand) ceil(3*rand)));*

Según se ha definido el servicio del ejemplo anterior, se obtendría la definición de un servicio 'A', constituido por 4 perfiles, los cuales obtienen los valores del tiempo de computación y del nodo físico de forma aleatoria.

Tal como se ha comentado anteriormente, tanto la cantidad de servicios como de perfiles no ha sido del todo variable a lo largo de las pruebas. Esto se ha realizado así para forzar la composición de los elementos del sistema con el fin de poder realizar un estudio más exacto de los resultados.

Todos los experimentos diseñados a lo largo de este proyecto han sido realizados un mínimo de 10 veces cada uno. La decisión de realizar cada experimento un número mínimo de veces ha sido tomada tras considerar en el diseño del prototipo que algunas de las variables introducidas fueran aleatorias, ya que de este modo se podría conseguir una mayor similitud con los sistemas de tiempo real. Con esa decisión no se podía considerar que los resultados hallados tras un único procesamiento fuera exacto, por lo que, los parámetros que sean susceptibles de cambios ante las variables aleatorias introducidas en el sistema vendrán definidos tanto por su media como por su varianza (indicando esta última la dispersión de los valores).

De este modo, para hallar la media y la varianza se han seguido las expresiones matemáticas que se indican siguientes:

$$media = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$varianza = \frac{\sum_{i=1}^{n} (x_i - media)^2}{n}$$

A continuación se comenzará con la evaluación y pruebas de los algoritmos de composición, realizando lo mismo para los algoritmos de reconfiguración en el *Capítulo 7*.

6.2. Estudio de los Algoritmos de Composición

En esta parte se evaluarán los algoritmos de composición. A la vez ser realizará un estudio comparativo entre las dos figuras de mérito implementadas y explicadas en el *Capítulo 4*. Asimismo se estudiarán las prestaciones ofrecidas por dichos algoritmos.

Se analizarán los tiempos de ejecución de los algoritmos y la cantidad de combinaciones que necesitan examinar cada uno de ellos, así como el error introducido por el algoritmo heurístico al hallar una solución *sub-óptima*.

Para probar los algoritmos se ha diseñado una herramienta analítica que permite la ejecución de todos los algoritmos bajo las mismas condiciones del sistema. Esto quiere que teniendo varios algoritmos definidos en la herramienta, se vayan ejecutando uno detrás de otro, pero reestructurando tanto las redes como los nodos a los valores iniciales que se tenían cuando comenzó la iteración. Esto quiere decir que, por ejemplo, en la primera iteración del sistema (en la que se consideró que tanto los nodos como las redes partieran de un estado inicial de *vacío*), cada vez que se analice un algoritmo, tanto los nodos como las redes tendrán el estado inicial de *vacío*. Del mismo modo, si ya se hubieran producido varias iteraciones, el estado de partida de las redes y los nodos sería el mismo para todos los algoritmos. De esta forma se consigue, como se comentó al principio, hacer una comparación más o menos exacta de cómo se comportan unos algoritmos u otros bajo las mismas condiciones del sistema.

Una vez comentado esto, es importante destacar que tanto las redes como los nodos se irán llenando de mensajes o tareas hasta que la aplicación deje de ser planificable.

Los resultados obtenidos tras la realización de las pruebas sobre los algoritmos implementados, se apoyarán sobre tablas y gráficos, de forma que su visualización permita un entendimiento mayor de los mismos.

6.2.1. Algoritmos de composición.

A lo largo de este apartado se realizará un estudio entre los dos algoritmos implementados en este proyecto, con el fin de poder encontrar el algoritmo más óptimo y que dé una mejor calidad. Se tendrá en cuenta si el error que produce el algoritmo heurístico es suficientemente pequeño respecto al tiempo que tarda en ejecutarse. Además se compararán los algoritmos en función de las combinaciones seleccionadas como más óptimas y el tiempo que tarda en encontrarlas.

Para ello se realizarán diferentes pruebas, y se compararán los tiempos de ejecución obtenidos entre los algoritmos heurísticos (introduciendo diferentes tamaños de bloque) y los tiempos de ejecución del algoritmo exhaustivo, que, como ya se ha explicado en capítulos anteriores, es el que genera la combinación más óptima, pero también es el que más tarda en ejecutarse. Por lo tanto, se comprobará qué algoritmo será el más óptimo en función del error cometido respecto de la figura de mérito global y el tiempo de ejecución del propio algoritmo.

De esta manera, se realizarán dos pruebas en este apartado:

- Prueba 1.1: Se analizarán los tiempos de ejecución y error cometido por el algoritmo heurístico variando el número de servicios a analizar, pero no la cantidad de perfiles de los mismos, los cuales se fijarán a 4.

- Prueba 1.2: se analizarán las combinaciones óptimas generadas por cada uno de los algoritmos, así como su tiempo de ejecución y la figura de mérito utilizada en cada algoritmo heurístico. En esta ocasión se variará tanto el número de servicios como el número de perfiles de los servicios en juego.

6.2.1.1. Prueba 1.1. Resultados variando el número de servicios.

En esta primera prueba se pretende mostrar la relación existente entre el número de combinaciones analizadas y el tiempo de ejecución medio de los algoritmos, a la vez de ver el error medio cometido por el algoritmo heurístico en función del tamaño de bloque que se haya elegido.

Los *bloques* en el algoritmo heurístico, tal y como se explicó en el *Capítulo 4*, se definen para limitar el número de combinaciones que se deben analizar por el mismo. Por lo que, antes de ejecutarse el algoritmo, los perfiles de servicio son ordenados siguiendo una figura de mérito relativa, de tal forma que si se escoge un tamaño de bloque igual a 1, sólo se analizan las combinaciones relacionadas con el mejor perfil, es decir, el primero de la lista ordenada de perfiles, si se escoge un tamaño de bloque 2, se analizan las combinaciones relacionadas con los dos mejores en función de la figura de mérito relativa, y así sucesivamente dependiendo del tamaño de bloque seleccionado.

Utilizando el método de los *bloques* en el algoritmo heurístico, los tiempos de ejecución respecto del algoritmo exhaustivo son mejorados sustancialmente, pero ello conlleva el tener que asumir un error respecto de la solución más óptima, puesto que lo que se consigue con ese algoritmo es una solución *sub-óptima*. Por lo tanto, a la hora de elegir el

tamaño de bloque del algoritmo heurístico, no sólo se debe mirar el tiempo de ejecución del mismo (algo bastante importante en sistemas de tiempo real), sino que también se debe de tener en cuenta el error que se está cometiendo.

Cuando se habla del error, se habla de la dispersión del valor del tiempo de respuesta total que se obtiene con el algoritmo heurístico respecto de la obtenida por el algoritmo exhaustivo. El error se mide en función de la diferencia del tiempo de respuesta de la combinación analizada respecto de la óptima conseguida mediante el algoritmo exhaustivo. En las tablas se mostrará el error en valores del tanto por ciento [%].

A continuación se muestran los resultados referentes a los tiempos de ejecución de los algoritmos y al error que se debe asumir en caso de que, al utilizar el algoritmo heurístico, éste se utilice con un tamaño de bloque inapropiado que provoque una situación en la que el tiempo de ejecución sea bastante bajo pero el error cometido respecto de la figura mérito global sea demasiado grande como para poder ser asumido como el algoritmo más viable.

Para la realización de esta prueba se ha elegido una aplicación sencilla, la cual únicamente está compuesta por nodos serie. Se han realizado 7 experimentos, repitiéndolos una media de 10 veces como mínimo cada uno. Por este motivo se pueden ver en las tablas que se muestran a continuación, como cada parámetro viene definido por un valor medio y una varianza.

Los experimentos se han realizado aumentando el número de nodos en la aplicación, ya que ellos tienen una relación directa con el número de combinaciones que se deben analizar. El primer experimento consiste en una aplicación con 2 nodos en serie únicamente, el siguiente con 3, y así sucesivamente hasta tener 8 nodos en serie en el último experimento.

Hay que tener claro, que todos los resultados obtenidos han sido generados en función de un número de perfiles por servicio predeterminado e igual a cuatro, por lo que es normal que con un tamaño de bloque igual 3 el error producido sea prácticamente nulo. Pero todos estos resultados son susceptibles de cambio si se producen modificaciones en los parámetros del sistema.

Los resultados se mostrarán en tablas. Éstas están compuestas por tres partes diferenciadas: el número de nodos introducidos en la aplicación, los resultados obtenidos del algoritmo exhaustivo (número de combinaciones y tiempo de ejecución) y los resultados obtenidos para el algoritmo heurístico (tiempo de ejecución y error respecto del

tiempo de respuesta). Se han realizado tres tablas por figura de mérito utilizada, divididas en función del tamaño de bloque que se haya escogido para la ejecución del algoritmo heurístico. Los tamaños de bloques que se han considerado han sido, tamaños de bloque igual a 1, 2 y 3. El hecho de introducir en la misma tabla los datos del algoritmo exhaustivo con los de los algoritmos heurístico hace posible una comparación de datos más eficaz, puesto que el algoritmo exhaustivo se tomará como línea de partida para sacar las conclusiones oportunas acerca de qué algoritmo heurístico es más favorable en función de los parámetros sacados.

Cómo se puede apreciar en las tablas de la prueba 1.1 únicamente se ha tenido en cuenta el número de combinaciones que han sido analizadas por el algoritmo exhaustivo, puesto que este parámetro se tomará como variable independiente a la hora de realizar las gráficas, y ahí es donde se verá realmente como el hecho de analizar cada vez menos combinaciones puede introducir mejoras realmente apreciables en tiempos de ejecución. Las gráficas en las que se mostrarán los resultados del error medio de cada algoritmo también tendrán como variable independiente el número de combinaciones del algoritmo exhaustivo, y asimismo se verá como el error aumenta en función del algoritmo que menos combinaciones analiza.

Como es de suponer, cuanto más pequeño es el tamaño de bloque utilizado por el algoritmo heurístico, mayor es el error cometido. Es totalmente lógico suponer este hecho, puesto que cuantas menos combinaciones se analicen, menos posibilidades se tienen de dar con la óptima, y además, más posibilidades existen también de que la combinación *sub-óptima* encontrada no sea la más idónea en cuanto a la cantidad de error que debería asumir el sistema.

La prueba 1.1 concluye analizando tanto las gráficas dedicadas a los tiempos de ejecución, como las dedicadas a las medidas del error acumulado, en donde se explicará dentro de los casos estudiados qué algoritmo sería el más adecuado en función de los parámetros estudiados, puesto que habrá situaciones en donde premiará la calidad de la aplicación, en cuyo caso se elegirá el algoritmo que menor error muestre, y habrá situaciones en donde lo que premiará será el tiempo que tarda en realizar la operación, en cuyo bastará con que el error sea lo suficientemente pequeño.

Para una presentación de los resultados más ordenada, se dividirán los datos en función de la figura de mérito relativa que se haya utilizado en la ordenación de perfiles de servicio en el algoritmo heurístico.

Figura de Mérito 1: *Minimización del tiempo de ejecución en el peor* caso.

En esta primera parte se ha utilizado la figura de mérito relativa que se explicó en el apartado 4.5.2: *"Minimización del tiempo de ejecución en el peor caso"*, la cual se basa en la relación entre el tiempo de ejecución en el peor caso de cada implementación de servicio y el plazo deseado por la aplicación:

$$\mathcal{F}_1(p) = \frac{C_p}{D_{aplicación}}$$

Para esta figura de mérito se han obtenido los resultados que se muestran en Tabla 1, Tabla 2 y Tabla 3.

Serv. Serie	Exhaustivo			Heurístico (Tamaño de bloque = 1)			
	Número de comb.	Tiempo de ejecución		Tiempo de ejecución		Error [%]	
		Media	Var.	Media	Var.	Media	Var.
2	16	0,0589	9,09E-06	0,0122	1,08E-05	1,3378	0,1092
3	64	0,249	2,02E-04	0,0167	8,89E-06	1,8519	6,77E-03
4	256	1,2091	0,005	0,0326	1,90E-03	1,9879	6,83E-04
5	1024	5,6069	0,3574	0,0215	1,16E-05	2,217	0,0281
6	4096	25,3082	9,6774	0,0204	1,33E-05	3,2088	0,1009
7	16384	146,4977	27,2742	0,0427	3,00E-03	3,9952	0,1571
8	65536	1808,7	43,99	0,0564	4,50E-01	4,5678	0,331

Tabla 1. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 1 (Bloq=1).

Serv. Serie	Exhaustivo			Heurístico (Tamaño de bloque = 2)			
	Número de comb.	Tiempo de ejecución		Tiempo de ejecución		Error [%]	
		Media	Var.	Media	Var.	Media	Var.
2	16	0,0589	9,09E-06	0,021	1,73E-05	0	0
3	64	0,249	2,02E-04	0,045	1,92E-05	0	0
4	256	1,2091	0,005	0,1166	2,00E-03	0,2604	6,6576E-03
5	1024	5,6069	0,3574	0,2235	9,27E-04	0,482	2,7343E-03
6	4096	25,3082	9,6774	0,4696	1,90E-03	1,4860	0,0652
7	16384	146,4977	27,2742	1,1212	2,94E-02	1,82	0,0325
8	65536	1808,7	43,99	1,4592	1,25E-02	2,4472	0,111

Tabla 2. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 1 (Bloq=2).

Serv. Serie	Exhaustivo			Heurístico (Tamaño de bloque = 3)			
	Número de comb.	Tiempo de ejecución		Tiempo de ejecución		Error [%]	
		Media	Var.	Media	Var.	Media	Var.
2	16	0,0589	9,09E-06	0,0361	1,13E-05	0	0
3	64	0,249	2,02E-04	0,1266	6,54E-05	0	0
4	256	1,2091	0,005	0,4996	4,10E-03	0	0
5	1024	5,6069	0,3574	1,6341	0,0527	0	0
6	4096	25,3082	9,6774	5,3015	0,2199	0	0
7	16384	146,4977	27,2742	18,9463	6,0694	0	0
8	65536	1808,7	43,99	45,647	25,3622	0	0

Tabla 3. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 1 (Bloq=3).

Además, a continuación se muestran dos gráficas que reflejan los datos expresados en la tabla anterior.

En la Figura 32 se puede observar la relación que hay entre los algoritmos estudiados y el tiempo de ejecución en media que tarda cada uno de ellos para los experimentos indicados.

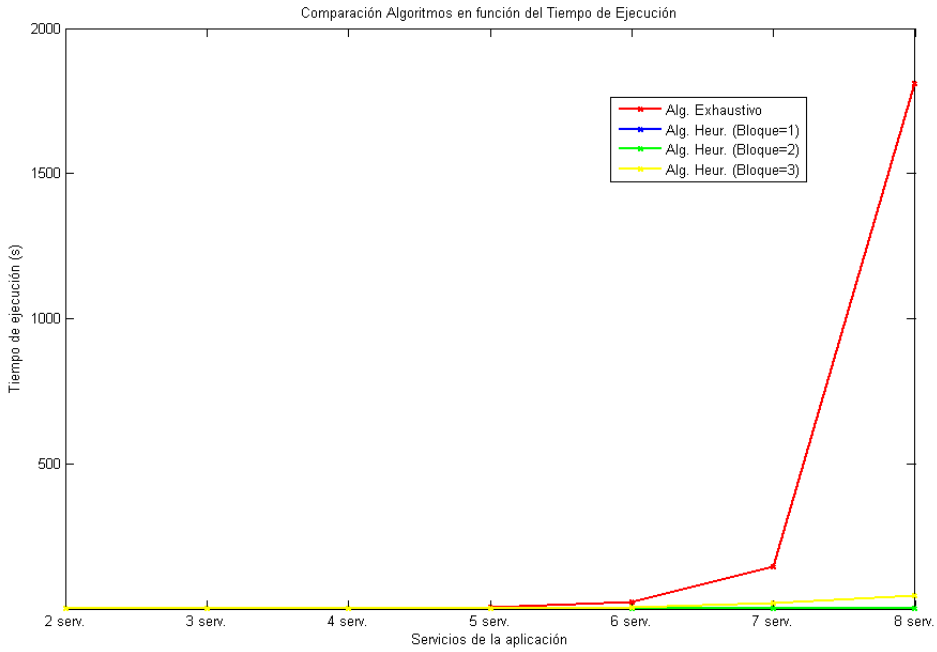


Figura 32. Comparación Algoritmos en función del Tiempo de Ejecución (Fig. Mer. 1)

En la Figura 33 se muestra una ampliación de la parte inferior derecha de la Figura 32, puesto que no se aprecia la trayectoria que sigue la línea perteneciente al algoritmo heurístico con tamaño de bloque igual a 1.

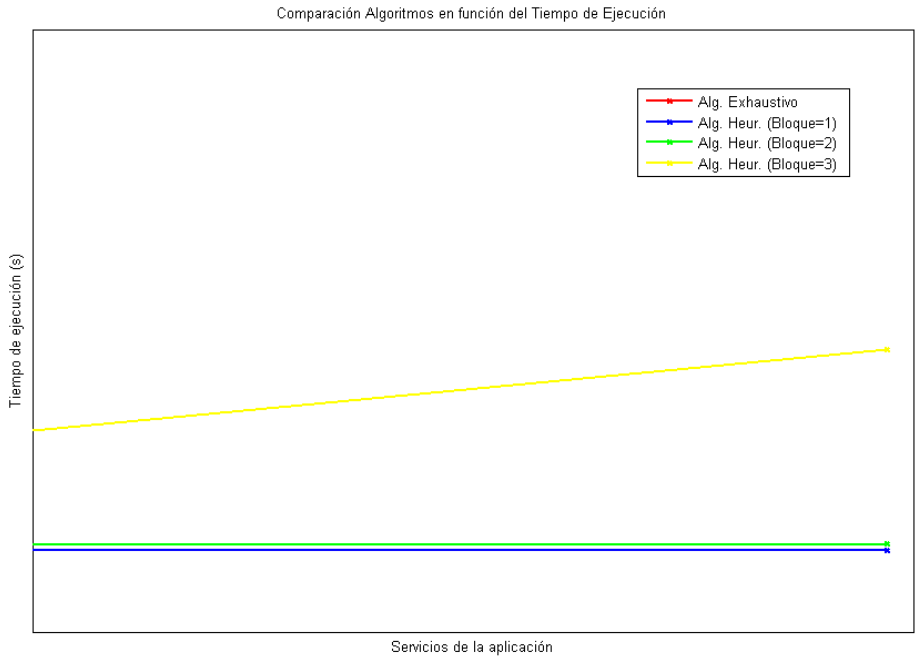


Figura 33. Zoom-Comparación Algoritmos en función del Tiempo de Ejecución (Fig. Mer. 1)

En la Figura 34 se puede observar la relación que hay entre los algoritmos estudiados y la media del error respecto de la figura de mérito global. Como es de suponer, el algoritmo exhaustivo no introduce error, puesto que siempre encuentra la solución óptima.

En dicha figura se puede ver como el error introducido por el algoritmo heurístico con un tamaño de bloque igual a tres es nula, por lo que habría que estudiar si es viable el tiempo que tarda en ejecutarse este algoritmo en una aplicación de un sistema de tiempo real. Es importante significar que la cantidad de error dependerá en gran medida del número de perfiles que contengan los servicios y el tamaño de bloque seleccionado por el algoritmo heurístico. Por ese motivo en esta prueba, el error para un tamaño de bloque igual a tres es mínimo, puesto que el número de perfiles por servicios se ha forzado a cuatro.

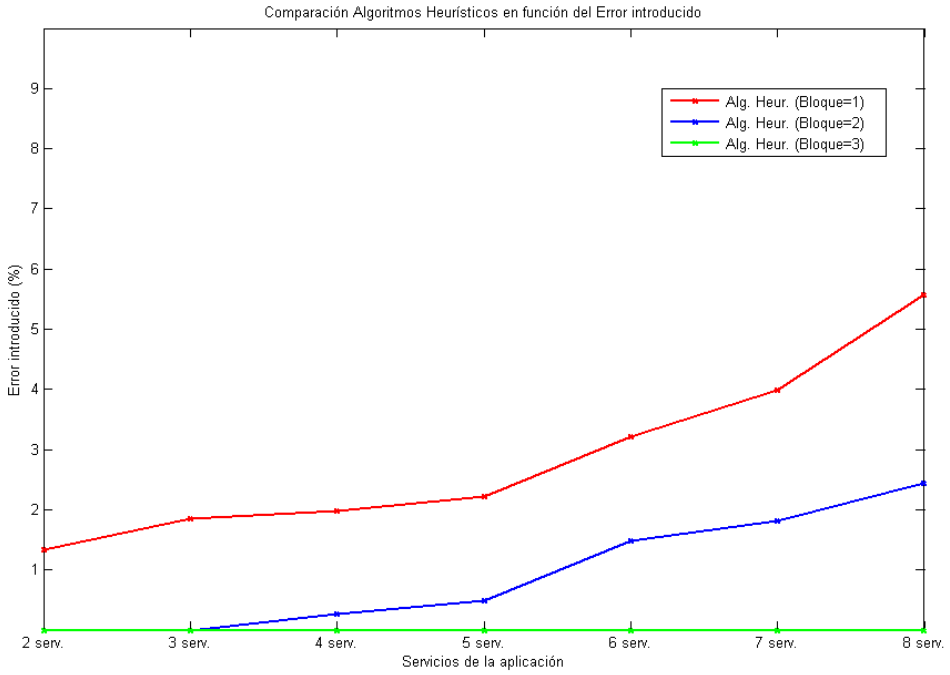


Figura 34. Comparación Alg. Heurísticos en función del Error respecto de la Figura de Mérito Global (Fig. Mer. Relativa 1)

Respecto al resto de los resultados, se puede observar como siempre es mayor el error introducido por los algoritmos heurísticos con un tamaño de bloque menor. Por ejemplo para un tamaño de bloque igual a 1 el tiempo que tarda en ejecutarse es totalmente despreciable, por lo que atendiendo al porcentaje de error que se pueda asumir en un sistema, puede resultar como una opción que no esté fuera de descarte.

Figura de Mérito 2: *Maximización del tiempo restante antes del Deadline*.

En este caso, la figura de mérito relativa utilizada es la que se explicó en el *Apartado 4.5.2*: *"Maximización del tiempo restante antes del deadline"* la cual ha sido implementada íntegramente en este proyecto. Como ya comentó, ésta se halla mediante la siguiente expresión:

$$\mathcal{F}_2(p) = \frac{D_{aplicación} - \left(\sum_{i=1}^{n} C_i^{min}\right) - \left(\sum_{i=1}^{n-1} C_i^{mensajes}\right)}{n} - C_p$$

En esta figura de mérito relativa las implementaciones de servicio se ordenan de mayor a menor en función del tiempo sobrante antes del *deadline*. Puesto que cuanto mayor sea este tiempo menos tarda en ejecutarse.

Para esta figura de mérito se han obtenido los resultados que se muestran en la Tabla 4, Tabla 5 y Tabla 6.

Serv. Serie	**Exhaustivo**			**Heurístico (Tamaño de bloque = 1)**			
	Número de comb.	**Tiempo de ejecución**		**Tiempo de ejecución**		**Error [%]**	
		Media	**Var.**	**Media**	**Var.**	**Media**	**Var.**
2	16	0,0532	1,1445E-05	0,0132	1,0759E-05	5,0191	0,2469
3	64	0,2547	4,2502E-04	0,0149	2,1182E-05	2,9274	0,08399
4	256	1,1361	0,0079	0,0177	1,5528E-05	4,2147	0,17409
5	1024	5,6576	0,6937	0,0204	1,2075E-05	6,2396	0,3815
6	4096	33,0266	2,7869	0,0252	1,9473E-05	5,6498	0,3128
7	16384	158,5492	6,9940	0,0273	3,5421E-05	4,1007	0,1256
8	65536	1165,4	10,89	0,0282	2,681E-03	8,5147	0,4366

Tabla 4. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 2 (Bloq=1).

Serv. Serie	**Exhaustivo**			**Heurístico (Tamaño de bloque = 2)**			
	Número de comb.	**Tiempo de ejecución**		**Tiempo de ejecución**		**Error [%]**	
		Media	**Var.**	**Media**	**Var.**	**Media**	**Var.**
2	16	0,0532	1,1445E-05	0,0206	1,2582E-05	2,0191	0,2469
3	64	0,2547	4,2502E-04	0,0455	2,2912E-05	1,8809	0,034674
4	256	1,1361	0,0079	0,0969	1,1663E-04	2,2615	0,05127
5	1024	5,6576	0,6937	0,2212	9,5953E-04	2,5988	0,0661
6	4096	33,0266	2,7869	0,6779	0,0067	2,1451	0,04513
7	16384	158,5492	6,9940	1,2808	0,0202	2.2128	0,05125
8	65536	11654	10,89	2,1178	0,0185	4,238	0,2864

Tabla 5. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 2 (Bloq=2).

Serv. Serie	Exhaustivo			Heurístico (Tamaño de bloque = 3)			
	Número de comb.	Tiempo de ejecución		Tiempo de ejecución		Error [%]	
		Media	Var.	Media	Var.	Media	Var.
2	16	0,0532	1,1445E-05	0,0337	7.5267E-06	0	0
3	64	0,2547	4,2502E-04	0,1253	1,1466E-04	0	0
4	256	1,1361	0,0079	0,4531	0,0034	0	0
5	1024	5,6576	0,6937	1,5974	0,05	0	0
6	4096	33,0266	2,7869	7,7491	0,8440	0	0
7	16384	158,5492	6,9940	22,1074	5,7573	0,3208	1,054E-03
8	65536	11654	10,89	55,4177	1,596	0,4569	2,896E-03

Tabla 6. Prueba 1.1. Variando el núm. de servicios. Fig. de Mér. 2 (Bloq=3).

Al igual que para la figura de mérito 1, a continuación se muestran las gráficas que reflejan los datos expresados en la tabla anterior.

En la Figura 35 se puede observar la relación que hay entre los algoritmos estudiados y el tiempo de ejecución en media que tarda cada uno de ellos para los experimentos indicados.

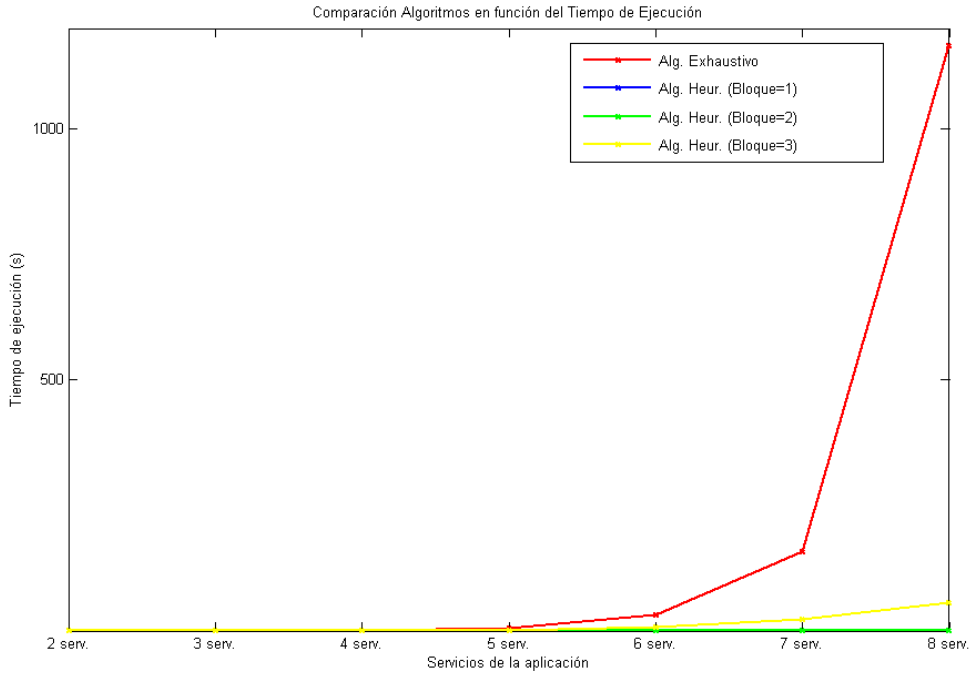


Figura 35. Comparación Algoritmos en función del Tiempo de Ejecución (Fig. Mer. 2)

En la Figura 36 se muestra una ampliación de la parte inferior derecha de la gráfica anterior, puesto que en la otra no se aprecia la trayectoria que sigue la línea perteneciente al algoritmo heurístico con tamaño de bloque igual a 1.

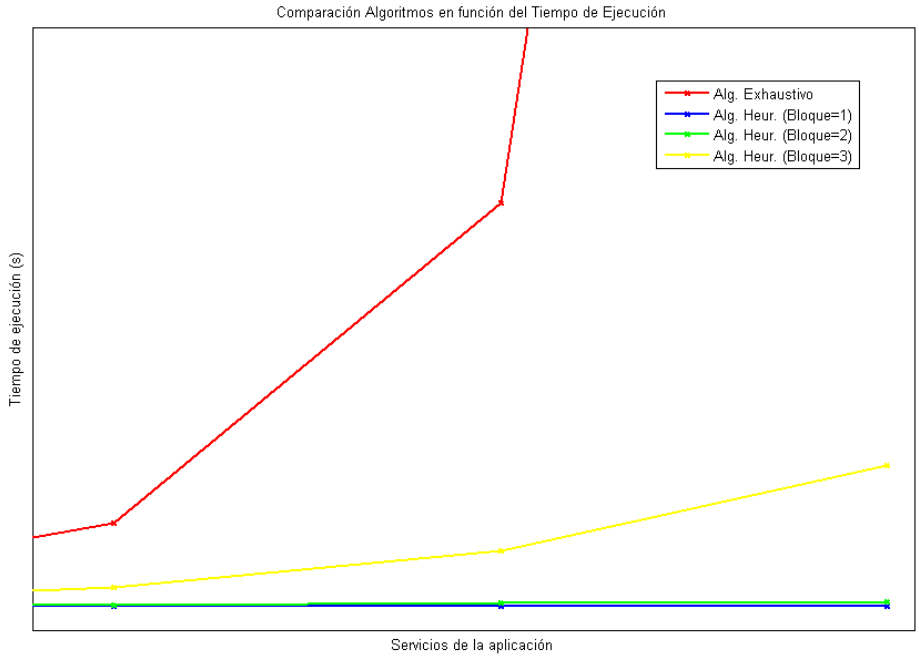


Figura 36. Zoom-Comparación Algoritmos en función del Tiempo de Ejecución (Fig. Mer. 2)

En la Figura 37 se puede observar la relación que hay entre los algoritmos estudiados y la media del error respecto de la figura de mérito global, asumiendo que el algoritmo exhaustivo no introduce error, puesto que es el que encuentra la solución óptima. En dicha figura se puede apreciar que la media del error de los algoritmos heurísticos con Figura de Mérito 2 sigue las mismas bases que las obtenidas con la Figura de Mérito 1. Teniendo más error los algoritmos que utilizan un tamaño de bloque menor, y el cual disminuye a medida que éste aumenta y se empiezan a analizar más combinaciones.

Además, si se quisiera asegurar una combinación totalmente óptima, en caso de que así se solicitara en la aplicación, la única alternativa posible sería la de elegir el algoritmo exhaustivo.

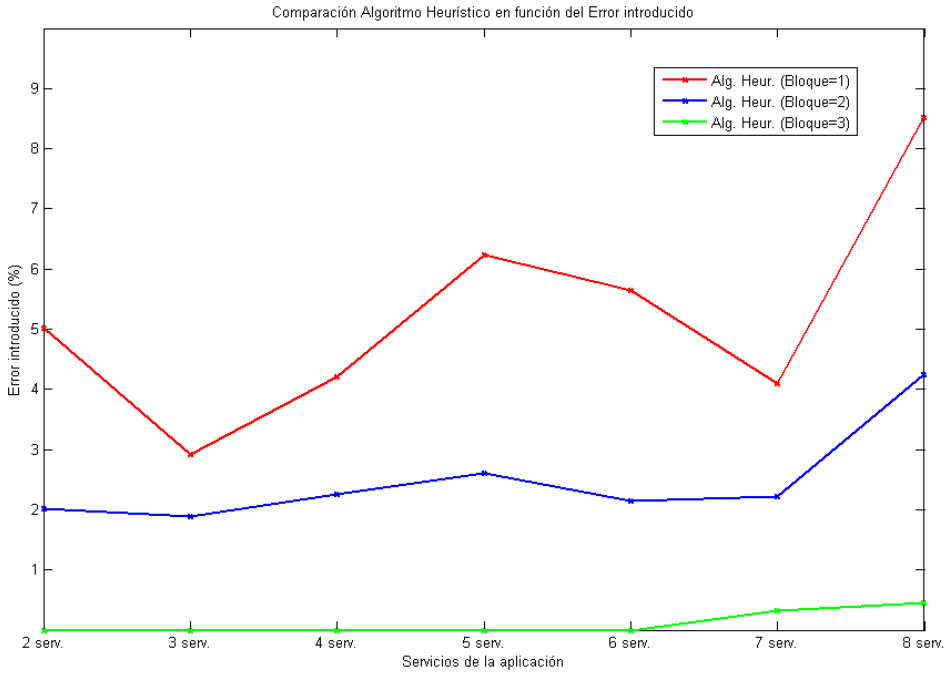


Figura 37. Comparación Alg. Heurísticos en función del Error respecto de la Figura de Merito Global (Fig. Mer. Relativa 2)

Conclusiones

Como se puede observar en los resultados obtenidos, el tiempo de ejecución va disminuyendo en función del algoritmo que analiza menos combinaciones así, el algoritmo heurístico con tamaño de bloque igual a 1 es el más rápido, a la vez que también es este último el que presenta mayor error, descartándole (casi seguro) como algoritmo más viable a la hora de tener que utilizarlo en un sistema de tiempo real.

No pasa lo mismo con el algoritmo heurístico que tiene un tamaño de bloque igual a 2, puesto que, el tiempo de ejecución se reduce considerablemente, y su error puede llegar a ser considerado como pequeño o irrelevante en algún tipo de aplicación, por lo que es una buena opción a tener en cuenta en sistemas en los que el margen de error sea alto.

Para el caso del algoritmo heurístico con tamaño de bloque igual a 3 no pasa lo mismo que en los anteriores, puesto que en caso de que la aplicación sea compleja el tiempo de ejecución es bastante mayor que el que obtenemos para sus dos hermanos (tamaños de bloque 1 y 2), pero sin embargo el error que se comete es prácticamente nulo. Por lo que para aplicaciones en las que pueda ser válido que el error cometido no sea nulo sí que podría considerarse como el más válido. Además, si se compara el tiempo de ejecución de

este algoritmo con el algoritmo exhaustivo, es claramente visible como, para un experimento de 8 nodos en serie, la mejoría es de hasta dos órdenes de magnitud. Como ya se comentó anteriormente, esta es la conclusión obtenida asignando cuatro perfiles por servicio; la propuesta de aumentar el número de perfiles, para una aplicación con un número de servicios ya determinada se realizará en la prueba 1.2 la cual se verá a continuación.

Por lo tanto, a la hora de elegir un algoritmo en el que sea necesario una media de error prácticamente nula (para el caso que nos ocupa), se tendrá la opción de escoger, o bien el algoritmo exhaustivo, en el cual se sabe que la solución encontrada siempre será la óptima, o bien, escoger cualquiera de los algoritmos heurísticos con un tamaño de bloque igual a 3.

Además, en cuanto a las gráficas de las medias del error respecto de la figura de mérito global, se puede ver como a mayor número de combinaciones, menor es el error obtenido, y cuantas menos combinaciones se examinan, mayor es el error.

Queda claro que la eficacia del tamaño de los bloques utilizado en el algoritmo heurístico está altamente relacionada con el número de perfiles que tenga el servicio, puesto que no es lo mismo elegir el tamaño de bloque para un servicio que tenga 4 servicios que para uno que tenga 8. En esta prueba en concreto se fijó el número de perfiles para los servicios utilizados a cuatro, por lo que al elegir un tamaño de bloque igual a 3, es altamente probable que se elija la combinación más óptima, y encima se reduzca el tiempo de ejecución considerablemente.

Tal como ya se comentó, todas las conclusiones obtenidas anteriormente son susceptibles de cambio, siempre que se realicen modificaciones en la estructura de la aplicación, en la dispersión de los valores del tiempo de ejecución en el peor caso del conjunto de perfiles de cada servicio, en el estado del sistema o en los parámetros de la propia red.

Respecto a la comparación entre las dos figuras de mérito implementadas, se puede observar como no hay diferencias significativas entre ambas, considerando que ambas proporcionan una eficiencia similar.

Tal como se indicó al principio de esta prueba, en esta ocasión se han basado todos los experimentos en servicios con 4 perfiles, variando únicamente la cantidad de servicios que componen la aplicación.

6.2.1.2. Prueba 1.2. Resultados variando el número de perfiles.

En esta segunda prueba lo que se hará es variar tanto el número de implementaciones de servicio, como el número de servicios que componen la aplicación, de tal forma que encontrar la solución más óptima sea más difícil para los algoritmos heurísticos, puesto que lo que sí se fijará será el tamaño de bloque a utilizar por cada uno de ellos.

Por lo tanto, para esta prueba se analizarán el algoritmo exhaustivo, el algoritmo heurístico con la figura de mérito relativa *"Minimización del tiempo de ejecución en el peor caso"* (a partir de ahora llamada figura de mérito 1), y el algoritmo heurístico con la figura de mérito relativa *"Maximización del tiempo restante antes del deadline"* (a partir de ahora llamada figura de mérito 2). Considerando para el algoritmo heurístico de figura de mérito 1 un tamaño de bloque igual a 3, y para el algoritmo heurístico de figura de mérito 2 un tamaño de bloque igual a 2.

Como la eficiencia de ambas figuras de mérito son parecidas, se realizará el estudio para diferentes tamaños de bloque en cada una. Y además se podrá ver la eficiencia de los tamaños de bloques variando el número de perfiles de los servicios, y cómo el error que introducen puede ser mínimo.

Al igual que en la prueba anterior, se ha escogido una aplicación que únicamente conste de servicios en serie. Aunque en esta ocasión la diferencia entre un experimento y otro no tiene por qué ser sólo la introducción en la aplicación de un nuevo servicio, puesto que también se introducirán nuevos perfiles de servicio. Con esto se pretende demostrar la validez del algoritmo heurístico con tamaño de bloque igual a 3 ante aplicaciones con mayor número de perfiles en sus servicios. Puesto que, como se explicó en la prueba anterior, no es lo mismo seleccionar 3 perfiles de un total de 4, que 3 de un total de 8, ya que la probabilidad de elegir el perfil óptimo es mucho menor en este último caso.

Asimismo, se realizarán también varios experimentos, igual que se hizo en la anterior prueba, y se irán incrementando el número de servicios y/o perfiles de los mismos para ir incrementando el número de combinaciones analizables, de forma que los experimentos estarán distribuidos en la tabla de menor a mayor número de servicios, y de menor a mayor número de perfiles.

En la Tabla 7, Tabla 8 y Tabla 9 se pueden ver los resultados obtenidos en esta prueba. Las tablas quedarán divididas de la siguiente manera: en el lado derecho (y siendo común en todas) se verá el experimento que se está realizando (número de perfiles y número de servicios de la aplicación), las siguientes columnas indican el número de combinaciones

analizadas y el tiempo de ejecución de los algoritmos (siendo éstas comunes también a todas las tablas), y por último, en las tablas donde se muestran los valores de los algoritmos heurísticos, se muestra la columna del error respecto de la figura de mérito global.

En esta ocasión, se utilizarán las conclusiones obtenidas en la prueba anterior para no volver a repetir los mismos datos que los hallados anteriormente.

Por esta razón, el algoritmo heurístico con figura de mérito 1 elegido en esta ocasión será el de tamaño de bloque igual a 3 (puesto que, aunque tardaba un poco más en ejecutarse, el error que se obtenía con él era totalmente nulo), y el algoritmo heurístico con figura de mérito 2 será el de tamaño de bloque igual a 2 (puesto que así se puede ver la diferencia en tiempos de ejecución y error entre el algoritmo heurístico con figura de mérito 1 y éste).

En la Tabla 7 se muestran los datos relativos al algoritmo exhaustivo, mientras que en la Tabla 8 y Tabla 9 se muestran los datos relativos a los algoritmos heurísticos, primero el que tiene figura de mérito 1, y segundo el que tiene figura de mérito 2.

Núm. de Nodos	**Perfiles por Servicio**	**Algoritmo Exhaustivo**		
		Núm de Comb.	**Tiempo de ejecución**	
			Media	**Var.**
4	**3**	81	0,4015	4,6497E-04
4	**4**	256	1,1933	0,0107
4	**5**	625	2,7187	0,0676
4	**6**	1296	5,8203	0,5629
5	**4**	1024	5,5186	0,2988
5	**5**	3125	17,3051	3,1332
5	**6**	7776	43,9031	1,7352
6	**4**	4096	26,3756	16,4667
6	**5**	15625	125,7948	32,9614
6	**6**	46656	593,4771	227,64

Tabla 7. Prueba 1.2. Variando el núm. de perfiles de servicio (Alg. Exh.).

Núm. de Servicios	**Perfiles por Servicio**	**Algoritmo Heurístico (Fig. mérito 1) (Tamaño bloque = 3)**				
		Núm de Comb.	**Tiempo de ejecución**		**Error [%]**	
			Media	**Var.**	**Media**	**Var.**
4	**3**	81	0,4918	0,0018	0	0
4	**4**	81	0,4693	0,0041	0	0
4	**5**	81	0,4801	0,0035	0	0
4	**6**	81	0,4623	0,0038	0	0
5	**4**	243	1,7167	0,0109	0	0
5	**5**	243	1,6661	0,0414	0	0
5	**6**	243	1,5950	0,1145	0	0
6	**4**	729	5,7113	1,4119	0	0
6	**5**	729	5,9614	1,1115	0	0
6	**6**	729	5,9939	0,4066	0	0

Tabla 8. Prueba 1.2. Variando el núm. de perfiles de servicio (Fig. Mer. 1).

Núm. de Servicios	**Perfiles por Servicio**	**Algoritmo Heurístico (Fig. mérito 2) (Tamaño bloque = 2)**				
		Núm de Comb.	**Tiempo de ejecución**		**Error [%]**	
			Media	**Var.**	**Media**	**Var.**
4	**3**	16	0,1068	1,3941E-04	0,2161	2,5597E-02
4	**4**	16	0,1007	1,2504E-04	0,1931	3,65E-04
4	**5**	16	0,1049	1,5105E-04	0,2121	4,41E-04
4	**6**	16	0,1011	1,7031E-04	0,2887	3,49E-04
5	**4**	32	0,2440	0,0013	0,2119	4,4E-03
5	**5**	32	0,2320	7,2617E-04	0,256	0,0017
5	**6**	32	0,2189	0,0018	0,2788	0,0267
6	**4**	64	0,5090	0,0108	0,6165	0,08233
6	**5**	64	0,5573	0,0036	0,9520	0,0026
6	**6**	64	0,6539	0,0621	1,0667	0,0178

Tabla 9. Prueba 1.2. Variando el núm. de perfiles de servicio (Fig. Mer. 2).

Analizando los resultados obtenidos se puede ver la eficacia del algoritmo heurístico aunque el número de implementaciones de servicio se aumente de forma considerable.

Por ejemplo, para el caso de que haya 4 nodos en serie y diverso número de perfiles, los algoritmos heurísticos analizan muchas menos combinaciones y los resultados obtenidos son bastante buenos en tiempos de ejecución (sobre todo para la figura de mérito 1). A la vez que se puede decir que el error cometido por el algoritmo heurístico con figura de mérito 1 es totalmente despreciable.

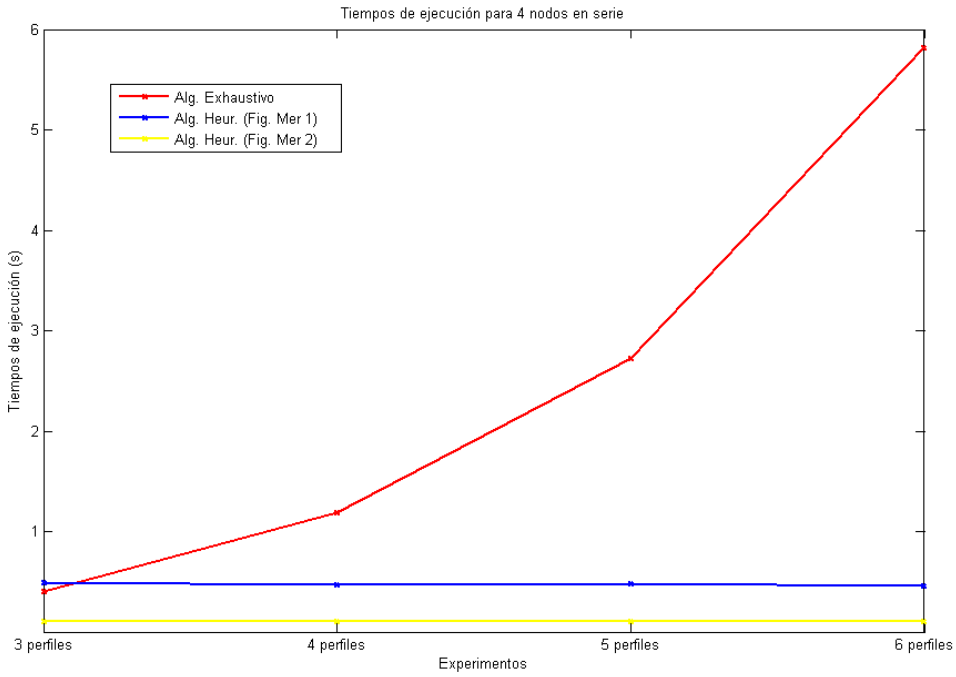


Figura 38. Comparación Algoritmos en función del Tiempo de Ejecución (4 nodos serie).

Mirando la Figura 38 se puede ver que el tiempo de ejecución para el algoritmo heurístico con figura de mérito 2 es menor, esto es debido porque en este caso se ha asumido que el tamaño de bloque más óptimo para dicho algoritmo es igual a 2. También por esta misma razón, aunque sí que es más rápido, el error que comete es mayor que para el algoritmo heurístico con figura de mérito 1.

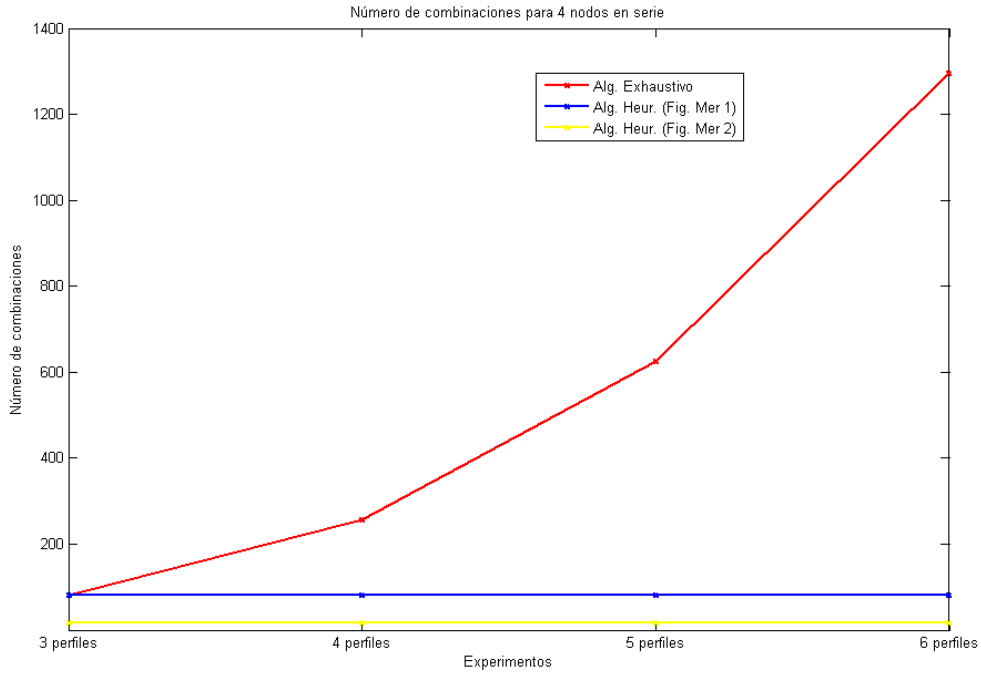


Figura 39. Comparación Alg. en función del Núm. de Combinaciones analizadas (4 nodos serie).

En la Figura 39 se puede ver cómo, para el algoritmo exhaustivo el número de combinaciones aumenta de forma exponencial, mientras que para los algoritmos heurísticos se mantiene constante. Si se observa la Figura 39 y la Figura 38, se puede apreciar cómo ambas gráficas son prácticamente iguales, lo que nos indica la importancia que tiene el limitar el número de combinaciones analizadas por los algoritmos en los sistemas de tiempo real, puesto que el tiempo de ejecución es directamente proporcional.

No pasa lo mismo con el error, puesto que éste crece a medida que disminuye el número de combinaciones examinadas. Por lo que es importante siempre tenerlo en cuenta a la hora de elegir un algoritmo u otro, y no considerar únicamente el tiempo de ejecución que presenten los mismos.

Si se analizaran el resto de los experimentos realizados con 5 y 6 nodos en serie se podría observar que los resultados son muy parecidos, puesto que el número de combinaciones que se analizan en los algoritmos heurísticos son de órdenes de magnitud menor que las que se analizan en el algoritmo exhaustivo.

Por lo tanto, los algoritmos heurísticos son más eficientes a medida que la estructura contenga más servicios y/o perfiles, ya que en caso de que aumenten el número de perfiles, las combinaciones que necesitan analizar son las mismas. Por lo tanto, lo que se

necesita es que el tamaño de bloque consiga que el error pueda ser asumible por las aplicaciones en sistemas de tiempo real, y por lo tanto será variable en función de la aplicación y sus parámetros (como puede ser la estructura de la aplicación o la cantidad de perfiles por servicio).

6.3. Conclusiones

Al comienzo del capítulo se ha realizado una breve introducción acerca de qué pruebas se iban a realizar sobre los algoritmos de composición implementados en el proyecto, así como las decisiones de diseño que se asumían en los sistemas a analizar.

Para el caso de los algoritmos de composición se ha desarrollado una herramienta que ha permitido probar todos los algoritmos bajo las mismas condiciones del sistema. Asimismo, esta herramienta también se ha utilizado para probar los algoritmos de reconfiguración.

Gracias a esta herramienta se ha podido comprobar la importancia que tiene el hecho de reducir el número de combinaciones a analizar, puesto que esto es lo que realiza el algoritmo heurístico, el cual consigue una mejoría en tiempos de ejecución que hace que el error introducido sea totalmente asumible, puesto que la relación entre estos dos parámetros es razonablemente buena.

Por lo tanto significar la validez del algoritmo heurístico frente al algoritmo exhaustivo por la eficiencia demostrada.

Capítulo 7

Evaluación y Pruebas de los Algoritmos de Reconfiguración

En este capítulo se van a mostrar los resultados obtenidos tras la realización de una serie de pruebas que se han definido para analizar los algoritmos de reconfiguración implementados en este proyecto. En un principio, se describirán brevemente las consideraciones tomadas en cuanto a decisiones de diseño, después se realizará una pequeña introducción de las pruebas que se han diseñado para la realización de una herramienta analítica que, siguiendo unas reglas extraídas como conclusiones de las pruebas realizadas, consiga reconfigurar un sistema en tiempo real utilizando el algoritmo más eficiente para ello.

7.1 Introducción

Este apartado se dedicará en exclusiva a realizar un estudio lo más exhaustivo posible de los algoritmos de reconfiguración. Se realizará una comparación de los mismos enfrentándolos a diferentes estructuras de aplicaciones, forzando, además, la caída de uno o varios servicios de dichas estructuras, para así poder comprobar mejor su comportamiento.

Tras este estudio, se realizará una herramienta analítica la cual, de forma transparente para el usuario, decidirá qué algoritmo es mejor escoger en función de la estructura y de la cantidad de perfiles que no estén disponibles en el sistema.

Por otro lado, tal como se comentó en el *Capítulo 6*, se ha considerado la misma definición de los parámetros del sistema para la evaluación de los algoritmos de reconfiguración. Considerando la misma forma de definir tanto los servicios como los perfiles, teniendo en cuenta que en esta ocasión el número de perfiles de los servicios oscilará entre 3 y 6 aleatoriamente.

Además, igual que se dijo, todos los experimentos diseñados han sido realizados un mínimo de 10 veces cada uno, por lo que los parámetros que sean susceptibles de cambios ante las variables aleatorias introducidas en el sistema vendrán definidos tanto por su media como por su varianza (indicando esta última la dispersión de los valores).

7.2 Estudio de los Algoritmos de Reconfiguración

Para probar los algoritmos de reconfiguración no se han seguido las mismas bases que las propuestas en el apartado anterior. En esta ocasión, y dado que la implementación de estos algoritmos ha sido diseñada en su totalidad para este proyecto, se realizará un estudio más a fondo, en el cual se busca encontrar un conjunto de reglas que nos indiquen qué el algoritmo de reconfiguración es mejor utilizar en función de las características de la aplicación sobre la que se esté aplicando.

Para ello, se seguirán tres pasos: análisis y comparación, estudio de resultados y, optimización y resolución. A continuación se presenta una breve introducción de cada uno, entrando en más detalle en los siguientes apartados:

1) Análisis y comparación: en este paso es donde se realizará el grueso del trabajo. Se analizarán los algoritmos exhaustivo y heurístico (recalculando la aplicación entera) y los algoritmos de reconfiguración exhaustivo y heurístico (recalculando en función de los perfiles de servicio que hubieran dejado de estar disponibles en la aplicación).

   Entre todos los parámetros que se analizan, esta parte se centrará en el número de combinaciones exploradas, los tiempos de ejecución de cada algoritmo y el error cometido por cada uno de ellos respecto de la figura de mérito global.

Dichos resultados se mostrarán tanto de forma gráfica como en tablas para así facilitar un entendimiento rápido de los resultados.

2) Estudio de resultados: en este paso se analizarán los algoritmos en función de los resultados obtenidos en el punto anterior. Se realizará un estudio individual de cada uno de ellos teniendo en cuenta los tiempos de reconfiguración obtenidos y el error medio que produce cada uno.

   De este modo, se sacarán una serie de conclusiones más o menos fehacientes acerca de cuál será el mejor algoritmo a utilizar a la hora de realizar la reconfiguración de una aplicación en función de la estructura que tenga el sistema.

   Dichas conclusiones serán denominadas como *reglas*, sobre las cuales se basará la herramienta analítica que se diseñe en el paso siguiente.

   Como curiosidad, es importante destacar en esta ocasión, que el hecho de recalcular de forma general el algoritmo exhaustivo no es ningún disparate, puesto que si la estructura de la aplicación es lo suficientemente pequeña como para que el tiempo de ejecución del algoritmo también lo sea, se estaría hablando de encontrar la combinación más óptima, es decir sin ningún tipo de error, en un tiempo de ejecución despreciable. Obteniendo así una relación *tiempo de ejecución vs. error cometido* bastante buena como para poder considerar la opción de escoger este algoritmo y clasificarlo como el más óptimo para cierto tipo de estructuras.

3) Optimización y Resolución: en la última parte se realizará la programación de una herramienta analítica que sea capaz de decidir qué algoritmo es mejor utilizar en la reconfiguración de un sistema en función de la estructura que tenga o el número de perfiles caídos que presente.

   Es decir, se diseñará un programa el cual realice la reconfiguración de un sistema de forma totalmente transparente para el usuario y que además seleccione el algoritmo más eficiente para ello.

Después de esta breve descripción de los bloques en los que se dividirá este apartado, a continuación se detallará cada uno de ellos ilustrando los resultados con tablas y figuras.

7.2.1 Paso 1: Análisis y comparación

Como ya se ha comentado, en este paso se analizará el comportamiento de los algoritmos de composición exhaustivo y heurístico (recalculando la aplicación general) y los algoritmos de reconfiguración mejorados basados en los algoritmos exhaustivo y heurístico. Se estudiarán aplicaciones con diferentes estructuras y se analizará el comportamiento de cada de los algoritmos para cada una de ellas.

Se tendrá muy en cuenta el número de combinaciones exploradas, los tiempos de ejecución y el error cometido respecto de la figura de mérito global (es decir, la conseguida por el algoritmo exhaustivo), puesto que estos parámetros serán los que se utilicen para denominar una serie de reglas que nos permitan saber que algoritmo es más eficiente para según qué estructura.

Para ello se ha creado un primer prototipo, en donde es posible analizar los cuatro algoritmos bajo las mismas condiciones del sistema (igual que se hizo para las pruebas realizadas sobre los algoritmos de composición).

A la hora de diseñar dicho prototipo, se consideró que ni la red ni los nodos estuvieran totalmente vacíos, puesto que, en un sistema cuando se cae un perfil lo más probable es que se produzca cuando ya estaba funcionando el mismo, por lo que antes de simular la caída de alguno de los nodos del sistema, la aplicación ya tenía que haber sido ejecutada por lo menos un par de veces.

Para la realización de este primer paso (*análisis y comparación*), se han considerado 4 aplicaciones básicas que serán susceptibles de análisis en este apartado. Dichas aplicaciones son:

Aplicación 1: 3 servicios en serie.

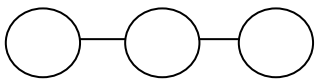


Figura 40. Reconfiguración - Aplicación 1.

Aplicación 2: 6 servicios en serie.

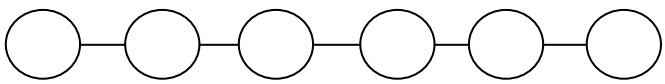


Figura 41. Reconfiguración - Aplicación 2.

Aplicación 3: Servicios en paralelo.

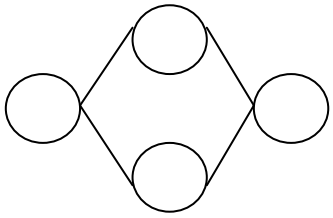


Figura 42. Reconfiguración - Aplicación 3.

Aplicación 4: Servicios mixtos.

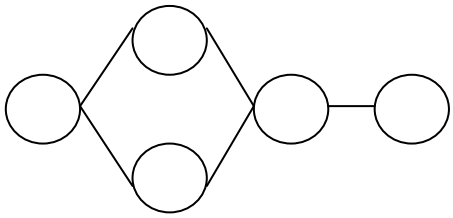


Figura 43. Reconfiguración - Aplicación 4.

Se realizará un análisis de cada una de ellas, el cual se basará en forzar la caída de alguno de los perfiles de los que se compone. Por ejemplo, para la primera estructura se forzará la caída de uno de los servicios, luego la de dos de ellos y por último la de los tres (Figura 44).

Tal como se explicó en el *Capítulo 5*, por cada servicio caído habrá que buscar otro perfil que consiga que la aplicación siga siendo planificable. Evidentemente, como en todos los casos de tiempo real, lo que premia es el tiempo que tarda la red en reconfigurarse para que todo siga funcionando, de ahí que uno de los parámetros que se estudien con más exhaustividad sea el tiempo de ejecución de los algoritmos.

También se estudiará el número de combinaciones que analiza cada algoritmo, puesto que como vimos en apartados anteriores está ligado de forma directamente proporcional al tiempo de ejecución, por lo que disminuyendo el número de combinaciones el tiempo de ejecución debería disminuir también.

Además se tendrá también en cuenta el error cometido respecto del resultado más óptimo en tiempos de respuesta. Este parámetro es igual de importante o incluso más que el tiempo de ejecución, puesto que, en algunos casos, no sirve de nada una reconfiguración en un tiempo despreciable cuando el error cometido es excesivamente alto. Por ello se

impondrán márgenes de error en cada una de las aplicaciones, puesto que dependiendo de la estructura que se elija el margen podrá ser más o menos alto.

Por lo tanto, lo que se pretende conseguir al final de todos los pasos es una herramienta que consiga elegir un algoritmo que permita una buena relación entre el error cometido y el tiempo de ejecución.

Los algoritmos que se utilizarán para el estudio de las aplicaciones descritas anteriormente son:

- Algoritmo exhaustivo.
- Algoritmo heurístico (utilizando la Figura de Mérito 1 y un tamaño de bloque igual a 2).
- Algoritmo de reconfiguración exhaustivo.
- Algoritmo de reconfiguración heurístico (utilizando la Figura de Mérito 1 y un tamaño de bloque igual a 2).

En esta ocasión para el caso de los algoritmos heurísticos (tanto el de composición como el de reconfiguración) se utilizará la figura de mérito 1, puesto que como se vio en el anterior capítulo, las dos figuras de mérito presentan resultados muy parecidos. Por lo tanto no se trabajará con la figura de mérito 2, dando por hecho que los resultados serían similares o iguales a los obtenidos con la figura de mérito 1.

La forma de considerar la caída de los perfiles en la herramienta diseñada ha sido la misma que la indicada cuando se explicaron los algoritmos de reconfiguración en el *Capítulo 5*, y es introduciendo un nuevo parámetro, el cual define el perfil del servicio que se ha caído poniéndolo a 1 en la posición que corresponda en la estructura, siendo así más fácil identificar cuántos perfiles han dejado de estar disponibles y en qué posición se encuentran dentro de la estructura.

Como ya se verá en los siguientes apartados, en especial en el *apartado 7.2.3*, la cantidad de perfiles que hayan dejado de estar disponibles será otro parámetro determinante, junto con la relación tiempo de ejecución/error cometido, para decidir qué algoritmo es mejor utilizar para la reconfiguración.

A la hora de realizar la reconfiguración de una aplicación, es importante tener en cuenta que los perfiles de los servicios que siguen estando disponibles en el sistema serán

exactamente los mismos que los que se tenían en la iteración anterior (no cambiará ni el tiempo de computación ni el nodo físico al que pertenecen), y con los cuales la aplicación era planificable. Por ello, al reconfigurar únicamente el/los nodo/s caído/s es de esperar que el error que se introduzca pudiera ser incluso mayor que el que pueda introducir el algoritmo de composición heurístico.

Además, también se debe tener en cuenta que el tiempo de ejecución de los algoritmos heurísticos depende en gran medida del tamaño de bloque que se esté utilizando, por ello, en todos los casos que se han estudiado se ha elegido el mismo tamaño de bloque para los dos. Se ha elegido el tamaño de bloque igual a dos, porque después de realizar las pruebas de optimización de los algoritmos de composición, se descubrió que dicho tamaño de bloque introducía un error completamente asumible por casi cualquier sistema reduciendo el tiempo de ejecución del algoritmo de forma notable.

A continuación se dividirá el apartado en 4 partes, éstas estarán compuestas por las pruebas realizadas sobre cada una de las aplicaciones citadas anteriormente (3 servicios en serie, 6 servicios en serie, servicios en paralelo y servicios mixtos). Asimismo se mostrarán tablas y gráficas para una visualización más clara de las cuatro situaciones.

7.2.1.1 Aplicación 1: 3 servicios en serie.

En esta parte se hará un estudio de la primera aplicación y se verá cómo afecta la misma a los algoritmos objeto de estudio en esta parte de *Análisis y Comparación*.

Al realizar la simulación se seguirán los pasos mostrados en la Figura 44 siguiendo exactamente el mismo orden que el indicado: primeramente se caerá el primer servicio de la izquierda, luego el del medio y finalmente el de la derecha, de forma que en la última situación estarán los tres servicios caídos y habrá que reconfigurar todo el sistema.

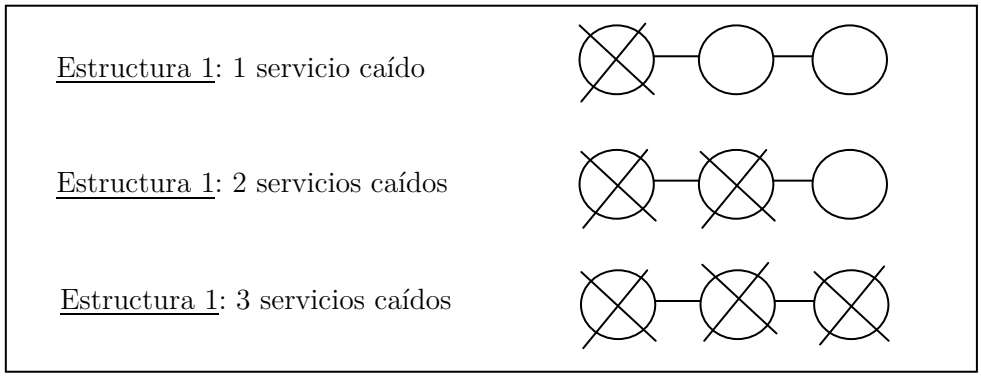


Figura 44. Alg. Reconfig. Aplicación 1. Secuencia de caída de servicios.

En las tablas que se muestran a continuación se pueden ver los parámetros más importantes a tener en cuenta: el número de combinaciones, el tiempo de ejecución y el error respecto de la figura de mérito global. Con los que se concluirán las reglas que se detallarán en el *Apartado 7.2.2* (*Estudio de resultados*).

Se han separado los resultados en dos tablas, por un lado la perteneciente al algoritmo exhaustivo (Tabla 10), que como ya se ha comentado en varias ocasiones en este proyecto, obtiene la combinación más óptima, por lo que el error que se comete en el resto de los algoritmos se mide con los resultados de éste.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**	
			Media	**Var.**
Composición Exhaustivo	**1**	48	0,08250	5,076E-05
	2	48	0,08573	6,249E-05
	3	48	0,090339	3,582E-04

Tabla 10. Reconfiguración. Alg. Exhaustivo - Aplicación 1.

En la Tabla 11 se han metido los otros tres algoritmos (de composición heurístico, de reconfiguración exhaustivo y de reconfiguración heurístico), en la cual se ha introducido una columna más referente a la medida del error.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**		**Error [%]**	
			Media	**Var.**	**Media**	**Var.**
Composición Heurístico	**1**	8	0,016984	2,064E-05	0,0175	0,016
	2	8	0,016819	2,421E-05	0,018	0,012
	3	8	0,017038	2,345E-05	0,023	0,021
Reconfig. Exhaustivo	**1**	3	0,0093	5,781E-05	0,053	0,025
	2	12	0,028	3,464E-04	0,042	0,024
	3	48	0,0876	6,144E-05	0	0
Reconfig. Heurístico	**1**	2	0,0077	5,941E-05	0,075	0,076
	2	4	0,0091	3,5309E-04	0,073	0,056
	3	8	0,0169	2,229E-05	0,023	0,021

Tabla 11. Reconfiguración. Alg. Heurístico y Alg. Reconf. - Aplicación 1.

Por otro lado las columnas comunes a las dos tablas son las referentes al número de perfiles caídos en el sistema, el número de combinaciones evaluadas y los tiempos de ejecución en cada caso.

Por otro lado y tal como se hizo en las pruebas de los algoritmos de composición cada experimento se ha realizado un mínimo de 10 veces para poder hallar tanto la media de los tiempos de ejecución como del error, puesto que los mismos son valores que varía en función de los servicios que haya, los perfiles que contenga cada servicio, y otros parámetros internos de las redes que se han considerado aleatorios.

Como se observa en las tablas anteriores, se puede apreciar el aumento del tiempo de ejecución en función del número de combinaciones a analizar. En el caso de los algoritmos de composición se analizan siempre las mismas combinaciones, por lo que los tiempos de ejecución apenas varían, pero en el caso de los algoritmos de reconfiguración el número de combinaciones varía en función de los perfiles caídos, por lo que a medida que éstos aumentan, también aumenta su tiempo de ejecución.

Este hecho puede verse claramente en la Figura 45 si observamos la línea que dibujan los dos algoritmos de reconfiguración.

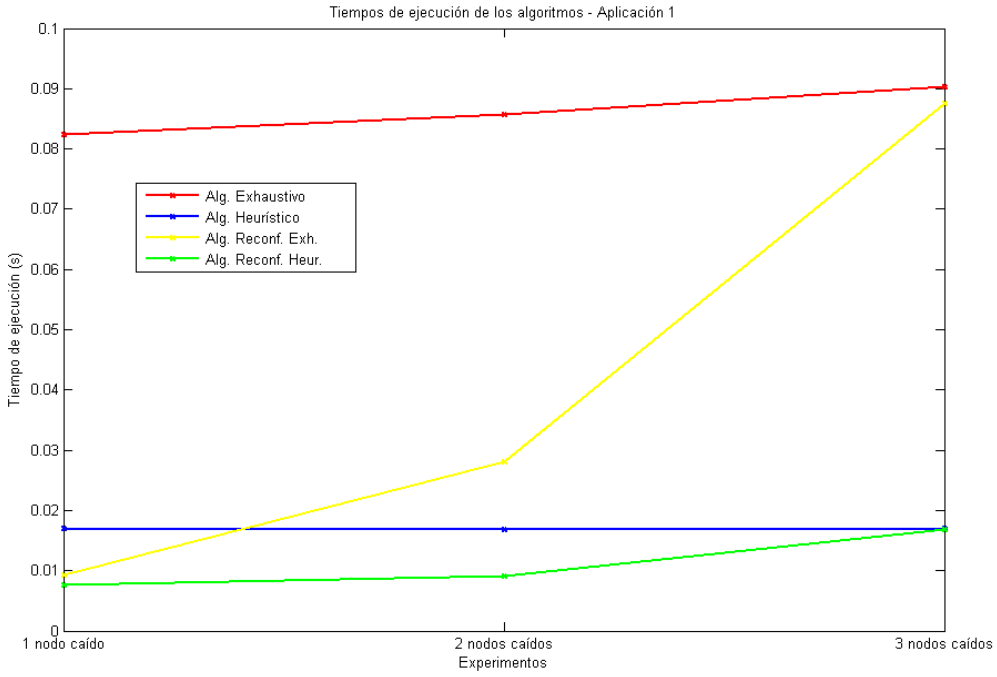


Figura 45. Tiempos de reconfiguración - Aplicación 1.

El error que están mostrando todos los algoritmos es despreciable, así como los tiempos de ejecución tampoco son excesivamente altos. Por lo que, si se tuviera que elegir un algoritmo entre todos ellos, se tendrían que tener en cuenta estos dos casos:

- Que se necesite un tiempo excesivamente bajo, en cuyo caso se elegiría cualquiera de los algoritmos heurísticos, tanto el de reconfiguración como el de composición, los cuales se diferencian únicamente por un mínimo margen de error, o incluso el algoritmo de reconfiguración exhaustivo si únicamente se produce la caída de 1 perfil en uso por el sistema.

- Que se necesite un error nulo, en cuyo caso se elegiría el algoritmo de composición exhaustivo, puesto que ofrece dicha posibilidad y además el tiempo de ejecución que presenta el mismo es lo suficientemente bajo como para considerarlo viable en un sistema de tiempo real.

7.2.1.2 Aplicación 2: 6 Servicios en serie

En esta parte se hará un estudio de la segunda aplicación y se verá cómo afecta la misma a los algoritmos objeto de estudio en esta parte de *Análisis y Comparación*.

Para la simulación de la *Aplicación 2* se seguirán los pasos mostrados en la Figura 46, siguiendo exactamente el mismo orden que el indicado: primeramente se caerá el primer servicio de la izquierda, luego el segundo, luego el primero, tercero y último servicio, y así sucesivamente hasta completar todos los experimentos que se indican a continuación.

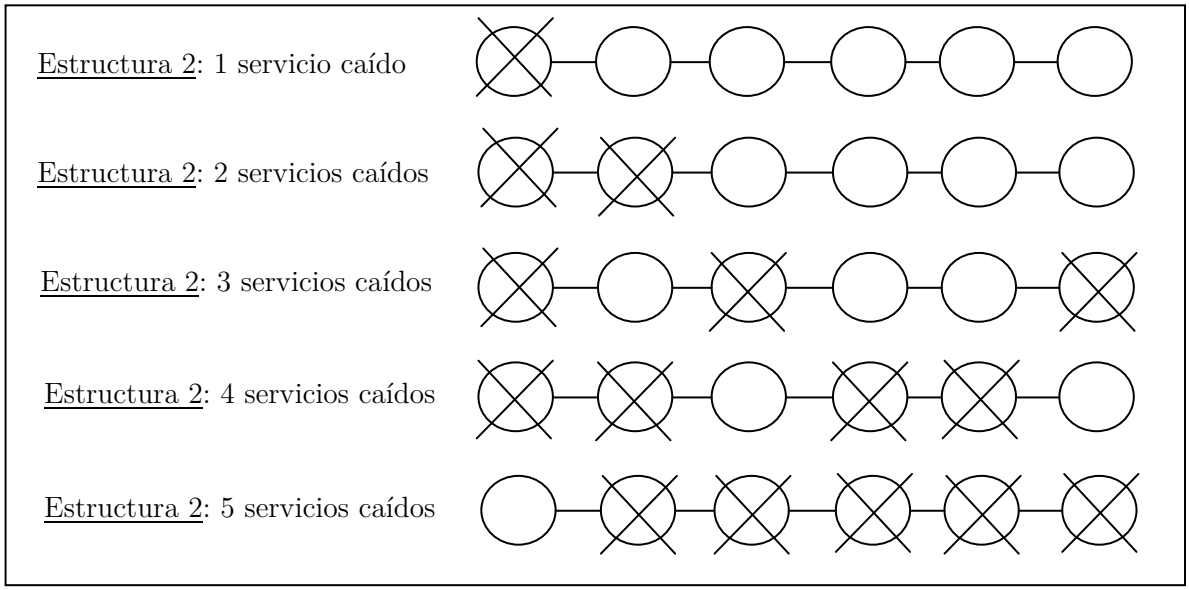


Figura 46. Alg. Reconfig. Aplicación 2. Secuencia de caída de servicios

La Tabla 12 y Tabla 13 siguen el mismo formato que las mostradas anteriormente. La única diferencia con las anteriores es el número de experimentos que se realizan, puesto que estos van de acuerdo con los fijados en la Figura 46.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**	
			Media	**Var.**
Composición Exhaustivo	**1**	2304	9,5555339	1.56437
	2	2304	9,759839	3,02034
	3	2304	9,577889	3,302871
	4	2304	8,77959	0,0611016
	5	2304	9,563676	2,46011

Tabla 12. Reconfiguración. Alg. Exhaustivo - Aplicación 2.

Igual que en el anterior, en el caso de los algoritmos que se muestran en la Tabla 13 se ha introducido la columna referente al error que introduce cada uno de ellos.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**		**Error**	
			Media	**Var.**	**Media**	**Var.**
Composición Heurístico	**1**	64	0,245012	0,0011175	2,2	1,29E-02
	2	64	0,24899	0,0017264	2,5	1,46E-02
	3	64	0,251688	0,0027503	2,2	4,89E-02
	4	64	0,2279705	5,849E-05	2,44	1,76E-02
	5	64	0,2461437	0,00129744	2,8	3,96E-02
Reconfig. Exhaustivo	**1**	3	0,0141	7,029E-05	5,6	3,14E-02
	2	12	0,0545	3,0605E-04	4,9	3,29E-02
	3	48	0,1985	0,003789	3,16	3,16E-02
	4	144	0,5328	0,0027599	1,2	1,66E-02
	5	768	2,9329	0,169737	0,85	1,15E-02
Reconfig. Heurístico	**1**	2	0,0126	3,984E-05	6,2	4,95E-02
	2	4	0,0251	3,4989E-04	5,5	3,02E-02
	3	8	0,0485	0,00104825	5,1	3,37E-02
	4	16	0,0608	3,2216E-04	4,5	1,66E-02
	5	32	0,1265	6,1565E-04	3,1	9,84E-02

Tabla 13. Reconfiguración. Alg. Heurístico y Alg. Reconf. - Aplicación 2.

En la Figura 47 se pueden ver los tiempos de ejecución de los algoritmos propuestos para la reconfiguración de la *Aplicación 2*. En este caso se puede ver a simple vista cómo el algoritmo de composición exhaustivo a pesar de que con él se obtiene la combinación óptima, no es el más eficiente en cuanto a tiempos de ejecución, por lo que para estructuras con varios servicios en serie con total seguridad no será el que se elija.

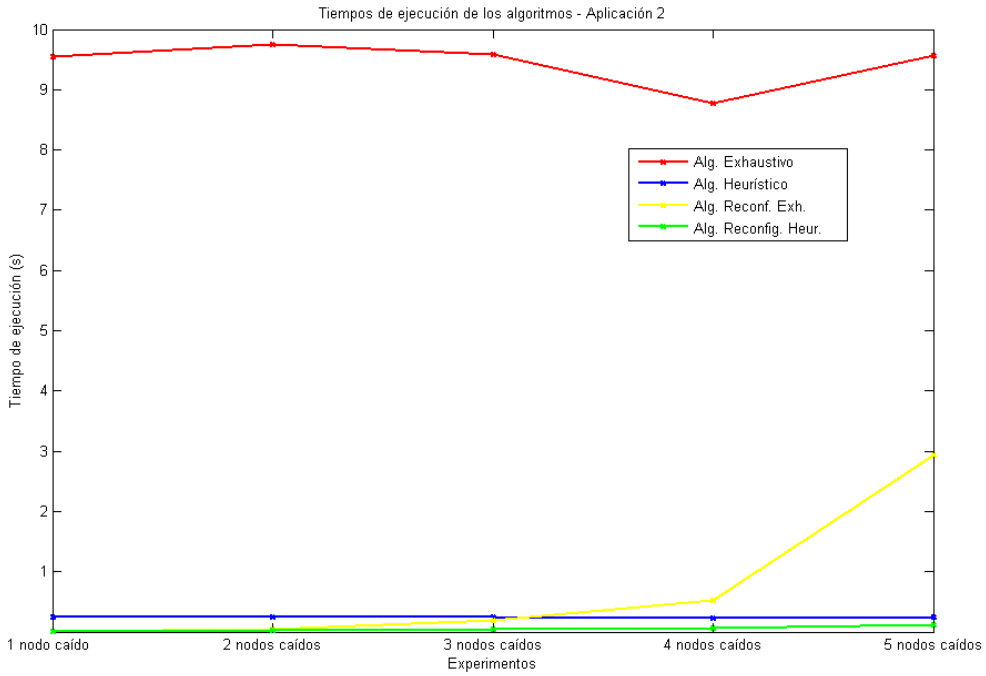


Figura 47. Tiempos de reconfiguración - Aplicación 2.

En la Figura 48, la cual presenta un zoom sobre la parte inferior de la figura anterior, se puede apreciar mejor la comparativa entre los otros tres algoritmos. En ella se ve como el tiempo de ejecución de los algoritmos de reconfiguración es notablemente inferior al del algoritmo de composición heurístico cuando los servicios caídos son 3 o menos. Si los servicios caídos superan la mitad de los que hay en el sistema, el tiempo del algoritmo de reconfiguración exhaustivo se dispara, puesto que su número de combinaciones crece de forma exponencial.

Por lo tanto para elegir uno de los dos algoritmos heurísticos (el de composición o el de reconfiguración) tendríamos que remitirnos al error introducido por ambos. Lo cual se puede ver en la Figura 49.

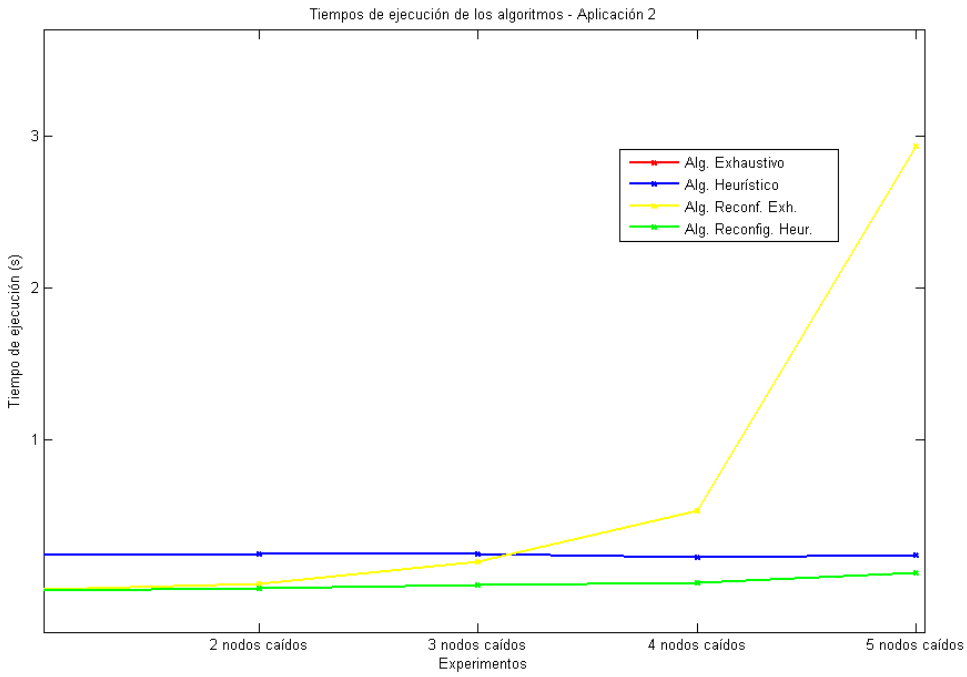


Figura 48. Zoom - Tiempos de reconfiguración - Aplicación 2

Si se asume la caída de pocos servicios en la red (3 o menos según la estructura que se está evaluando en este apartado), el algoritmo elegido sería el de composición heurístico, puesto que incluso es el que menor error presenta de los tres. Si se cayeran más de la mitad de los nodos del sistema, entonces, tal como se comentó anteriormente, los dos algoritmos seleccionados serían los dos heurísticos.

Viendo la gráfica a partir de 3 servicios caídos, se puede ver como el error en el algoritmo exhaustivo se va haciendo mínima, como es lógico, puesto que cada analiza más combinaciones y por lo tanto es más fácil que encuentre la óptima, pero este algoritmo se descartó porque su tiempo de ejecución empezaba a crecer de forma exponencial.

Por otro lado, si se mira las líneas que siguen los algoritmos heurísticos a partir de la caída del 4 servicio, se podría decir que es la misma. Por lo tanto, como conclusión se podría decir que a partir de la caída del 60% de los servicios del sistema, el algoritmo más eficiente por mejor relación entre el tiempo de ejecución y el error cometido, sería el algoritmo de reconfiguración heurístico.

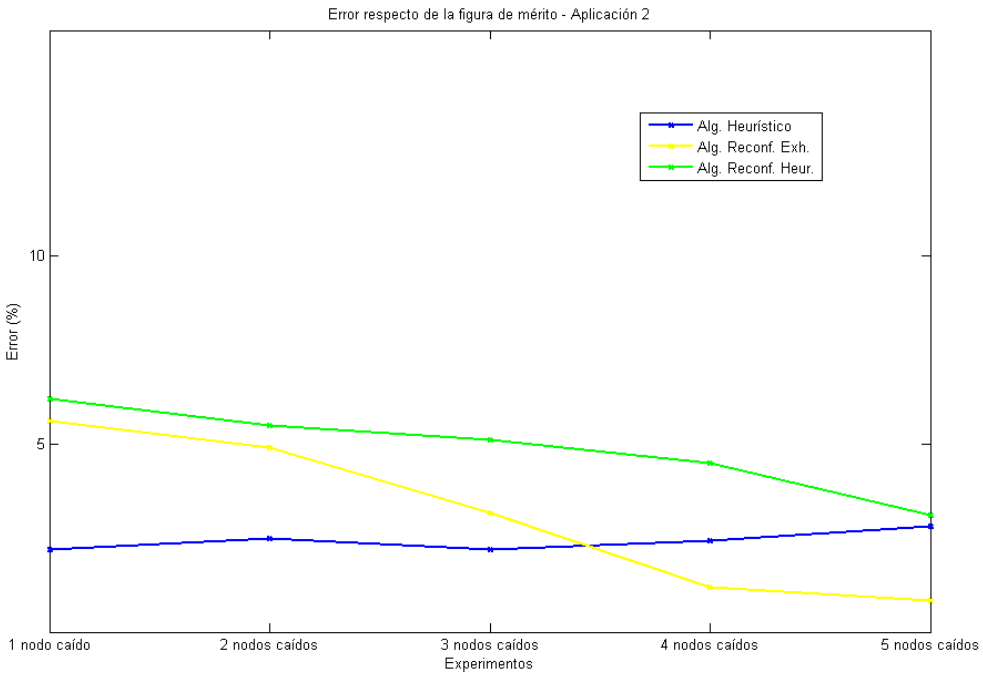


Figura 49. Error de los Alg. Reconfig. respecto de la figura de mérito global - Aplicación 2.

Para finalizar este apartado, y como resumen del mismo, se ha llegado a las siguientes conclusiones para una estructura con bastantes servicios:

- Si se caen entre 0 y el 60% de los servicios del sistema, el algoritmo a utilizar sería el algoritmo de composición heurístico.

- Si se caen el 60% de los servicios hasta el 100%, el algoritmo a utilizar sería el algoritmo de reconfiguración heurístico.

La evaluación seguida es altamente dependiente del margen de error que se quiera para el sistema, puesto que si se pretende minimizar dicho margen a cierto valor entonces los rangos indicados arriba cambiarían, y los correctos serían los siguientes:

- Si se caen entre 0 y el 60% de los servicios del sistema, el algoritmo a utilizar sería el algoritmo de composición heurístico.

- Si se caen el 60% de los servicios hasta el 80%, el algoritmo a utilizar sería el algoritmo de reconfiguración exhaustivo, puesto que es el que menos error introduce, aunque su tiempo de ejecución sea mayor. Este caso se utilizará cuando premie el tener un error muy reducido frente al tiempo que tarde en hallar la solución.

- Si se caen el 80% de los servicios hasta el 100%, el algoritmo a utilizar sería el algoritmo de reconfiguración heurístico. Este último caso sería muy dependiente también de la relación entre el error y el tiempo de ejecución, puesto que los algoritmos exhaustivos (tanto el de reconfiguración como el de composición) elevan de forma exponencial su número de combinaciones y por lo tanto su tiempo de ejecución, por lo que para sistemas de tiempo real son poco eficientes. Mientras que con los algoritmos heurísticos se mantiene un error más o menos constante, y además los tiempos de ejecución siguen siendo bastante bajos. Los tiempos del algoritmo de reconfiguración heurístico son ligeramente más bajos, de ahí que se haya determinado utilizar dicha opción.

7.2.1.3 Aplicación 3: Servicios en paralelo

En esta parte se hará un estudio de la tercera aplicación y se verá cómo afecta la misma a los algoritmos objeto de estudio en esta parte de *Análisis y Comparación*.

Para la simulación de la *Aplicación 3* se seguirán los pasos mostrados en la Figura 50, siguiendo exactamente el mismo orden que el indicado, leyendo primero de izquierda a derecha, y luego de abajo a arriba, por lo tanto primeramente se caerá uno de los servicios del paralelo, luego los dos servicios del paralelo, luego el primero de la izquierda, y por último se caerán todos los servicios de la estructura, tal como si indica a continuación.

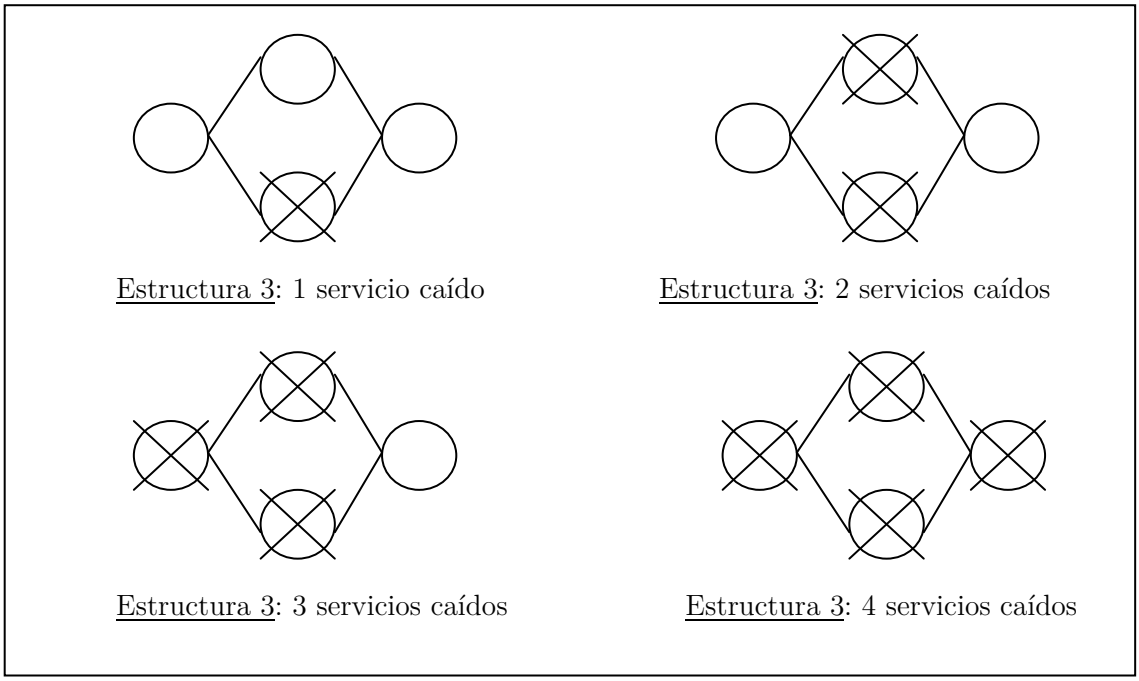


Figura 50. Alg. Reconfig. Aplicación 3. Secuencia de caída de servicios

La Tabla 14 y Tabla 15 siguen el mismo formato que las mostradas anteriormente. La única diferencia con las anteriores es el número de experimentos que se realizan, puesto que estos van de acuerdo con los fijados en la Figura 50.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**	
			Media	**Var.**
Exhaustivo	**1**	256	0,52002	2,1565E-03
	2	256	0,526176	1,0464E-03
	3	256	0,6 10855	6,1E-04
	4	256	0,521575	1,4229E-03

Tabla 14. Reconfiguración. Alg. Exhaustivo - Aplicación 3.

Igual que en el apartado anterior, en el caso de los algoritmos que se muestran en la Tabla 15 se ha introducido la columna referente al error que introduce cada uno de ellos.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**		**Error**	
			Media	**Var.**	**Media**	**Var.**
Heurístico	**1**	16	0,044987	2,281E-05	1,352	0,012
	2	16	0,0350468	5,161E-05	1,412	0,361
	3	16	0.0326266	2.229E-05	1,61	0,141
	4	16	0,03876	2,0381E-04	1,385	0,58
Reconfig. Exhaustivo	**1**	4	0,0142	3,9276E-04	3,8	0,16
	2	16	0,0458	5,016E-05	1,5	19,6E-02
	3	64	0,2517	0,00018	1,1	15,3E-02
	4	256	0,5287	9,9761E-03	0	0
Reconfig. Heurístico	**1**	2	0,0078	5,91E-04	4,15	0,076
	2	4	0,0196	3,569E-03	3,73	0,056
	3	8	0,0275	2,59E-04	2,23	0,021
	4	16	0,0408	2,216E-04	1,586	1,66E-02

Tabla 15. Reconfiguración. Alg. Heurístico y Alg. Reconf. - Aplicación 3.

En la Figura 51 se pueden ver los tiempos de ejecución de los algoritmos propuestos para la reconfiguración de la *Aplicación 3*. En este caso se pueden apreciar que los tiempos de ejecución siguen un patrón parecido a las aplicaciones anteriores.

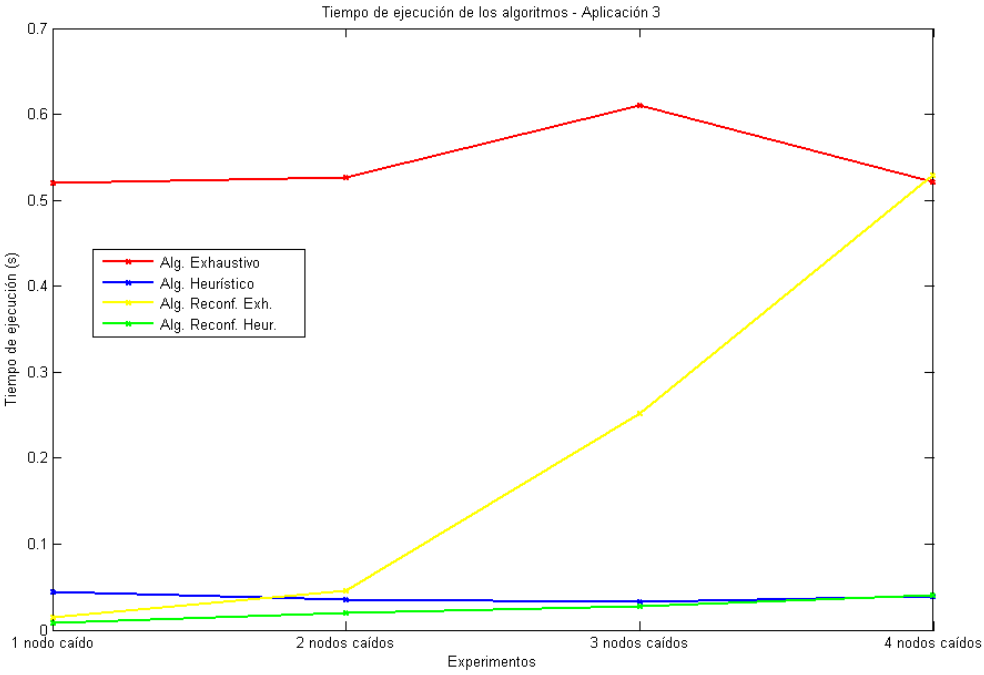


Figura 51. Tiempos de reconfiguración - Aplicación 3.

Como ha sucedido ya en el resto de las pruebas se puede ver cómo los tiempos de ejecución de los algoritmos de composición son bastante lineales, puesto que siempre analizan el mismo número de combinaciones, sin embargo los tiempos de los algoritmos de reconfiguración van incrementándose a medida que va aumentado el número de servicios caídos.

Esto se puede ver perfectamente en esta gráfica con el algoritmo de reconfiguración exhaustivo, que aumenta de forma exponencial su tiempo de ejecución hasta llegar a juntarse con la línea de tiempos de ejecución del algoritmo de composición exhaustivo, esto se produce porque en ese momento todos los servicios de la aplicación habían dejado de estar disponibles. Por lo tanto, en estas situaciones es donde más se aprecia la eficiencia de los algoritmos heurísticos.

En Figura 52, se muestra el error respecto de la figura de mérito de la *Aplicación 3*, tal y como pasaba con los tiempos de ejecución, pasa con el error, se puede ver como el algoritmo de composición heurístico mantiene un error constante, mientras que los dos algoritmos de reconfiguración van disminuyendo ese error a medida que aumenta el número de servicios caídos.

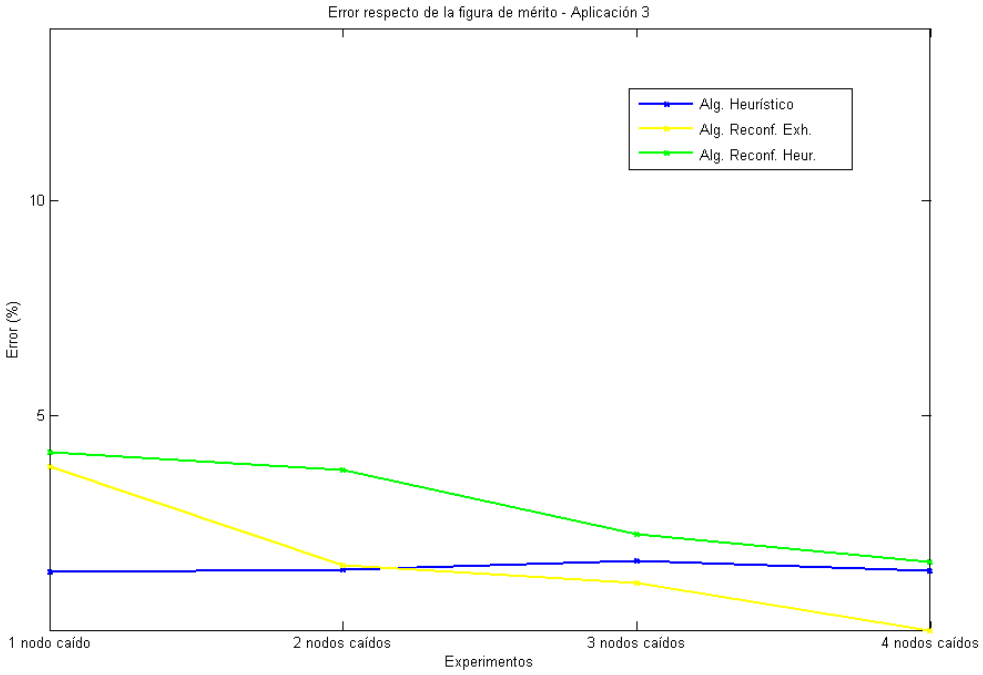


Figura 52. Error de los Alg. Reconfig. respecto de la figura de mérito global - Aplicación 3.

Se puede ver en las gráficas, cómo el hecho de introducir servicios en paralelo en las aplicaciones no influye excesivamente en el aumento del error. Puesto que la reconfiguración de cualquier servicio puede ocasionar un aumento del error si la dispersión de los valores de los tiempos de computación de los perfiles es muy variable.

Por lo tanto, se puede concluir que el incremento de servicios en paralelo influye más negativamente sobre los tiempos de ejecución (puesto que hace que éstos se incrementen de forma exponencial), que sobre el error, por lo que a la hora de hacer una clasificación de aplicaciones no se dará ninguna importancia a la estructura de la aplicación, sino a la cantidad de servicios que la componen.

Aún así, es importante señalar que dependiendo del margen de error que se estime como máximo, así se elegirán los algoritmos.

7.2.1.4 Aplicación 4: Servicios mixtos.

En esta parte se hará un estudio de la cuarta aplicación y se verá cómo afecta la misma a los algoritmos objeto de estudio en esta parte de *Análisis y Comparación*.

Para la simulación de la *Aplicación 4* se seguirán los pasos mostrados en la **Figura 53**, siguiendo exactamente el orden indicado: primeramente se caerá el primer servicio de la izquierda, luego el siguiente que hay en serie, y así sucesivamente.

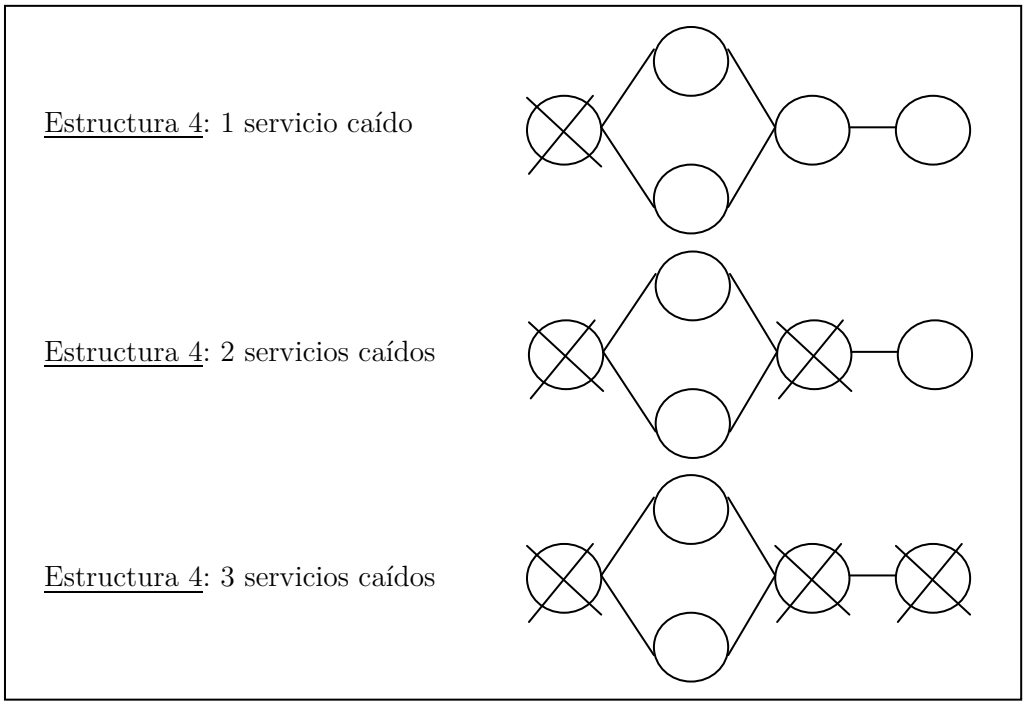


Figura 53. Alg. Reconfig. Aplicación 4. Secuencia de caída de servicios

En esta ocasión sólo se analizará qué es lo que ocurre en los casos indicados, puesto que ya se ha estudiado qué pasa en el caso de que se caigan los nodos del paralelo.

La Tabla 16 y Tabla 17 siguen el mismo formato que las mostradas anteriormente. La única diferencia con las anteriores es el número de experimentos que se realizan, puesto que estos van de acuerdo con los fijados en la Figura 53.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**	
			Media	**Var.**
Exhaustivo	**1**	1024	3,0351859	9,056565E-02
	2	1024	3,297780	0,0198318
	3	1024	3,153689	0,021544

Tabla 16. Reconfiguración. Alg. Exhaustivo - Aplicación 4.

Igual que en el apartado anterior, en el caso de los algoritmos que se muestran en la Tabla 17 se ha introducido la columna referente al error que introduce cada uno de ellos.

Algoritmo	**Núm. de perfiles caídos**	**Núm de Comb.**	**Tiempo de ejecución**		**Error**	
			Media	**Var.**	**Media**	**Var.**
Heurístico	**1**	32	0,183362	0,0170624	2,65	1,21E-03
	2	32	0,187743	0,0018968	2,49	0,065
	3	32	0,184023	5,796E-05	2,57	0,044
Reconfig. Exhaustivo	**1**	4	0,0498	2,2956E-04	4,2	4,26E-02
	2	16	0,1204	0,001311	2,9	1,6E-02
	3	64	0,3613	0,0238898	1,45	7,65E-03
Reconfig. Heurístico	**1**	2	0,0312	4,9316E-04	4,3	4,56E-02
	2	4	0,0625	5,8605E-04	4	0,01561
	3	8	0,0858	4,5476E-04	3,1	1,49E-02

Tabla 17. Reconfiguración. Alg. Heurístico y Alg. Reconf. - Aplicación 4.

En la Figura 54 se pueden ver los tiempos de ejecución de los algoritmos propuestos para la reconfiguración de la *Aplicación 4*. En este caso se pueden apreciar que la gráfica que se obtiene de los tiempos de ejecución es muy parecida a las obtenidas para el resto de las aplicaciones anteriores.

Como ya se ha comentado en otras ocasiones, es normal, puesto que si hay pocos servicios caídos los tiempos de ejecución de los algoritmos de reconfiguración son despreciables, pero a medida que se aumenta el número de servicios no disponibles, comienzan a aumentar los tiempos de ejecución, mientras que los algoritmos de composición se mantienen más o menos constantes.

De ahí que la línea del algoritmo de composición heurístico se cruce con la del algoritmo de reconfiguración exhaustivo, puesto que éste último crece de forma exponencial conforme se caen los servicios.

Igualmente pasa con la gráfica del error respecto a la figura de mérito global que se presenta en la Figura 55. Se puede ver como los algoritmos de reconfiguración bajan su error por cada servicio caído, esto es porque cada vez necesitan analizar mayor número de combinaciones. En los algoritmos estudiados obtenemos unos tantos por cientos de error

que se mueven entre el 3,5% y el 5,5%, variando según el algoritmo y el número de nodos caídos que haya en el sistema.

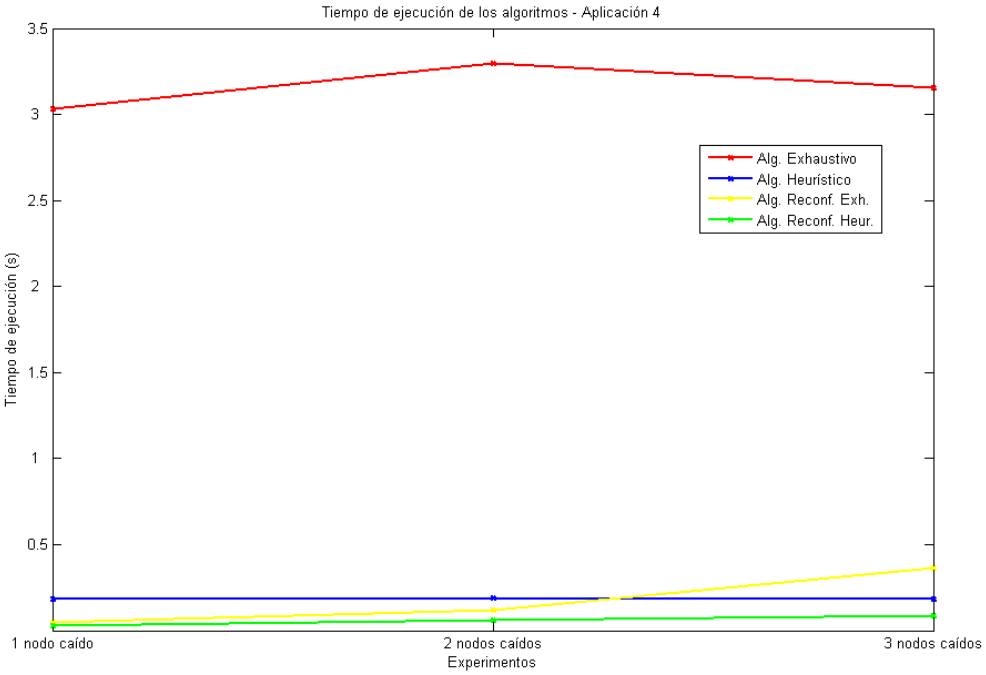


Figura 54. Tiempos de reconfiguración - Aplicación 4.

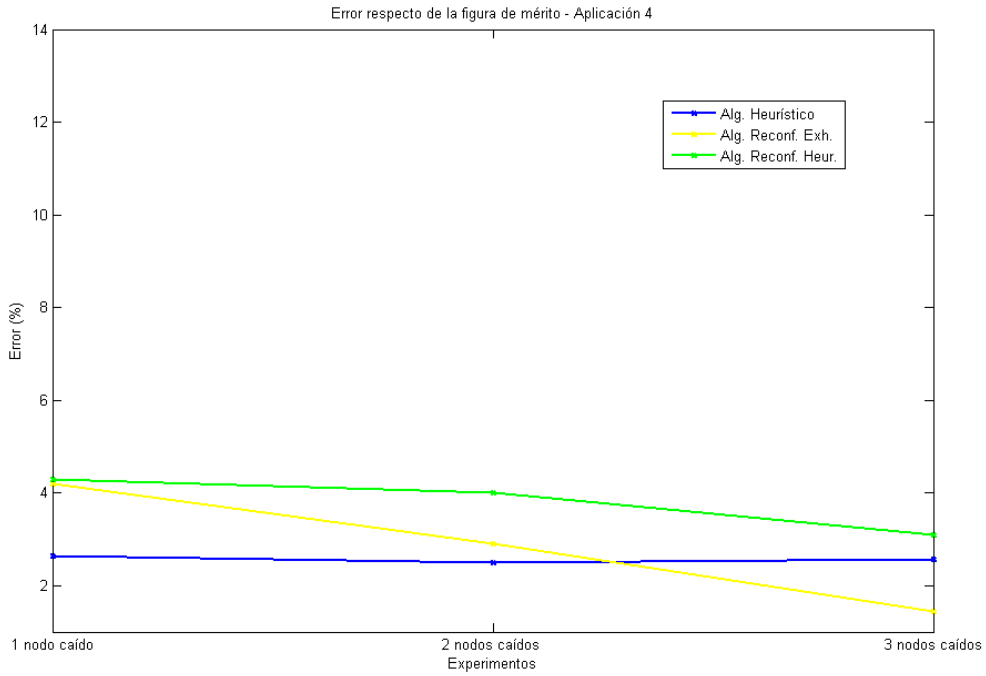


Figura 55. Error de los Alg. Reconfig. respecto de la figura de mérito global - Aplicación 4.

Por otro lado, se puede apreciar como el comportamiento, a pesar de ser una aplicación totalmente diferente, sigue siendo bastante parecido al resto de las aplicaciones, concluyendo de la misma forma que en la anterior aplicación, se podría decir que para aplicaciones que tengan más de 4 servicios, se determinarán las reglas indicadas en la *Aplicación 2*.

Como ya se comentó al inicio de este apartado, aun habiendo introducido un servicio en paralelo, el resultado no ha variado de manera significativa, igual que pasaba en la *Aplicación 3*.

7.2.2 Paso 2: Estudio de resultados

Como ya se comentó al inicio de este capítulo, en este apartado se analizarán los resultados obtenidos en las pruebas realizadas sobre los cuatro algoritmos implementados a lo largo del presente proyecto, los cuales han sido utilizados para la reconfiguración de sistemas en tiempo real.

A raíz de ese análisis se desarrollarán una serie de reglas o normas, las cuales permitirán, de forma transparente para un usuario cualquiera de una aplicación de un sistema de tiempo real, la mejor reconfiguración posible en caso de caída del sistema gracias a la elección del algoritmo más óptimo en cada caso. Para ello se realizará una herramienta analítica diseñada para conseguir la transparencia citada frente al usuario.

Tras las pruebas realizadas a lo largo de los *Capítulos 6 y 7*, se ha podido encontrar ciertas normas que se cumplen para cualquier tipo de aplicación, por ejemplo:

- El tiempo de ejecución de los algoritmos es directamente proporcional al número de combinaciones analizadas por los mismos.
- El error respecto del tiempo de respuesta óptimo es, en general, decreciente a medida que se aumenta el número de combinaciones analizadas por los mismos.

Estos dos casos son totalmente lógicos, ya que cuantas más combinaciones analices más posibilidades tienes de dar con la más óptima, el único problema es que se tarda más tiempo en encontrarla, por eso para el diseño del prototipo se valorará de forma importante la posibilidad de introducir los márgenes de error por parte del usuario, asumiendo que es dicho valor es el máximo error asumible por la aplicación.

Además es importante tener en cuenta el *time-out* que se está introduciendo, puesto que dependiendo del valor de éste se podría reducir también el número de combinaciones. Como ya se ha comentado en otras ocasiones, dicho parámetro es variable en función de la estructura de la aplicación que se vaya a estudiar.

A continuación se realiza un resumen de las principales conclusiones obtenidas en las pruebas realizadas anteriormente:

7.2.2.1 Reglas sobre la Aplicación 1

Tanto los tiempos de ejecución de los algoritmos obtenidos en esta prueba, como el error que comenten los mismos frente a la figura de mérito es muy baja. Por lo que se llega a la conclusión que, para aplicaciones con estructuras con un número de servicios inferior a 3, la elección del algoritmo se realizará en función de si se necesita un margen de error completamente nulo o un tiempo de ejecución bastante bajo.

Asimismo la clasificación de los algoritmos se realizará en función del:

- Tiempo de ejecución y 1 único servicio caído: se elegiría el algoritmo de reconfiguración exhaustivo.
- Tiempo de ejecución y más de 1 servicio caído: se elegiría cualquiera de los algoritmos heurísticos, tanto el de reconfiguración como el de composición, los cuales se diferencian únicamente por un mínimo margen de error.
- Error: se elegiría el algoritmo de composición exhaustivo, puesto que ofrece la combinación óptima y además su tiempo de ejecución es lo suficientemente pequeño como para ser considerado despreciable para un sistema de tiempo real.

7.2.2.2 Reglas sobre la Aplicación 2

En esta estructura se empiezan a notar diferencias notables entre los cuatro algoritmos que se están estudiando. En el apartado correspondiente a dicha aplicación se hizo una clasificación de qué algoritmo es mejor utilizar en cada situación, la cual se mantendrá sin ningún cambio en este apartado.

Por lo tanto, la clasificación de los mismos para estructuras con más de 3 servicios en serie es:

- Si se caen entre 0 y el 60% de los servicios del sistema, el algoritmo a utilizar sería el algoritmo de composición heurístico.

- Si se caen el 60% de los servicios hasta el 80%, el algoritmo a utilizar sería el algoritmo de reconfiguración exhaustivo, puesto que es el que menos error introduce, aunque su tiempo de ejecución sea mayor.

- Si se caen el 80% de los servicios hasta el 100%, el algoritmo a utilizar sería el algoritmo de reconfiguración heurístico. Aunque ambos algoritmos heurísticos mantienen un error más o menos constante, y además unos tiempos de ejecución bastante bajos. En cualquier caso, los tiempos del algoritmo de reconfiguración heurístico son ligeramente más bajos, de ahí que se haya determinado utilizar dicha opción.

7.2.2.3 Reglas sobre la Aplicación 3

Para el caso de la Estructura 3, en la cual el dato destacable es la introducción de un servicio en paralelo sobre una estructura con 3 servicios en serie, se ha podido destacar que el tiempo de ejecución de los algoritmos aumenta exponencialmente debido a la introducción de otro servicio más. Además de no apreciarse cambios significativos derivados de la inserción de este nuevo servicio.

Por lo que, a la hora de realizar la clasificación de los algoritmos para aplicaciones con algún servicio en paralelo, no se tendrá en cuenta en que parte de la estructura se encuentre el servicio, sino que simplemente se contará como un servicio más, Puesto que prevalece el hecho de aumentar el número de combinaciones de forma exponencial frente al error que se comete.

7.2.2.4 Reglas sobre la Aplicación 4

Tal y como se concluyó en el apartado referente a esta aplicación, y en el anterior apartado, no se han extraído reglas diferentes a las ya incluidas en las anteriores estructuras. Puesto que si se introduce un servicio nuevo (da igual en paralelo que en serie) aumenta el número de combinaciones de forma exponencial.

7.2.3 Paso 3: Optimización y resolución

Este es el último paso en la evaluación de los Algoritmos de Reconfiguración propuestos. En él se ha realizado una herramienta analítica capaz de elegir por sí misma el algoritmo más eficiente para realizar la reconfiguración de un sistema (en caso de que caída de alguno de los servicios de los que se compone) en función de ciertos parámetros introducidos por el usuario en cuestión, consiguiendo además que todo se realice de forma transparente para el mismo.

Para ello se han tenido en cuenta todas las reglas obtenidas en el apartado anterior, las cuales han sido generadas tras un estudio previo de varias de las estructuras posibles que podría tener una aplicación distribuida de un sistema de tiempo real.

Por lo tanto, teniendo en cuenta todas las conclusiones sacadas, el diseño final que se ha programado se basa en dos parámetros obligatorios, los cuales además deben ser introducidos por el usuario:

- Estructura de la aplicación.
- Margen de error (introducido en tanto por ciento - %).

Por otro lado, existen otros dos parámetros importantes, pero sobre los cuales el usuario no necesita introducir ningún valor:

- Tiempo de reconfiguración: se considerará siempre el mejor tiempo de reconfiguración que pueda ofrecerse dentro del margen de error introducido. Por lo que no es un parámetro obligatorio para el usuario.
- Cantidad de nodos caídos en el sistema y ubicación dentro de la estructura (éste será un parámetro impuesto por el programador para la comprobación del correcto funcionamiento del prototipo, en la realidad el sistema se reconfiguraría sabiendo únicamente los valores anteriores, puesto que no se sabe a priori la cantidad de servicios que se van caer en la aplicación).

Otro de los parámetros importantes, y que se explicó previamente en el Capítulo 5, es el *time-out*, Dicho parámetro no es necesario que sea introducido por el usuario, puesto que el mismo varía cuando la herramienta estudia la estructura de la aplicación que se va a analizar, tomando así, valores más elevados para estructuras más complicadas, y más bajos para las menos complicadas.

Como ya se comentó en el *Capítulo 5*, el valor del *time-out* que se ha supuesto en este proyecto, está basado en la relación directa que existe entre el tiempo de ejecución y el número de combinaciones, por ello, si se decrementa el número de combinaciones, se decrementa el tiempo de ejecución. La siguiente fórmula aplica este principio, por lo que, sabiendo el número de combinaciones que se analizan, se puede especular acerca del tiempo que puede tardar en ejecutarse, y por lo tanto limitarlo de esta manera.

$$time\ out = \frac{Num\ comb}{600}$$

Este valor consigue reducir el número de combinaciones posibles que se analizan, disminuyendo así el tiempo de ejecución de los algoritmos.

A continuación se resumirá el diseño de la herramienta creada, además de probar su funcionamiento con una estructura diferente a las que ya se han visto en puntos anteriores.

7.2.3.1 Diseño final

En este apartado se centrará la atención en el diseño que se ha realizado para la creación de una herramienta analítica capaz de dotar de flexibilidad en tiempo de ejecución a un sistema de tiempo real, consiguiendo la reconfiguración del mismo en caso de que algún perfil deje de estar disponible.

Se ha creado una nueva función en la que se engloban todos los algoritmos diseñados en este proyecto, y la cual será capaz de decidir en cada momento qué algoritmo es más eficiente para cada situación.

A continuación se explicará paso por paso el funcionamiento de dicha función, la cual se muestra en forma de diagrama de bloques en la Figura 56:

1. Esta función recibe por parámetro la estructura de la aplicación y el margen de error que no se debe superar en caso de que hubiera que realizar una reconfiguración del sistema. Con el parámetro de la estructura, además se sabe qué servicios forman la aplicación, las implementaciones que tiene cada uno y las redes que unen cada uno de los nodos.

2. A continuación se calculan la cantidad de servicios del sistema y cuántos de ellos han dejado de estar disponibles. A la vez que hallar el tanto por ciento de servicios caídos totales.

3. El siguiente paso es clasificar las estructuras, y luego dentro de cada una, seleccionar qué algoritmo sería más conveniente aplicar:

    a. Aplicaciones con pocos servicios. Aquí sólo se considerarían aplicaciones que tengan 3 o menos servicios, y la clasificación sería la siguiente:

        i. Si el error tiene que ser 0 obligatoriamente, sólo se puede utilizar el algoritmo exhaustivo, que para este tipo de estructuras no ofrece mal rendimiento en cuento a tiempos de ejecución. Algoritmo 1.

        ii. Si sólo se ha caído 1 servicio, el error no tiene por qué ser cero, se elegiría el algoritmo de reconfiguración exhaustivo, puesto que es el que mejor relación tiempo de ejecución/error tiene. Algoritmo 3.

        iii. Si se ha caído más de 1 servicio, entonces el algoritmo más conveniente en cuanto a relación tiempo de ejecución/error es el algoritmo de reconfiguración heurístico. Algoritmo 4.

    b. Aplicaciones con varios servicios. Es decir aplicaciones con más de 3 servicios en serie. La clasificación sería la siguiente:

        i. Si el error tiene que ser 0 obligatoriamente, sólo se puede utilizar el algoritmo exhaustivo, aunque para este tipo de estructuras sea poco eficiente en cuanto a tiempos de ejecución se refiere. Algoritmo 1.

        ii. Si el tanto por ciento de servicios caídos no supera el 60%, se elegiría el algoritmo de composición heurístico. Algoritmo 2.

        iii. Si el tanto por ciento de servicios caídos supera el 60% y no rebasa el 80%, entonces se elegiría el algoritmo de reconfiguración exhaustivo. Algoritmo 3.

        iv. Si el tanto por ciento de servicios caídos supera el 80%, entonces el algoritmo más eficiente es el algoritmo de reconfiguración heurístico. Algoritmo 4.

    c. A continuación, se ejecuta el algoritmo que se haya seleccionado en la clasificación anterior, de forma que:

- El algoritmo 1 es el Algoritmo de Composición Exhaustivo.
- El algoritmo 2 es el Algoritmo de Composición Heurístico.
- El algoritmo 3 es el Algoritmo de Reconfiguración Exhaustivo.
- El algoritmo 4 es el Algoritmo de Reconfiguración Heurístico.

d. Para finalizar, la función devuelve la combinación óptima y el tiempo de respuesta.

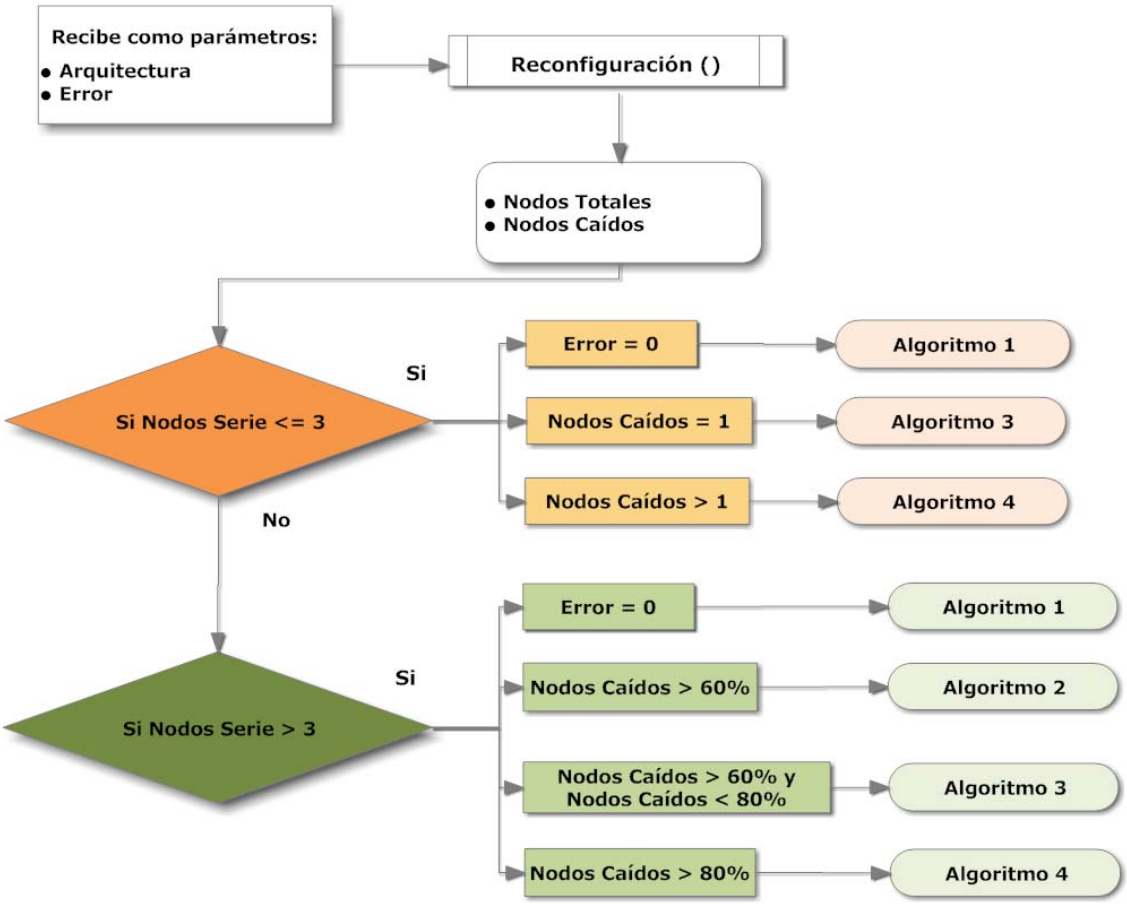


Figura 56. Diagrama de bloques Herramienta Analítica de Reconfiguración

7.2.3.2 Pruebas del funcionamiento de la Herramienta Analítica

Se ha elegido una estructura como la que se muestra en la Figura 57, y que además, no se ha estudiado en los anteriores apartados:

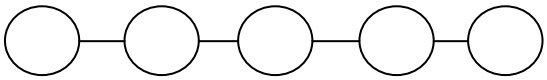


Figura 57. Reconfiguración - Estructura Prueba para la Herramienta Analítica

Se trata de una estructura con 5 servicios en serie, sobre la cual se observará si las normas tomadas en el apartado anterior, e introducidas en la realización del diseño del programa final son acertadas. Para ello se seguirán las normas relativas a una estructura con más de 3 servicios, que serían las pertenecientes a la estudiada anteriormente como *Aplicación 2*.

Así pues, teniendo la estructura y el margen de error, se procede a ver qué ocurre con la caída consecutiva de los servicios. Todos los resultados se muestran en la Tabla 18, en la cual se puede ver el número de servicios que han dejado de estar disponibles, los algoritmos elegidos para la reconfiguración, el número de combinaciones analizadas, los tiempos de reconfiguración, y el error introducido respecto de la figura de mérito.

En la prueba realizada, después de introducir los parámetros necesarios (estructura y margen de error), se ha empezado a provocar la caída de los servicios, permitiendo así que alguno de los algoritmos diseñado fuera el encargado de ejecutar la reconfiguración del sistema. Para ello se han utilizado las reglas marcadas en el apartado anterior, y además, para esta prueba, el parámetro más utilizado ha sido el *tanto por ciento de nodos caídos*, puesto que con los análisis que ya se habían realizado en puntos anteriores, se podía predecir también el error aproximado que se produciría.

Núm. de Servicios caídos	**Algoritmo**	**Núm de Comb.**	**Tiempo de ejecución**		**Error**	
			Media	**Var.**	**Media**	**Var.**
1	Composición Heurístico	32	0,098755	6,589E-05	2,4	2,74E-02
2	Composición Heurístico	32	0,108559	9,789E-05	2,7	1,41E-02
3	Reconfig. Exhaustivo	64	0,160272	2,0329E-4	1,3	5,49E-02
4	Reconfig. Heurístico	16	0,049124	8,856E-05	3,1	8,65E-02

Tabla 18. Resultados de la Herramienta Analítica para Reconfiguración.

En la Figura 58 se puede ver la evolución de los tiempos de ejecución. Si se comparan éstos con los obtenidos en las pruebas anteriores, se puede observar cómo los tiempos de reconfiguración siguen un patrón lineal y reducido ante la caída de cualquier servicio de los que se compone.

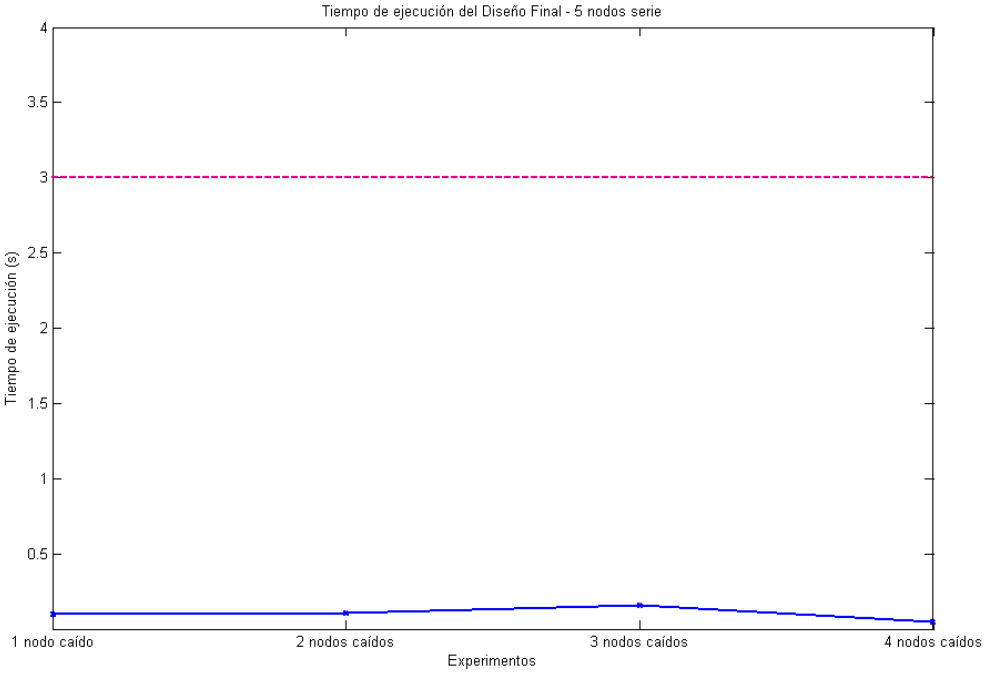


Figura 58. Tiempos de reconfiguración - Diseño Final.

Se puede ver como los tiempos de reconfiguración son mucho menores que la media de lo que hubiera tardado en reconfigurar el algoritmo de composición exhaustivo, la cual está marcada en la Figura 58 como una línea recta roja discontinua.

Es evidente que todos los tiempos de ejecución son sustancialmente más bajos de lo que el algoritmo con la solución más óptima puede ofrecer, pero esto conlleva un aumento del error, el cual se muestra en la Figura 59.

A pesar de que el error es difícil de evitar, sí que se puede minimizar, y es lo que se consigue con esta herramienta, minimizar el error frente a mejorar el tiempo de reconfiguración, escogiendo el algoritmo más eficiente en cada momento.

En la Figura 59 se ha dibujado el error respecto de la figura de mérito global para cada uno de los experimentos, mientras que con una línea discontinua roja se ha marcado cuál

es el margen error introducido por el usuario y el cual no supera en ningún momento el 4%.

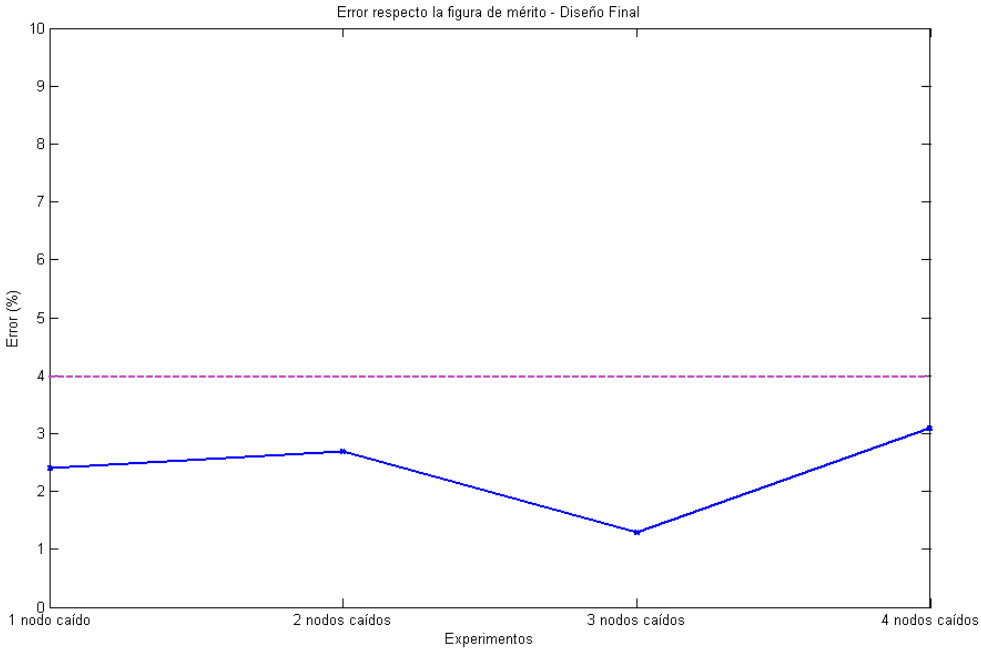


Figura 59. Error de Alg. Reconfig. respecto de la figura de mérito global - Herramienta Analítica.

Por lo tanto, y finalizando con este apartado, se puede apreciar, a través de esta pequeña prueba, el buen funcionamiento de la herramienta analítica que se ha diseñado, cumpliendo así con todos los requisitos que se pedían de él. Generando así buenos resultados tanto si hablamos de tiempos de reconfiguración, como de error cometido respecto del resultado más óptimo.

7.3 Conclusiones

En este apartado se han realizado la evaluación y pruebas de todos los algoritmos desarrollados en este proyecto, tanto los algoritmos de composición como los algoritmos de reconfiguración.

Al comienzo del capítulo se ha realizado una breve introducción acerca de qué pruebas se iban a realizar sobre los algoritmos de implementados en el proyecto, así como las decisiones de diseño que se asumían en los sistemas a analizar.

Posteriormente se ha realizado un estudio sobre la reconfiguración de un sistema en caso de que alguno de los perfiles de la aplicación hubieran dejado de estar disponibles. En dicho estudio se consiguieron encontrar una serie de normas que permitían decidir qué algoritmo era más válido utilizar para realizar la reconfiguración de un sistema, en función de parámetros como el tipo de estructura o la cantidad de servicios caídos.

En este estudio se tenían en cuenta tanto los algoritmos de composición como los algoritmos de reconfiguración, puesto que dependiendo de los parámetros indicados anteriormente la relación entre el tiempo de ejecución y el error era mejor o peor.

Como ejemplo se puede poner las aplicaciones compuestas por pocos servicios, las cuales, en función del número de servicios que hayan dejado de estar disponibles, pueden escoger como mejor opción la reconfiguración mediante el algoritmo de composición exhaustivo, puesto que con este algoritmo no se tiene que asumir ningún error, y teniendo en cuenta que es una aplicación pequeña, el tiempo que tarda en encontrar la solución más óptima no es excesivamente alta.

Se puede concluir que para aplicaciones con varios servicios, el algoritmo de composición exhaustivo, nunca es el más óptimo. Y si la cantidad de servicios que han dejado de estar disponibles es elevada, los algoritmos de reconfiguración tampoco trabajan todo lo eficientemente que se desea. Por lo que no hay un algoritmo único para poder realizar la reconfiguración, sino que, en función de cada situación será más óptimo utilizar uno u otro.

Además, cabe destacar, que el hecho de tener servicios en paralelo, no afecta de manera significante al aumento del error. Puesto que éste está muy ligado a la dispersión que exista en los tiempos de computación de los perfiles.

Por último, señalar también que todos los cálculos que se han realizado en este capítulo han sido hallados con algoritmos heurísticos de tamaños de bloque 2, lo cual también es determinante a la hora de hallar los errores introducidos.

Capítulo 8

Conclusiones y Líneas Futuras

En este capítulo se explican las conclusiones obtenidas a lo largo del desarrollo de este proyecto, así como una breve explicación de los algoritmos implementados en el mismo. Además, se listan los posibles trabajos futuros que se pueden realizar a partir de este proyecto.

8.1 Conclusiones

El principal objetivo que se ha perseguido en este proyecto, ha sido la realización de una herramienta analítica que permitiera la reconfiguración de aplicaciones distribuidas de tiempo real basadas en servicios.

Con ello se ha conseguido dotar de una mayor flexibilidad al análisis de este tipo de aplicaciones, permitiendo que pudieran reconfigurarse en tiempos de ejecución en caso de que algún perfil dejara de estar disponible en un momento determinado, consiguiendo así una planificación dinámica, en la cual las demandas al sistema cambiarán dinámicamente y se deberán poder modificar el conjunto de servicios que soportan y extender sus funciones para poder satisfacerlas.

En un principio, para poder cumplir con el objetivo propuesto, se consideró la opción de continuar con el proyecto realizado por Jorge Díez Sánchez [2], utilizando los algoritmos de composición implementados por él, pero tras la realización de un estudio de la

situación desde la que se debía partir, se tomó la decisión de implementar unos nuevos algoritmos de composición sobre los que trabajar, puesto que las estructuras de datos sobre las que se basaba la implementación de los algoritmos de [2] no eran lo suficientemente flexibles para poder diseñar una herramienta de reconfiguración eficiente.

Por esta razón, se implementaron dos nuevos algoritmos de composición (exhaustivo y heurístico) reutilizando parte del código realizado en [2]. El algoritmo de composición exhaustivo permite generar y analizar todas las combinaciones posibles para así encontrar la más óptima respecto de una figura de mérito global. Y el algoritmo de composición heurístico permite generar y analizar un número limitado de combinaciones y evalúa que combinación de las que se ha generado es la más óptima; asimismo, primeramente ordena debidamente los perfiles de servicio en función de una figura de mérito relativa, la cual está ligada a la figura de mérito global que se toma como referencia.

Se realizó una evaluación de los algoritmos de composición para comprobar el buen funcionamiento de los mismos, analizando tanto los tiempos de ejecución, como el error respecto de la figura de mérito global (en este caso, se ha considerado que la figura de mérito global es la minimización del tiempo de respuesta extremo a extremo). Además de evaluar la nueva figura de mérito relativa que se había implementado (maximización del tiempo restante antes del *deadline*), comparándola con la ya implementada en [1] y [2] (Minimización del tiempo de ejecución en el peor caso).

Para poder llevar a cabo el objetivo principal por el que se realizaba este proyecto, es decir la reconfiguración de aplicaciones distribuidas, se implementó sobre las bases del esquema [10] un diseño que permitiera la reconfiguración de dichas aplicaciones utilizando los algoritmos de composición ya implementados, surgiendo así el *algoritmo de reconfiguración exhaustivo* y el *algoritmo de reconfiguración heurístico*.

Éstos utilizan, prácticamente, el mismo código que los algoritmos de composición correspondientes, la principal diferencia reside en que, una vez que hay que reconfigurar un sistema, la búsqueda se debe realizar sólo sobre los perfiles que componen la aplicación que hayan dejado de estar disponibles, limitando así el número de combinaciones, y por lo tanto, reduciendo el tiempo de ejecución del algoritmo. En caso de que la aplicación no pudiera planificarse con esas combinaciones, entonces se ampliaría la búsqueda de combinaciones utilizando los nodos adyacentes al caído, y así sucesivamente sobre el resto de los nodos hasta que se encontrara una combinación planificable o expirara el *time-out*.

Como se puede deducir, el proceso de reconfiguración y búsqueda de una nueva combinación que permita que la aplicación pueda ser planificable puede alargarse en el tiempo más de lo conveniente o lo deseable, por lo que se ha delimitado este tiempo con un nuevo parámetro denominado *time-out*. El *time-out* mantiene una relación directa con el tipo de estructura de la aplicación, puesto que si la aplicación es compleja el *time-out* a utilizar debe ser mayor, mientras que si la aplicación es más sencilla el *time-out* debe ser menor. Por lo que la elección del *time-out* puede realizarse de dos maneras: hacer que el mismo algoritmo, considerando los parámetros más importantes, halle el valor de forma automática, o forzar al usuario a introducir un tiempo límite en el que necesite que se realice la reconfiguración (en este proyecto se ha elegido la primera).

Para evaluar el proceso de reconfiguración se creó una herramienta que permitiera probar, bajo las mismas condiciones del sistema (carga de la red o los nodos), las prestaciones de cada uno de los algoritmos implementados en este proyecto analizando diferentes estructuras de aplicaciones, indicando para los mismos qué perfiles de la aplicación habían dejado de estar disponibles.

Como es obvio pensar, el algoritmo que más tardaba en reconfigurar el sistema era el algoritmo de composición exhaustivo, mientras que el que menos era el algoritmo de reconfiguración heurístico. Asimismo, el primer algoritmo citado no introducía ningún error, mientras que el segundo era el que más introducía, debido, principalmente, a la diferencia del número de combinaciones que analizaba cada uno.

Por lo tanto se propuso escribir una serie de normas, o reglas, en las que se clasificaban los algoritmos a utilizar en cada aplicación según la eficiencia mostrada en la reconfiguración. Dicha eficiencia era determinada en función del tiempo de ejecución y del error respecto la figura de mérito introducido por ellos.

Con estas normas se desarrolló una pequeña herramienta analítica que realizaba la elección del algoritmo más eficiente para la reconfiguración de una aplicación distribuida basada en servicios en función de la estructura de la aplicación y la cantidad de perfiles que componen la aplicación que hayan dejado de estar disponibles. De este modo, considerando si la aplicación era más compleja o menos, si el número de perfiles que componen la aplicación que habían dejado de estar disponibles era mayor o menor, o el error que pudiera asumir el sistema era más alto o más bajo, se elegía uno de los cuatro algoritmos siguientes:

- Algoritmo de composición exhaustivo.

- Algoritmo de composición heurístico.
- Algoritmo de reconfiguración exhaustivo.
- Algoritmo de reconfiguración heurístico.

Después de mostrar una visión general acerca de los algoritmos implementados en este proyecto, se ha concluido lo siguiente:

- El *algoritmo de composición exhaustivo*: evalúa todas las combinaciones posibles, lo cual permite obtener el resultado más óptimo. Pero esto conlleva un coste adherido, y es que el tiempo que tarda en encontrar la mejor solución es excesivamente alto para tenerlo en consideración en sistemas de tiempo real, en los que uno de los parámetros más importantes a tener en cuenta es el cumplimiento de los plazos, el cual está ligado, en numerosas ocasiones a la rapidez en tiempos de ejecución.
- El *algoritmo de composición heurístico*: evalúa un número limitado de combinaciones, y por lo tanto el tiempo que tarda en ejecutarse es menor que el que tarda el algoritmo exhaustivo. Sin embargo, igual que en el anterior caso, lleva un coste incluido, y es que la solución que es capaz de encontrar este algoritmo no tiene por qué ser la óptima, sino una aproximación a ella, introduciendo así un error sobre la figura de mérito global elegida.
- El *algoritmo de reconfiguración exhaustivo*: es capaz de realizar la reconfiguración de una aplicación *on-line*, dotando así de mayor flexibilidad al sistema. Realiza una búsqueda de nuevas combinaciones considerando únicamente los perfiles que componen la aplicación que hayan dejado de estar disponibles en el sistema, y además debe dar una solución en un tiempo limitado. Por ello, en caso de que el número de perfiles no disponibles sea pequeño, es más rápido en tiempos de ejecución que los algoritmos de composición anteriores. Pero, al igual que el algoritmo anterior, el coste adherido de este sistema es el error que introduce, puesto que este algoritmo devuelve la primera combinación planificable que encuentra.
- El *algoritmo de reconfiguración heurístico*: es capaz de realizar la reconfiguración de una aplicación *on-line*, dotando así de mayor flexibilidad al sistema. Este algoritmo utiliza partes del algoritmo de composición heurístico y partes del algoritmo de reconfiguración exhaustivo, puesto que realiza una búsqueda de

nuevas combinaciones considerando únicamente los perfiles que componen la aplicación que hayan dejado de estar disponibles en el sistema, pero limitando el número de perfiles a analizar, reduciendo así, aún más, el número de combinaciones a examinar. De entre todos, este algoritmo es el más rápido a la hora de reconfigurar un sistema, aunque también es con el que más error se debe asumir, produciendo que no siempre sea el más efectivo.

Es importante destacar que el error resultante de las pruebas realizadas es muy variable en función de varios parámetros, como pueden ser la dispersión de los valores de los tiempos de computación de la tareas, el tamaño de bloque utilizado para los algoritmos heurísticos, o la cantidad de perfiles que tenga cada servicio de la aplicación.

8.2 Líneas Futuras

En respuesta al progresivo aumento en el uso de sistemas con requisitos temporales, y continuando con los estudios realizados por *Iria Estévez Ayres* [1] o *Jorge Díez Sánchez* [2] en relación con el paradigma de las aplicaciones distribuidas de tiempo real basadas en servicios, se ha realizado este proyecto, y sobre los que todavía se pueden abrir muchas líneas de investigación, por ejemplo, creando algoritmos aún más eficientes.

Por ello, cuando se realizó la implementación de los algoritmos de este proyecto se intentó que el desarrollo del código de los mismos fuera lo más estructurada posible, permitiendo así que otros proyectos pudieran basarse en él.

Posible Trabajos Futuros

- Mejoras de los algoritmos de composición y reconfiguración implementados. O implementación de unos nuevos algoritmos.
- Mejorar la implantación del *time-out* a la hora de simular los algoritmos, permitiendo que el proceso de reconfiguración pueda finalizar en el mismo momento en el que éste es superado, bien con interrupciones o con otros agentes externos.
- Realización de un estudio del valor del *time-out* más adecuado en función de las características de la aplicación a ser reconfigurada.

- La implementación de algoritmos con otras figuras de mérito que tengan en cuenta otros parámetros del sistema.

- El estudio del comportamiento de los algoritmos dependiendo de la situación del sistema, o de la calidad del procesador en el que se ejecute.

- La creación de una herramienta de simulación mediante interfaz gráfica, que permita a un usuario crear una aplicación basada en servicios, y elegir los algoritmos para los que se desea analizar el comportamiento.

Apéndice

Presupuesto del Proyecto

En este apéndice se muestra el estudio económico y la planificación estimada de las diferentes fases que componen este proyecto fin de carrera.

El estudio económico del proyecto está íntimamente ligado a la cantidad de horas necesarias para realizar cada una de las fases de las que se compone, por lo que primeramente se realizará una explicación de las mismas, para luego entrar en detalle acerca de la cantidad de horas que se han necesitado para cada una de ellas.

A. Fases del Proyecto

Este proyecto se compone de diferentes etapas, las cuales se listan a continuación en orden cronológico:

Fase de búsqueda de información y análisis. Estudio y análisis del problema. Búsqueda bibliográfica relacionada con los sistemas de tiempo real y métodos de planificación de tiempo real con especial énfasis en aplicaciones de tiempo real basadas en servicios. Investigación de las herramientas existentes actualmente para la planificación de sistemas de tiempo real.

Fase de implementación. Implementación de algoritmos de composición y de reconfiguración de aplicaciones distribuidas de tiempo real utilizando el programa

MATLAB para la realización de los mismos. Así como el desarrollo de las herramientas necesarias para probar el funcionamiento de los mismos.

Fase de pruebas. Evaluación y validación de los algoritmos utilizando las herramientas desarrolladas para el efecto. Simulación de un sistema de tiempo real forzando la reconfiguración del mismo en tiempo de ejecución. Evaluación de los resultados obtenidos.

Fase de documentación: Documentación en forma de memoria de cada una de las fases anteriores.

En la **Tabla 19** se muestra el número de horas que se ha destinado a cada una de las etapas de las que se ha compuesto este proyecto.

Fase	Nº de horas
Búsqueda de información y análisis	220
Implementación	400
Pruebas	180
Documentación	160

Tabla 19. Fases del Proyecto

B. Costes del Proyecto

En este proyecto se tendrán en cuenta tanto los costes derivados de personal, como los derivados de material. A continuación se explica cuáles han sido las necesidades a cubrir para cada uno de ellos, mostrando detalladamente en tablas el desglose de los mismos.

Costes de Personal

Los costes en cuanto al personal que ha participado en el proyecto se dividen de la siguiente forma, quedando reflejado de forma desglosada en la **Tabla 20**:

- Ingeniero Técnico: Ha trabajado un total de 940 horas. Debe tenerse en cuenta que la tarifa acordada sobre el honorario de un Ingeniero Técnico toma asciende a 35 €/h.

- Jefe de Proyecto: Ha trabajado un total de 160 horas. Como Jefe de Proyecto, los honorarios son mayores que los de un Ingeniero Técnico, y la tarifa acordada asciende a 60 €/h.

Personal	**Nº de horas**	**Coste €/h**	**Coste total (€)**
Jefe de Proyecto	160	60	9.600
Ingeniero Técnico	940	35	32.900
		TOTAL	**42.500 €**

Tabla 20. Costes derivados de personal

Costes de Material

Los costes en cuanto al material que ha participado en el proyecto se dividen de la siguiente forma, quedando reflejado de forma desglosada en la **Tabla 21**:

Material	**Cantidad**	**Coste por unidad (€)**	**Coste total (€)**
Licencia Software	1	6.000	6.000
PC's	3	1.299	3.897
Switch	3	24,95	74,85
Cable (m)	15	1,77	26,55
		TOTAL	**9.998,4 €**

Tabla 21. Costes derivados de material

C. Presupuesto final

El presupuesto total de este proyecto se deriva de la suma de los dos costes detallados en las tablas anteriores (tanto los costes derivados de personal como los costes derivados de material).

Coste	Coste total (€)
Personal	42.500
Material	9.998,4
	52.498,4 €

Tabla 22. Presupuesto final del diseño.

De este modo, la suma de todos los costes asciende hasta la cantidad de: 52.498,4 €, tal como se detalla en la **Tabla 22**.

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] Jorge Diez Sánchez *Implementación y evaluación de algoritmos de composición de aplicaciones distribuidas de tiempo real basadas en servicios*. Proyecto Fin de Carrera, Universidad Carlos III de Madrid, Marzo 2010.

[3] COIT. Colegio de Ingenieros de Telecomunicaciones. http://www.coit.es.

[4] H. Leung V.C.M.D.W. Gillies, Le Pocher. *Real time multimedia scheduling policies for end to end delay jitter and loss guarantees across atm satellite systems*. IEE Transactions Selected Areas en Comunications, 17(2), 1999.

[5] Alan Burns and Andy Wellings. *Real-Time Systems and Programming Languages (Fourth Edition).* Ada 2005, Real-Time Java and C/Real-Time POSIX.

[6] K. Schwan, W. Bo, and P. Gopinath. *A high performance, object-based operating system for real-time robotics application.* In Proceedings of the IEEE Real –Time Systems Symposium, December, 1986.

[7] J. Blazewicz et al. *Scheduling in Computer and Manufacturing Systems*. Springer – Verlag, 1993.

[8] C. L. Liu and J. W. Layland, *"Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment"* Journal of ACM, Vol. 20, No. 1, pp. 46-61, 1973.

[9] J. Leung and J.W. Whitehead. *On the complexity of fixed priority scheduling of periodic real-time tasks*. Performance Evaluation, 2(4), 1982.

[10] Iria Estévez-Ayres, Marisol García-Valls and Pablo Basanta-Val. *On the reconfiguration of service-based real-time applications*. IEEE RTAS-2010 WiP Session, Suecia, 12-15 de Abril 2010

[11] http://docencia.izt.uam.mx/sgm8/nos/sistemas%20distribuidos%2001.pdf

[12] Mario Calha. A Holistic *Approach Towards Flexible Distributed Systems*. PhD thesis, DET / IEETA-LSE. Universidade de Aveiro, 2006.

[13] W. Horn. *Some simple scheduling algorithms*. Naval Research Logistics Quarterly, 21, 1974.

[14] 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.

[15] Ken Tindell. *Adding Time_O_sets to Schedulability Analysis*. Technical Report YCS221, Department of Computer Science, University of York, England, 1994.

[16] Robert Davis and Andy J. Wellings. *Dual Priority Scheduling*. In IEEE Real-Time Systems Symposium, pages 100_109, 1995

[17] 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.

[18] 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.

[19] Propiedades de MATLAB. *http://es.wikipedia.org/wiki/MATLAB*.

[20] J.A. Stankovic, M. Spuri, M. Di Natale, and G. Buttazzo. *Implications of classical scheduling results for real-time systems*. IEEE Computer, 28(6), June 1995.

[21] M. Satyanarayanan. *Pervasive computing: vision and challenges*. IEEE Personal Communications, 8(4):10_17, August 2001.

[22] M. H. Huhns and M. P. Singh. *Service-oriented computing: Key concepts and principles.* IEEE Internet Computing, 9(1):75_81, January/February 2005.

[23] R. Garey and S. Johnson. *Complexity results for multiprocessor scheduling under resource constraints*. SIAM Journal of Computing, 4(4):397–411, 1975.

[24] Jose Carlos Palencia. *Análisis de planificabilidad de sistemas distribuidos de tiempo real*. PhD thesis, Grupo de Computadores y Tiempo Real. Universidad de Cantabria, 1999.

[25] M. H. Huhns and M. P. Singh. *Service-oriented computing: Key concepts and principles*. IEEE Internet Computing, 9(1):75–81, January/February 2005.

[26] Keneth William Tindell. *Fixed Priority Scheduling of Hard Real–Time Systems*. PhD thesis, University of York, 1993.

[27] http://laurel.datsi.fi.upm.es/~ssoo/STR/