Facultad de Ciencias

Modelado y análisis de tiempo de respuesta del software de misión y control de un dron

Modeling and response time analysis of drone mission and control software

Trabajo de Fin de Máster
para acceder al

MÁSTER EN INGENIERÍA INFORMÁTICA

Autor: Jorge Polanco Peña

Director: J. Javier Gutiérrez García

Octubre – 2020

Resumen

En 2018, como parte del congreso ECRTS (*30th Euromicro Conference on Real-Time Systems*) se organizó el WATERS (*9th Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*), en el que se planteó un *challenge* industrial. El *challenge* consiste en aportar una solución, tanto hardware como software, del diseño de un dron. Basándose en un caso de uso del proyecto industrial europeo RESACC sobre certificación aeronáutica.

El dron se diseñó para transportar paquetes de hasta 10 Kg y debe ser capaz de volar de manera autónoma o por control remoto. Como se quiere proporcionar una mayor fiabilidad y disponibilidad, el dron tiene redundado todos sus componentes, aumentando así la complejidad de su arquitectura. Esta arquitectura debe cumplir ciertos requisitos de tiempo para poder realizar las operaciones de control del dispositivo y de la misión.

El objetivo del TFM es responder parcialmente a este challenge, modelando y realizando un análisis temporal de extremo a extremo del sistema teniendo en cuenta las especificaciones que se han proporcionado. A la hora de realizar el modelado, se utilizará la política de planificación por prioridades fijas.

Recientemente se ha desarrollado un método de cálculo de tiempos de respuesta, más preciso que los anteriores, que puede ser aplicable al análisis del software de este dron. Esta técnica se ha incorporado a la herramienta de modelado y análisis MAST desarrollada por el grupo ISTR de la UC. Por lo tanto, se propondrá una batería de test para MAST con el objetivo de contrastar los resultados de la técnica de análisis nueva con la ya existente.

Palabras clave: tiempo real, dron, análisis de tiempos de respuesta, modelado, planificación por prioridades fijas, 1oo2

Abstract

In 2018, as a part of the 30th Euromicro Conference on Real-Time Systems (ECRTS), the 9th Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS) was organized, and an industrial challenge was proposed. The challenge consist on getting a solution, both hardware and software, for the design of a drone. Based on a use case from RESSAC European industrial project on aeronautical certification.

The drone was designed to transport packages of up to 10 kg and must be able to fly autonomously or by remote control. One of his objectives is to provide greater reliability and availability, and for that, the drone has all its components redundant (increasing the complexity of its architecture). This architecture must guarantee certain timing requirements in order to perform device and mission control operations.

The objective of this TFM is to address partially to this challenge, modeling and performing an end-to-end timing analysis of the system taking into account the specifications that have been provided. The fixed priority scheduling policy will be used for this model.

Recently a response time analysis method has been developed, it is more precise than the previous ones, and it can be applicable to the analysis of the software of this drone. This technique has been implemented into the MAST tool, developed by the ISTR group of the UC. Therefore, a benchmark battery for MAST will be proposed in order to contrast the results of the new analysis technique with the existing one.

Keywords: real time, drone, response time analysis, modeling, fixed priority scheduling, 1oo2

Agradecimientos

Quiero agradecer a todos los profesores del grado y del master de ingeniería informática de la Universidad de Cantabria por todo el esfuerzo que emplean para formar a los futuros ingenieros. Además, sin ellos no habría descubierto un mundo que tanto me apasiona.

En concreto, me gustaría agradecer a J. Javier Gutiérrez García por haberme guiado y ayudado durante este tiempo.

Por último, me gustaría dar las gracias a mi familia y amigos, porque siempre han estado ahí y me han ayudado a superar todos los obstáculos que han ido surgiendo a lo largo de este trayecto.

Índice

Índice de ilustraciones

Índice de tablas

1 Introducción

Hoy en día dependemos de la tecnología constantemente y aunque nosotros no seamos conscientes de ello, utilizamos una gran variedad de sistemas en nuestro día a día. Algunos de estos sistemas pueden ser: el control de inyección o la navegación de un vehículo, el control de los semáforos de una ciudad, la Bolsa o el sistema de seguridad de una casa.

Algunos de estos sistemas que se han comentado anteriormente son sistemas de tiempo real. Esto significa que su correcto funcionamiento no depende solo de que proporcione una solución acertada, sino de que lo haga dentro del plazo temporal exigido [1].

El hecho de que un sistema de tiempo real no funcione correctamente tiene consecuencias. La gravedad de estas consecuencias dependerá de la criticidad del sistema. Por ejemplo, si el sistema de navegación de un avión falla y está activado el piloto automático, el avión podría estrellarse contra un edificio y acabar con la muerte de los pasajeros.

Sin embargo, no todos los sistemas de tiempo real son tan críticos como el de un avión. Por ejemplo, tenemos los robots aspiradora. Mientras aspiran el suelo de una habitación calculan las dimensiones de esta. Un fallo en este sistema implica que una zona del suelo no sea aspirada, en el peor de los casos, el robot dejaría de funcionar.

Debido a la gran cantidad de sistemas de tiempo real, existen diferentes tipos de arquitecturas, dependiendo del número de procesadores y de la forma en que se comunican, podemos distinguir tres tipos:

- Sistemas monoprocesadores. Solo existe un procesador que se encarga de ejecutar todas las actividades del sistema.
- Sistemas multiprocesadores. Están compuestos por varios procesadores y cada uno de ellos cumple un conjunto determinado de actividades. Para comunicarse se suele utilizar mecanismos como la memoria compartida o buses de datos.
- Sistemas distribuidos. Están formados por dos o más computadores (que disponen de uno o varios procesadores) que se comunican entre sí utilizando redes de comunicación.

Antes de implementar completamente uno de estos sistemas de tiempo real, y como parte del proceso de desarrollo de software, es conveniente modelarlo y analizarlo para comprobar si es planificable. Para el modelado y análisis hay que tener en cuenta una serie de conceptos, los cuales se describen a continuación.

1.1 Conceptos básicos

- Tarea o actividad. *Thread* o hilo de ejecución que ejecuta una función del sistema.
- Flujos de extremo a extremo (del inglés *end-to-end flow*), en adelante flujo e2e. Conjunto de tareas que se ejecuta de forma secuencial y que tiene como objetivo cumplir una funcionalidad del sistema. Su activación puede ser periódica o esporádica con un intervalo mínimo entre activaciones (MIT, *Minimum Interarrival Time*) entre llegadas.
- Tiempo de ejecución de una tarea. Suele proporcionarse el peor tiempo de ejecución, también denominado WCET (*Worst-Case Execution Time*).
- Tiempo de respuesta. Tiempo que tarda una tarea en ejecutarse desde que se lanza su evento de activación. Dado que este valor no es fijo, en los análisis se suele presentar el peor tiempo de respuesta.
- *Deadline* o plazo. Tiempo máximo de respuesta permitido.
- *Jitter*. Variabilidad en la que activación real de una tarea respecto a su evento de activación.
- *Offset*. Cantidad de tiempo que debe pasar (como mínimo) hasta que una tarea pueda comenzar su ejecución tras la activación de un evento.
- *Slack* del sistema. Porcentaje que en caso de ser positivo, indica cuánto se pueden incrementar los tiempos de ejecución de todas las tareas sin que el sistema deje de ser planificable. Y en el caso de ser negativo, indica cuánto deben reducirse los tiempos de ejecución de todas las tareas para que el sistema sea planificable.

1.2 Modelado y análisis

Para asegurarse de que el sistema que se está diseñando es planificable, lo mejor es desarrollar un modelo que contenga toda la información relevante del mismo, de forma que posteriormente se puedan aplicar diferentes técnicas de análisis para comprobar su planificabilidad. La OMG (*Object Management Group*) definió el estándar MARTE [2] (*Modeling and Analysis of Real Time and Embedded systems*) para cumplir con este objetivo.

Alineado con el estándar MARTE, el Grupo de Ingeniería Software y Tiempo Real de la Universidad de Cantabria ha desarrollado MAST [3] [4] [5], que es un conjunto de herramientas *open-source* que te permite definir un modelo de un sistema de tiempo real, analizarlo y optimizarlo. Una de estas herramientas es la de análisis de planificabilidad, la cual nos permite utilizar diferentes técnicas de cálculo de tiempos de respuesta sobre un mismo modelo teniendo en cuenta la política de planificación que se ha definido.

1.2.1 Política de planificación

Actualmente existen varias políticas de planificación. Aunque las más conocidas son la política basada en prioridades fijas (*Fixed Priority*) y EDF (*Earliest Deadline First*). En este informe se hablará únicamente de la política basada en prioridades fijas, ya que es la que se ha utilizado en el desarrollo de este TFM.

Al utilizar una política basada en prioridades fijas hay que asignar una prioridad a todas las tareas del sistema. Cuanta mayor prioridad tenga una tarea, antes se ejecutará tras su evento de activación. Además, tenemos que tener en cuenta si la política es expulsora o no, y sí utiliza una planificación FIFO (*First In First Out*) o cíclica (*Round Robin*) para las tareas de la misma prioridad.

Que una política de planificación sea expulsora, implica que, sí se activa una tarea con mayor prioridad que la que se está ejecutando actualmente, el planificador expulsa esta tarea para empezar a ejecutar la de mayor prioridad.

Para desarrollar este TFM, utilizaremos una política basada en prioridades fijas expulsora y que utiliza una planificación FIFO para las tareas de la misma prioridad.

1.2.2 Técnicas de análisis

Para desarrollar este TFM se utilizarán cuatro técnicas, las cuales se describen a continuación:

- Holística. Toma una visión holística del sistema y analiza el sistema distribuido en su conjunto. Aunque considera a cada recurso de manera independiente. Esto provoca que los resultados que se obtienen de esta técnica sean pesimistas (se obtienen cotas superiores a los tiempos de respuesta reales de las tareas).

  Esta técnica fue desarrollada por Tindell y Clark [6], y refinada por Palencia [7].

- Basada en *offsets*. Las técnicas de análisis basadas en *offsets* tienen en cuenta que las tareas de un mismo flujo e2e no son independientes. Teniendo en cuenta eso, el algoritmo calcula los *offsets* de cada tarea y realiza el análisis. De esta forma se obtienen unos resultados mejores que en la técnica de análisis holística.

  Esta técnica realiza un análisis aproximado del modelo. Inicialmente fue desarrollada por Tindell [8], y posteriormente extendida a sistemas distribuidos por Palencia y González Harbour [9].

- Basada en *offsets* de fuerza bruta. Se basa en la técnica de offsets con la diferencia que realiza un análisis exacto del modelo (no es aconsejable utilizar este algoritmo en modelos complejos). También fue desarrollada por Tindell [8].

- Basada en *offsets slanted.* Es una versión mejorada del algoritmo basado en *offsets*. Para ello, se actualiza la función que se encarga de calcular la interferencia de tarea en otras de menos prioridad. Obteniendo así unos resultados menos pesimistas que en los anteriores algoritmos en algunos casos.

  Esta técnica fue desarrollada por Mäki-Turja y Nolin [10].

1.3 Aplicación de tiempo real

Este informe tiene como objetivo describir el trabajo realizado durante el TFM, que consiste en el modelado y análisis de una aplicación distribuida de tiempo real con requisitos de seguridad funcional. Dicha aplicación es un dron cuyo diseño está implementado parcialmente en el *challenge* WATERS 2018 [11] (*Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems*), el cual se llevó a cabo en el congreso ECRTS [12] (*Euromicro Conference on Real-Time Systems*) de 2018.

El sistema que se describe en el *challenge* se basa en un caso de estudio del proyecto industrial europeo RESSAC (*Re-Engineering and Streamlining the Standards for Avionics Certification),* que tiene como objetivo experimentar en el desarrollo de sistemas aeronáuticos certificados.

El dispositivo se diseñó para transportar cargas de hasta 10 Kg y debe ser capaz de volar de forma autónoma o por control remoto. La misión debe durar menos de 15 minutos y se debe garantizar el vuelo a pesar de las condiciones climáticas. Por lo tanto, debe asegurar alta disponibilidad, fiabilidad, robustez y eficiencia energética.

Además de estas características, se han propuesto los siguientes requisitos para reforzar la seguridad del dron:

- Un fallo no debe provocar la pérdida del dron.
- Ningún incidente o evento (periódico o aperiódico) debe provocar la pérdida de control del dispositivo.
- Si durante el transcurso de la misión las condiciones climáticas empeoran, se abortará la misión realizando un aterrizaje de emergencia para evitar la pérdida del dron y posibles daños o víctimas.

Teniendo en cuenta estos puntos, se ha diseñado el siguiente diagrama que representa los diferentes estados que puede tener el dron.

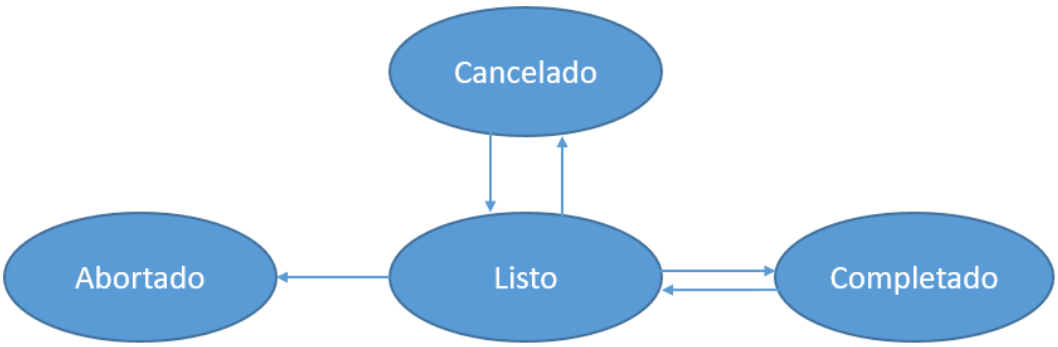

Ilustración 1: Estados del dron

Inicialmente, el dron comienza en el estado 'Listo'. En el momento que la estación de tierra envía los parámetros de vuelo e indica que comience la misión, el dron inicia el proceso de despegue. Si durante este proceso se detecta algún tipo de problema, se cancela la misión.

Una vez que comienza la misión, el dron tiene dos opciones:

- Completarla aterrizando en el lugar indicado y en buenas condiciones.
- Abortarla aterrizando en algún punto de la ruta. A la hora de realizar el aterrizaje el dron tiene dos opciones. Dependiendo de las condiciones (energía, fallos del sistema, condiciones climatológicas,…) decidirá realizar el aterrizaje normal o el de emergencia.

1.4 Objetivos del *challenge* WATERS 2018

El *challenge* plantea proponer una solución que responda a tres cuestiones:

- Escoger los componentes hardware de los diferentes componentes del dron. La elección de los recursos debe realizarse teniendo en cuenta que hay que minimizar el peso, el consumo de energía, el coste y el riesgo de obsolescencia del dispositivo.
- Realizar un análisis temporal de extremo a extremo. Para realizar el análisis será necesario generar el modelo del sistema teniendo en cuenta lo siguiente:
    - Las tareas del sistema y sus dependencias. El WCET de las tareas se proporciona en la documentación excepto cuando se proporciona el código fuente.
    - Los mensajes que se transmiten para establecer una comunicación entre los diferentes componentes del dron.
    - La política de planificación de las tareas.

Si uno o varios componentes fallan pueden provocar una reconfiguración de las tareas de los sistemas del dron, por lo que el análisis deberá realizarse teniendo en cuenta el peor caso.

- Generar simulaciones de los escenarios de tiempo crítico teniendo en cuenta los puntos anteriores.

1.5 Objetivos del TFM

Debido a la complejidad del *challenge*, en este TFM solo se desarrollará una de las cuestiones planteadas, la cual consiste en el desarrollo del análisis temporal de extremo a extremo. Para desarrollar esta cuestión, se diseñará el modelo de acuerdo a los requisitos temporales que se proponen en el *challenge* y se realizará el análisis temporal utilizando la herramienta MAST.

Como se podrá observar posteriormente, mientras se genera el modelo con la información que se proporciona en el *challenge*, se llega a la conclusión de que el sistema no es planificable. Por ese motivo, se presentan otras dos soluciones denominadas caso redundado lineal y caso redundado 1oo2.

Estos casos son una versión simplificada del sistema que se propone y contienen tres de los siete flujos e2e que se presentan en la documentación del *challenge*.

Mientras se realizan los análisis con el caso redundado 1oo2, se detecta que algunas de las técnicas de análisis de la herramienta MAST no calculan correctamente los tiempos de respuesta. Debido a esto, fue necesario revisar el código de la aplicación para corregir este bug. Tras generar la nueva versión de la herramienta, se comprobó que los cálculos se realizaban correctamente.

Para finalizar este trabajo, se decidió comprobar el funcionamiento de la herramienta MAST a la hora de procesar los flujos e2e. Para ello, se evaluaron las restricciones que se describen en la documentación proporcionada en la página web de la herramienta y se proponen otros *benchmarks* que prueban el funcionamiento de la aplicación.

2 Interpretación del modelo del *challenge*

En este apartado se expone el modelo propuesto por el *challenge* WATERS 2018. Se describirán componentes del sistema (tanto hardware como software), así como los diferentes flujos e2e que se ejecutan en el dispositivo.

Por último, se realizará un breve análisis comentando la viabilidad del modelo y exponiendo sus características.

2.1 Arquitectura del sistema

El sistema que se propone está formado por tres subsistemas, los cuales se describen a continuación:

- MMS (*Mission Management System*). Se encarga de controlar el funcionamiento de los otros subsistemas indicándoles las acciones que deben realizar. También procesa la información de los sensores, gestiona las comunicaciones con la estación de tierra, monitoriza el gasto de energía y el progreso de la misión.

  Por último, tiene la capacidad de determinar si la misión sigue adelante o no (abortar) realizando un aterrizaje normal o de emergencia.

- EPS (*Electrical Propulsion System*). Procesa los comandos de vuelo que recibe (por parte del MMS) en comandos de voltaje que envía a los motores eléctricos. Este subsistema se encuentra en modo esclavo con respecto a MMS y solo se convierte en maestro cuando MMS falla. Para determinar el momento en el que debe tomar el control del sistema, monitoriza el estado de MMS.

  Cuando EPS se convierte en maestro, intenta completar la misión con los parámetros de vuelo que ha registrado MMS (no puede comunicarse con la estación de tierra para recibir nuevos parámetros). Para realizar esto, procesa la información de los sensores, monitoriza el gasto de energía y el progreso de la misión. Con la información de la que dispone puede decidir continuar con la misión o no.

- HBS (*Hydraulic Bracking System*). Se encarga de reducir la altitud o velocidad del dron degradando la energía cinética. También se hace cargo del aterrizaje de emergencia intentado que la altitud y la velocidad del dron tengan el valor cero de forma simultánea. Solo puede gestionar los frenos del dispositivo y esto implica que dependiendo de las condiciones climatólogas, el dron disponga de cierta velocidad residual en el momento de aterrizar.

  Al igual que EPS, este subsistema está en modo esclavo y solo se convierte en maestro cuando los otros subsistemas fallan. Cuando se da este caso, el único objetivo de HBS es realizar un aterrizaje de emergencia teniendo en cuenta la información de los sensores, y el gasto de energía.

Como se ha comentado anteriormente, este sistema se basa en el modelo maestro-esclavo, siendo MMS el maestro y el resto de los subsistemas esclavos. En el caso de que MMS falle, EPS se convierte en maestro, y si se da el caso de que EPS también falla, HBS toma control.

Para evitar que un fallo en el sistema provoque una pérdida de funcionalidad, se han redundado todos los componentes esenciales del dispositivo (sensores, motores, frenos hidráulicos,…). Este detalle, hace que la complejidad del sistema aumente, ya que, en vez de tres subsistemas, hay seis (dos MMS, dos EPS y dos HBS). A continuación, se muestra una ilustración indicando la prioridad que tienen los subsistemas a la hora de tomar el control para convertirse en el subsistema maestro.

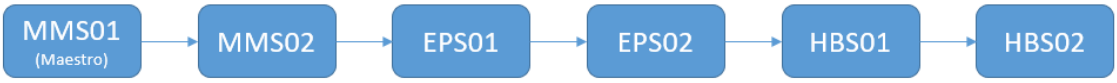

Ilustración 2: Toma de control de los subsistemas

Todos los subsistemas monitorizan al subsistema que se encuentra por delante para asegurarse de que no falla (MMS02 a MMS01, EPS01 a MMS02,…). Si se da el caso de que el subsistema al que están monitorizando falla, empiezan a monitorizar al siguiente (si EPS01 falla, EPS02 monitoriza a MMS02). Si se da el caso de que el subsistema maestro falla, el subsistema que le está monitorizando se convierte en el maestro.

Para representar esta arquitectura, se ha desarrollado el siguiente modelo AADL [13] con la herramienta OSATE [14]:

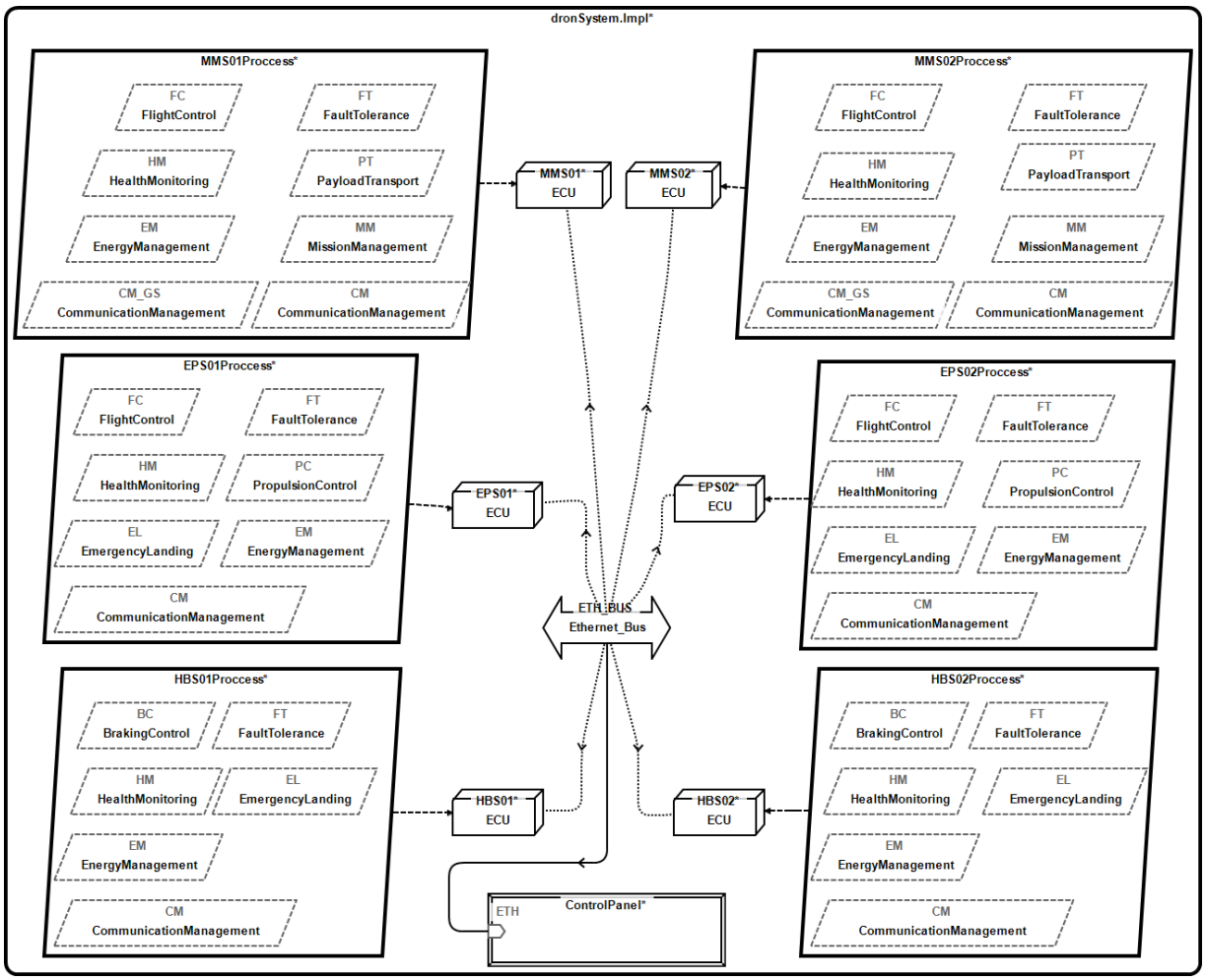

Ilustración 3: Arquitectura del sistema del *challenge*

En el modelo, solo se han representado los componentes que utilizan una señal digital para comunicarse. El resto de componentes (como por ejemplo, los sensores o los motores) utilizan una señal analógica.

Además de los subsistemas que están formados por una unidad de control electrónica (ECU - *Electronic Control Unit*), el dron dispone de un panel de control que permite realizar las funciones básicas (como el apagado o encendido del dron, o la inserción manual de los parámetros del vuelo).

Como se muestra en el modelo, cada unidad de control electrónica dispone de las tareas que necesita para realizar sus funciones. A continuación se describen brevemente sus funciones:

- PT (*Payload Transport*). Asegura el cumplimiento de la misión.
- EL (*Emergency Landing*). Realiza el aterrizaje de emergencia.
- HM (*Health Monitoring*). Monitoriza el estado del subsistema.

- FT (*Fault Tolerance*). Garantiza el aislamiento y la posible recuperación del subsistema en caso de fallo.
- CM (*Communication Management*). Gestiona las comunicaciones que envía o recibe el subsistema.
- CM_GS (*Communication Management, Ground Station*). Gestiona las comunicaciones con la estación de tierra.
- EM (*Energy Management*). Recaba información relacionada con el consumo de energía.
- FC (*Flight Control*). Controla la ruta de vuelo que realiza el dron.
- MM (*Mission Management*). Gestiona la misión.
- PC (*Propulsion Control*). Controla los motores del dron.
- BC (*Bracking Control*). Controla los frenos hidráulicos del dron.

Que estas tareas estén en diferentes subsistemas no significa que realicen las mismas acciones. Una tarea puede ejecutar una o varias funciones dependiendo del subsistema en el que se encuentre y de si comparten las mismas restricciones temporales. A continuación, se muestra una tabla con todas las tareas del sistema y su periodo de activación:

Subsistema o dispositivo	Tareas	Periodo (ms)
EPS	EM, PC, FC, EL, CM y HM	50
EPS	FT	Aperiódica
HBS	EM, BC, EL, CM y HM	50
HBS	FT	Aperiódica
MMS	PT, EM, MM, FC, CM y HM	50
MMS	FT y CM_GC	Aperiódica
Panel de control	CM	100
Sensores	CM	20
Comunicación con la estación de tierra	CM	1000

Tabla 1: Requisitos temporales de las tareas del *challenge*

2.2 Flujos e2e

En este apartado se comentan los diferentes flujos e2e que se describen en la documentación del *challenge* WATERS 2018. Como se ha comentado anteriormente, mientras se genera el modelo de este sistema se llega a la conclusión, por simple inspección, de que el sistema que se propone no es planificable con los datos que aporta la documentación. Por ello, se decide generar otros dos modelos basándose en este sistema con nuevos datos inventados, pero que puedan tener cierto sentido.

Por esta razón en este apartado solo se van a describir los flujos e2e que se utilizan en estos dos casos y son relevantes para este trabajo. El resto de flujos se describen en el anexo I de este documento.

2.2.1 Flujo principal

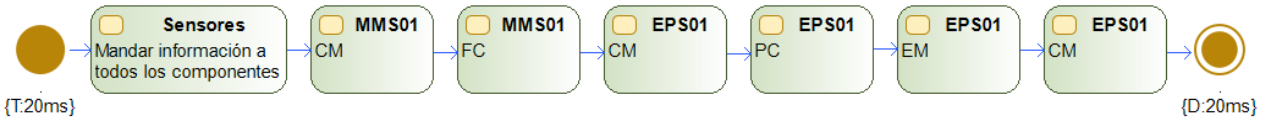

Ilustración 4: Flujo principal del *challenge*

Cada 20ms los sensores transmiten la información a todos los subsistemas. El subsistema maestro (por defecto MMS01) se encarga de procesar la información recibida (CM) y de generar los siguientes comandos de vuelo (FC).

A continuación, se envían los comandos a EPS01. Teniendo en cuenta que para transmitir los comandos es necesario enviarlos a través del bus de Ethernet, se entiende que las tareas CM de MMS01 y EPS01 deben interactuar entre ellas. Por lo tanto, se considera que la información proporcionada es errónea y que el diagrama, en realidad, es como se muestra a continuación.

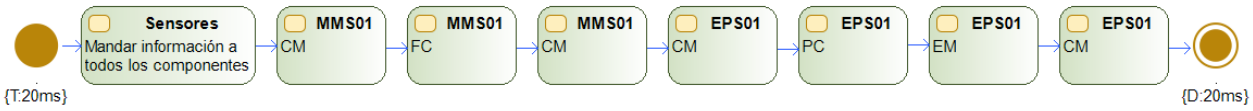

Ilustración 5: Corrección del flujo principal del *challenge*

Cuando EPS01 recibe los comandos de vuelo, los transforma en comandos de voltaje (PC), analiza la situación actual de la energía del dron (EM) y transmite los comandos de voltaje a los motores (CM).

2.2.2 La estación de tierra actualiza los parámetros de vuelo

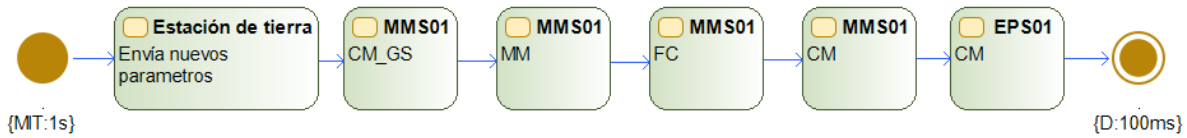

Ilustración 6: Flujo de cambio de coordenadas del *challenge*

Este es un flujo aperiódico, y se determinó que el tiempo mínimo de activación es de un segundo porque es el periodo de la tarea que se encarga de enviar la información al dron.

En cuanto el dron recibe la información que envía la estación de tierra, MMS01 se encarga de procesar la información recibida (CM_GS). A continuación, comprueba que con los recursos actuales el dron puede realizar las acciones que solicita la estación de tierra (MM).

Por último, se generan los comandos de vuelo (FC) para enviárselos a EPS1 (CM de MMS01). Cuando EPS1 reciba los parámetros (CM) los almacenará para su posterior uso.

2.2.3 Gestión de la energía

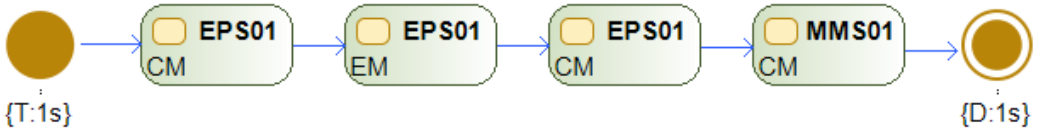

Ilustración 7: Flujo de la gestión de la energía del *challenge*

Durante el vuelo, MMS debe ser consciente de la cantidad de energía disponible para poder tomar la decisión de abortar la misión y aterrizar. Por ese motivo, cualquier subsistema de EPS o HBS se encarga de analizar la situación y mandar la información al subsistema maestro (MMS01). Por defecto, el subsistema encargado de realizar esta acción es EPS01.

Una vez que EPS1 ha analizado la situación actual, envía la información a MMS1 para que la almacene y pueda utilizarla la próxima vez que tenga que evaluar la situación de la misión.

2.3 Interpretación del sistema

En este apartado se interpretará el sistema que se propone en el *challenge*. Para ello, se tendrán en cuenta diferentes aspectos como por ejemplo, los flujos e2e, los tiempos de ejecución de las tareas o los componentes del dron. Finalmente, se realizará una pequeña conclusión indicando la planificabilidad del sistema.

2.3.1 Componentes del sistema

Como se ha visto en el apartado anterior, el dron está formado por tres subsistemas, los cuales están redundados para evitar la pérdida de funcionalidad del dron a causa de un fallo. Además de estos subsistemas, dispone de otros componentes que trabajan de forma conjunta con ellos, como por ejemplo, los motores, los sensores o el panel de control.

Actualmente, estos subsistemas trabajan en modo maestro-esclavo, de forma que, el subsistema MMS es el maestro y el resto son esclavos. Pero, hay que tener en cuenta que todos los subsistemas están redundados y esto, implica que entre los subsistemas que son del mismo tipo, se utilice también el patrón maestro-esclavo.

Poniendo como ejemplo los subsistemas de EPS, EPS01 sería el maestro y EPS02 sería el esclavo. Esto, implica que la actividad de EPS02 está prácticamente suspendida, ya que, la única tarea que realiza es monitorizar a EPS01 para asegurarse de que no falla y de que no tiene que tomar el control.

El hecho de que MMS02, EPS02 y HBS02 son esclavos, implica que prácticamente el 50% del sistema está inactivo a la espera de que los otros subsistemas fallen. Sí que es cierto que la arquitectura del sistema cumple con su objetivo (evitar la pérdida de funcionalidad del dron a causa de un fallo), pero se podría ir un paso más allá para aprovechar los recursos del sistema.

Por ejemplo, se podría balancear la carga de trabajo entre los diferentes subsistemas que cumplan las mismas funciones. Otra opción, consiste en aprovechar estos subsistemas utilizando una arquitectura de redundancia modular dual [15] [16].

2.3.2 Flujos e2e

En el anterior apartado (2.2) y en el anexo I se han descrito los diferentes flujos e2e del *challenge*. Algunos de los flujos no estaban bien definidos y se han corregido. Además, mientras se analizaban se llegó a la conclusión de que faltan varios flujos.

Algunos de ellos, se describen a continuación:

- El dron dispone de un panel de control que permite introducir los parámetros de vuelo y evaluar el estado del dron. Pero, no se ha descrito ningún flujo que se encargue de sincronizar la información.
- Todos los subsistemas disponen de un flujo que se encarga de auto-testear su funcionamiento, y de otro que se encarga de reconfigurar su funcionamiento en caso de recibir algún evento de error. Pero, no existe un flujo que se encargue de monitorizar a otro subsistema para comprobar que no falla

- Cuando MMS falla, se activa el flujo que trabaja en modo degradado, el cual, permite que EPS se encargue de gestionar el estado de la misión. De la misma forma que este flujo se activa cuando MMS falla, debería existir otro flujo que se active cuando MMS y EPS fallan. La función principal de este flujo es realizar un aterrizaje de emergencia.
- La estación de tierra tiene posibilidad de actualizar los parámetros de vuelo, pero en la situación actual, la estación de tierra no tiene la posibilidad de saber cuál es el estado del dron. Por lo tanto, debe haber un flujo que se encargue de recopilar la información y enviársela a la estación de tierra.
- Este dron se encarga de transportar paquetes de un punto a otro. Antes de realizar el despegue debe comprobar si es capaz de completar la misión (teniendo en cuenta el peso del paquete, la energía disponible, etc.).

2.3.3 Interpretación de los datos temporales del sistema

Para realizar el modelo del sistema es necesario conocer todos los datos de los flujos e2e que se proponen en el *challenge*. Por lo tanto, es necesario conocer cierta información como: el periodo o MIT de las tareas y los flujos, el plazo de los flujos y el tiempo de ejecución de las tareas.

En la descripción de los flujos no se proporciona toda la información que se necesita para completar el modelo. O proporcionan el periodo (o MIT en caso de que el flujo sea esporádico) o proporcionan su plazo. Por esta razón, se decidió que cuando se proporciona el periodo, el plazo tendría el mismo valor. De esta forma, al mismo tiempo no puede haber dos ejecuciones de un mismo flujo.

En el caso de que se proporcione el plazo, hay que determinar si el flujo es periódico o aperiódico dependiendo de la función que realiza. Si se determina que el flujo es periódico, se considera que el periodo del flujo es igual al periodo de la primera tarea que se ejecuta, como por ejemplo, en el flujo principal. Y si se considera que el flujo es aperiódico, el MIT es igual al periodo de la primera tarea de ejecución, entre otros, el flujo de actualización de parámetros de vuelo.

Tras establecer el periodo (o el MIT) y el plazo de los flujos, se procedió a determinar el tiempo de ejecución de las tareas. Estos tiempos no se proporcionan en la documentación inicial. Y en teoría, se proporciona el código fuente de la tarea.

El caso es que no se proporciona el código, lo que se proporciona es un modelo SCADE. De esta forma, se podrá generar el código con una herramienta específica para el entorno que se desee. Y el problema es que no disponemos de esta herramienta y por ello no tenemos forma de obtener los tiempos de ejecución de las tareas.

Antes de plantear una solución sobre cómo establecer los tiempos de ejecución de las tareas, se procedió a interpretar el funcionamiento temporal de los flujos teniendo en cuenta todos los datos recabados hasta el momento (periodos, MIT y plazos). Se llegó a la conclusión de que el sistema que se propone no es planificable.

Para demostrarlo, utilizaremos como ejemplo el flujo que trabaja en modo degradado cuando MMS falla (descrito en el anexo I). Se ha propuesto como ejemplo este flujo porque no ha sido modificado y tiene un periodo y un plazo de 20 milisegundos.

En la ilustración 8 se ha supuesto que los tiempos de ejecución de todas las tareas son de 4 milisegundos, y que el momento de activación coincide con el momento en que acaba la tarea antecesora.

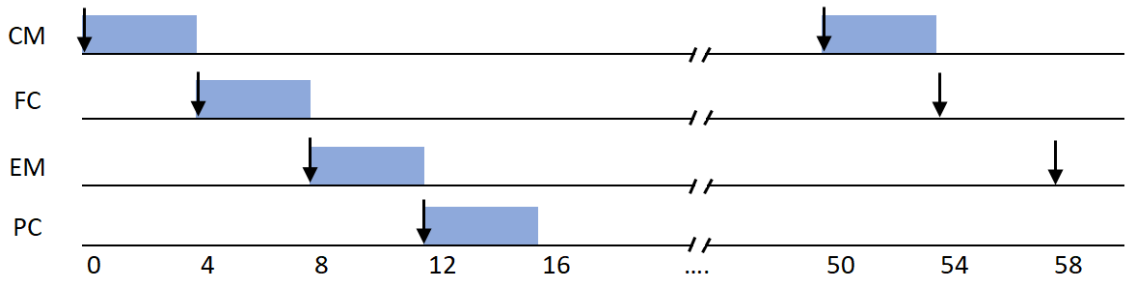

Ilustración 8: Ejemplo de ejecución de un flujo e2e

Todas las tareas tienen un periodo de 50 milisegundos y en este flujo la tarea CM se ejecuta 2 veces. El flujo comienza en el momento 0 (cuando se activa por primera vez CM) y como se puede observar, la primera ejecución de CM y las ejecuciones de FC, EM y PC entran dentro de los primeros 20 milisegundos.

Para finalizar la ejecución de este flujo, la tarea CM debe ejecutarse una segunda vez. Y como la tarea tiene un periodo de 50 milisegundos, no podrá volverse a activar hasta que pasen 50 milisegundos desde su primera activación. Esto significa que la segunda activación de CM no entrará dentro de los primeros 20 milisegundos y por lo tanto, el sistema que se propone en el *challenge* no sería planificable.

2.3.4 Conclusiones de la interpretación del sistema

Tras estudiar el sistema propuesto en el *challenge* se llegó a la conclusión de que falta mucha información en la propuesta para tratar de construir un modelo viable.

Por un lado, tenemos una arquitectura con varios subsistemas redundados que utilizan el patrón maestro-esclavo con el objetivo de que el dron no pierda funcionalidad a causa de un fallo. Por otro lado, el sistema dispone de siete flujos e2e (de los cuales, dos de ellos no están bien definidos) que solo cumplen con una parte de la funcionalidad que se pide.

Diseñar e implementar los cambios que se comentan en los apartados anteriores puede formar parte del *challenge* que proponen, pero la cuestión más importante es que, teniendo en cuenta los requisitos temporales que se describen en la documentación, el sistema no sería planificable.

Actualizar los requisitos temporales además de los cambios que se comentaron anteriormente implicaría diseñar un sistema totalmente diferente al que se propone.

Por lo tanto, se decidió implementar dos casos para el análisis basándonos en el modelo que se propone en el *challenge*.

El primero de ellos se denomina caso redundado lineal. Este caso describe un sistema cuyos componentes son totalmente independientes. Esto da lugar a que el sistema esté formado por dos subsistemas idénticos, que realizan las mismas funciones, y que obtienen los mismos tiempos de respuesta.

El segundo caso se llama caso redundado 1oo2. Esta solución, se basa en el caso redundado lineal. La diferencia está en que se utilizará el esquema 1oo2 [16] [17] para que ambos subsistemas se comuniquen entre sí. De esta forma, el sistema tendrá mayor fiabilidad, usabilidad y seguridad.

3 Modelado y análisis del caso redundado lineal

En esta apartado se describirá el caso redundado lineal, el cual es una versión simplificada del sistema que se propone en el *challenge*. Al igual que el sistema original, los componentes del sistema están redundados, pero a diferencia de este, los componentes son totalmente independientes.

Es decir, este caso está formado por dos sistemas independientes que no se comunican entre ellos. Por un lado tenemos un sistema formado por MMS01 y EPS01 y por el otro, un sistema formado por MMS02 y EPS02.

Como consecuencia, los resultados obtenidos en ambos sistemas son totalmente idénticos. Por lo tanto, se ha decidido describir solo uno de estos sistemas para simplificar la descripción del modelado y el análisis.

3.1 Arquitectura del sistema

Para diseñar la arquitectura de este sistema se llevó a cabo una investigación para averiguar qué enfoque se estaba utilizando actualmente. Con los resultados obtenidos [18] [19] [20] se pudo observar que todos los sistemas disponían de un módulo encargado de gestionar la misión. Así pues, se decidió mantener la arquitectura que se propone en el *challenge*.

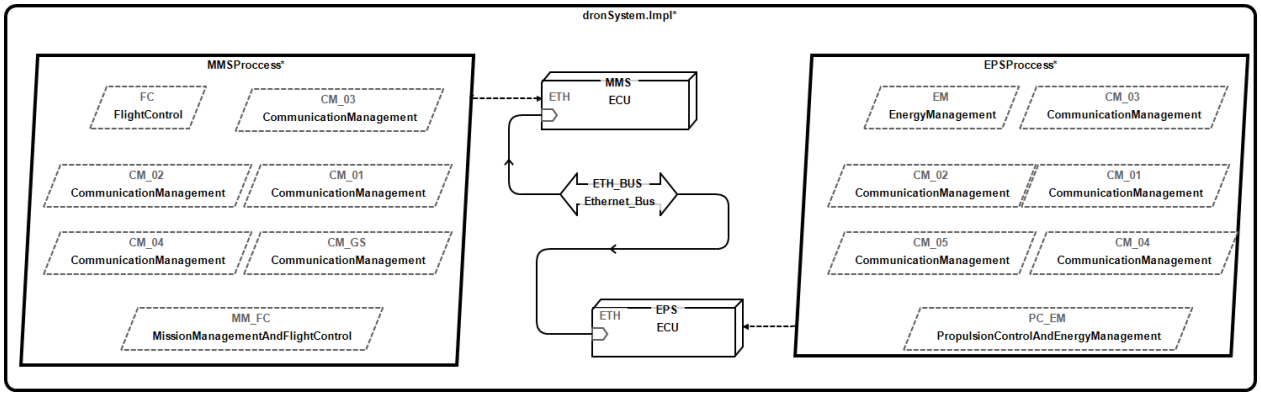

Ilustración 9: Arquitectura (simplificada) del caso redundado lineal

Como se puede observar en la ilustración, el sistema está formado por dos subsistemas:

- MMS (*Mission Management System*). Este componente se encarga de gestionar la misión del dron y de comunicarse con la estación de tierra para recibir nuevos parámetros de vuelo.

- EPS (*Electrical Propulsion System*). Se encarga de transformar las instrucciones que le manda MMS en comandos de voltaje para enviárselos a los motores eléctricos. Además, se encarga de recoger datos relacionados con el gasto de energía y enviárselos a MMS.

Cada subsistema dispone de un conjunto de tareas que les permite realizar sus funciones. En principio estas tareas iban a ser las mismas que en el modelo anterior pero se ha decidido realizar algunos cambios.

Por un lado, aquellas tareas que se realizan en un mismo flujo e2e y en un mismo subsistema se han combinado en una sola tarea. De esta forma, en el subsistema MMS tenemos la tarea MM_FC (*Mission Management and Flight Control*) y en el subsistema EPS la tarea EM_PC (*Energy Management and Propulsion Control*).

Por otro lado, se han dividido las diferentes funciones que realiza la tarea CM dando lugar a varias tareas (CM_01, CM_02,…). De esta manera, cada tarea CM es asignada a un flujo e2e y con una sola función.

3.2 Flujos e2e

En este apartado, se comentan los flujos e2e del sistema. Como se ha comentado anteriormente, este caso es una versión simplificada del *challenge*.

3.2.1 Flujo principal

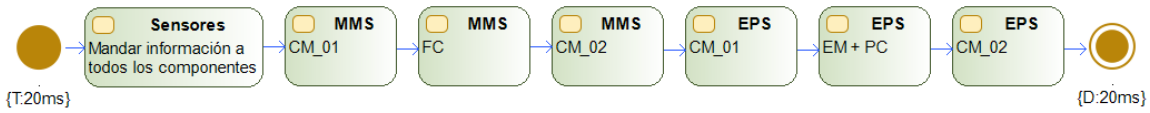

Ilustración 10: Flujo principal del modelo lineal

Este flujo es el encargado de controlar el vuelo del dron. Los sensores envían la información a los subsistemas cada 20 milisegundos. A continuación MMS recoge y procesa esos datos para generar los siguientes comandos de vuelo y enviárselos a EPS. Por último, EPS transforma esos comandos de vuelo en comandos de voltaje para enviárselos a los motores.

Si se compara este flujo con el del modelo anterior, su función es la misma, pero lo que cambia, son las tareas que ejecutan esa función. La mayor diferencia es que ahora cada subsistema dispone de dos tareas CM: una de ellas se encarga de recibir los datos y la otra de enviarlos.

3.2.2 La estación de tierra envía nuevos parámetros de vuelo

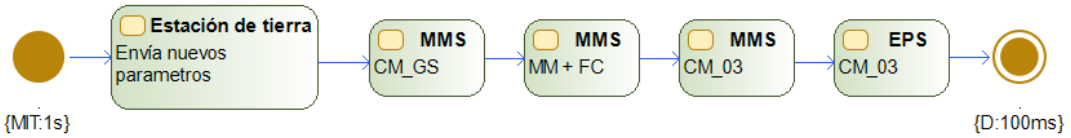

Ilustración 11: Flujo de cambio de coordenadas del modelo lineal

Si se da el caso de que la estación de tierra actualiza los parámetros de vuelo, MMS se encarga de procesar los datos que recibe de la estación de tierra. A continuación determina si la misión puede seguir adelante teniendo en cuenta los nuevos parámetros de vuelo. Si es así, genera los comandos de vuelo y se los envía a EPS. En el caso de que se determine que la misión no es viable con los nuevos parámetros, se ignoran.

3.2.3 Gestión de la energía

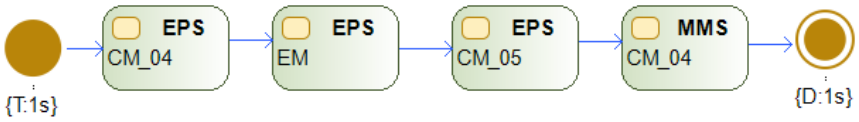

Ilustración 12: Flujo de gestión de la energía del modelo lineal

Para que MMS pueda decidir cómo continuar con la misión, es necesario que conozca la situación de los recursos del dron. Por esta razón, EPS se encarga de analizar el gasto de energía y enviar los resultados a MMS.

Si se compara el flujo con el del *challenge*, se puede observar que el flujo es el mismo. Y teniendo en cuenta que quien necesita la información para decidir continuar con la misión es MMS, se podría llegar a plantear la actualización del flujo para que la ejecución de la tarea EM la realice MMS.

Al dejar el flujo tal como está se consiguen varias ventajas, como por ejemplo, balancear la carga de trabajo (en los anteriores flujos MMS tiene más carga de trabajo que EPS).

3.3 Tiempos de ejecución de las tareas

Como se ha comentado anteriormente, en el *challenge* no se especificaban los tiempos de ejecución de las tareas (se proporcionaba un modelo SCADE que te permitía generar el código de la tarea). Por lo que se decidió asignar unos tiempos de ejecución ficticios a las tareas. Para ello, se tuvieron en cuenta los siguientes puntos:

- La utilización del sistema no debe superar el 25% - 30%.
- Las funciones que realizaban las tareas.
- El plazo de los flujos e2e.

Considerando estos puntos, se establecieron los tiempos de ejecución en el peor caso (WCET) tal como se muestra en la tabla 2.

Subsistema	Flujo e2e	Tarea	WCET (ms)	Prioridad
MMS	**Flujo principal**	CM_01	0.2	30
		FC	2	26
		CM_02	0.8	22
	Actualización de coordenadas	CM_GS	0.4	18
		MM_FC	4	14
		CM_03	0.6	10
	Gestión de energía	CM_04	10	6
EPS	**Flujo principal**	CM_01	0.4	30
		PC_EM	2	26
		CM_02	0.6	22
	Actualización de coordenadas	CM_03	2	18
	Gestión de energía	CM_04	2	14
		EM	40	10
		CM_05	8	6

Tabla 2: Tiempos de ejecución y prioridades de las tareas del caso redundado lineal

Además de determinar el WCET de las tareas, también se han especificado las prioridades de las tareas. Para ello, se decidió asignar prioridades dependiendo de los plazos de los flujos e2e, y dentro de un flujo prioridades decrecientes a medida que avanza el flujo.

Por ejemplo, las tareas del flujo principal tienen más prioridad que las de los otros dos flujos (debido a que su plazo es más pequeño). Y dentro del flujo principal, la tarea CM_01 de ambos subsistemas es la que tiene mayor prioridad porque es la primera que se ejecuta.

Por último, queda pendiente el periodo de activación de las tareas. En el modelo anterior, la mayoría de las tareas eran periódicas y se activaban cada 50 milisegundos. En este caso, las tareas se activarán mediante eventos al ritmo de la activación del flujo e2e al que pertenecen.

3.4 Análisis temporal

En este apartado se desarrolla el análisis temporal teniendo en cuenta el caso descrito en los apartados anteriores. Para desarrollar el análisis se utilizarán las cuatro técnicas de análisis descritas en la introducción: holística, basada en *offsets*, basada en *offsets* por fuerza bruta y basada en *offsets slanted*.

Además de analizar la situación con la carga original, queremos saber cómo se comporta el sistema al incrementar la carga de trabajo. Para hacer esto, tenemos dos opciones: incrementar el WCET de las tareas o reducir el *speed factor* de los procesadores (ya que, este factor es el inverso proporcional del WCET). En este caso, reduciremos el *speed factor* de los procesadores hasta que la utilización del sistema esté cerca del 100%.

Por último, además de proporcionar los datos básicos de un análisis temporal (los peores tiempos de respuesta de los flujos e2e y la utilización de los procesadores), también se indica el *slack* del sistema.

3.4.1 Resultados obtenidos

Speed factor	Técnica utilizada	*Slack* del sistema	Utilización MMS	Utilización EPS	Flujo Principal	Cambio Coordenadas	Gestión energía
1	Algoritmos basados en *offsets*	233.20%	26%	24%	6 ms	19 ms	91 ms
	Holística	78.52%			11.2 ms	23.8 ms	146 ms
0.8	Algoritmos basados en *offsets*	166.41%	32.5%	30%	7.5 ms	23.75 ms	121.25 ms
	Holística	42.58%			14 ms	29.75 ms	196.25 ms
0.6	Algoritmos basados en *offsets*	100.00%	43.33%	40%	10 ms	31.67 ms	180 ms
	Holística	7.03%			18.67 ms	39.67 ms	288.33 ms
0.4	Algoritmos basados en *offsets*	33.20%	65%	60%	15 ms	47.5 ms	350 ms
	Holística	-28.91%			28 ms	67 ms	596.5 ms
0.3	Algoritmos basados en *offsets*	0.00%	86.67%	80%	20 ms	98 ms	808.67 ms
	Holística	-46.48%			37.33 ms	109.33 ms	1238.67 ms
0.27	Basada en *offsets* y basada en *offsets slanted*	-10.16%	96.30%	88.89%	22.22 ms	147.41 ms	1483.70 ms
	Basada en *offsets* por fuerza bruta	-10.16%			22.22 ms	147.41 ms	1442.96 ms
	Holística	-51.95%			41.48 ms	161.85 ms	2102.22 ms

Tabla 3: Resultados del análisis temporal del caso redundado lineal

En general, los resultados obtenidos entran dentro de lo esperado. En la situación normal (*speed factor* = 1) la utilización de los procesadores coincide con lo previsto. Ya que, a la hora de proponer los WCET de la tareas se intentó que la utilización del sistema estuviese entre el 25%-30%.

También se puede apreciar que la técnica holística es más pesimista que cualquiera de las técnicas de análisis de *offsets*. No solo por el hecho de que el *slack* del sistema es mucho menor, sino porque el tiempo de respuesta de dos de los flujos es casi el doble (el flujo principal y el de la gestión de energía).

Además, se puede apreciar que las técnicas de análisis basadas en *offsets* ofrecen los mismos resultados, excepto cuando el *speed factor* es igual a 0.27. En este caso la técnica basada en *offsets* por fuerza bruta ofrece un resultado diferente respecto a las otras dos técnicas. Es normal ya que esta técnica realiza un análisis exacto y puede ofrecer mejores resultados que los otros dos algoritmos.

También se puede apreciar que en las técnicas de *offsets*, el flujo principal obtiene el mejor tiempo de respuesta, el cual es la suma de todos los WCET de las tareas del flujo. Es normal, teniendo en cuenta que las tareas de este flujo son las que tienen más prioridad y no hay ninguna tarea que las pueda expulsar.

3.4.2 Representación de los resultados

Para observar la evolución del sistema a medida que desciende el *speed factor,* se presentan varios gráficos.

3.4.2.1 Slack del sistema

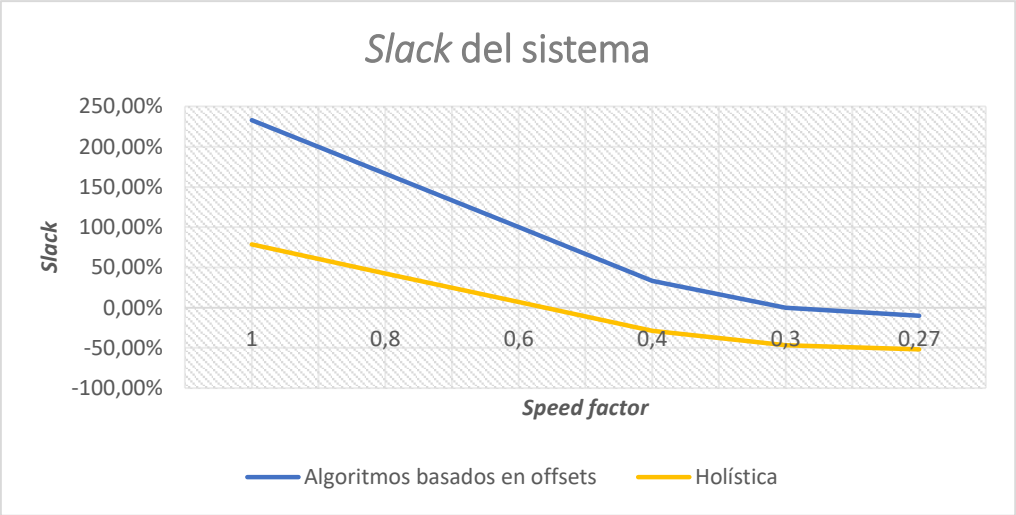

Ilustración 13: Evolución del *slack* del sistema

Como se puede observar, la técnica holística siempre es más pesimista que cualquiera de las tres técnicas basadas en *offsets*. Inicialmente, todas las técnicas indican que el sistema es planificable. Aunque la técnica holística indica que el sistema tiene un *slack* del 42.5% y las técnicas de *offsets* del 166.41%.

A medida que se reduce el *speed factor*, el *slack* va descendiendo. Cuando está por debajo de 0.6, el algoritmo holístico indica que el sistema ya no es planificable. Sin embargo, las técnicas basadas en *offsets* indican que el sistema tiene un *slack* del 100%. Hasta que el *speed factor* no está por debajo de 0.3 no indican que el sistema no es planificable.

3.4.2.2 Tiempos de respuesta

Otra forma de observar la evolución del sistema es analizar el tiempo de respuesta en los diferentes flujos. En la ilustración 14, se puede observar cómo se incrementa el tiempo de respuesta en el flujo principal a medida que desciende el *speed factor.*

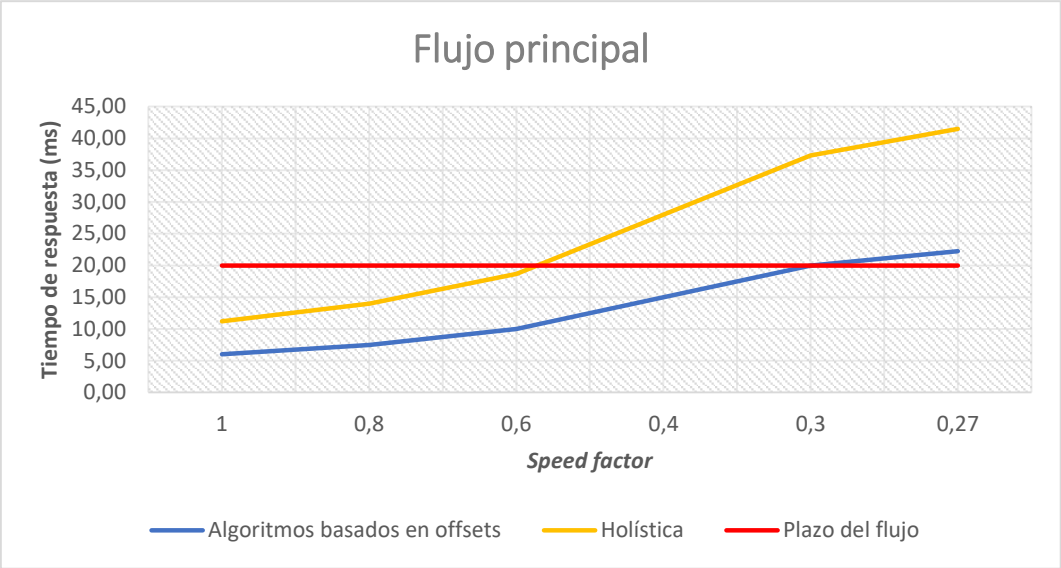

Ilustración 14: Evolución del tiempo de ejecución en el flujo principal

Al igual que en el gráfico del *slack* del sistema (ilustración 13), se puede apreciar que la técnica holística es más pesimista que las que se basan en *offsets*. También se puede apreciar que el tiempo de respuesta que ofrece la técnica holística en este flujo, es el doble que el que ofrece cualquiera de las técnicas basadas en *offsets*.

Además, si se examinan los dos gráficos a la vez, se puede llegar a la conclusión de que este flujo es el que controla la planificabilidad del sistema. Ya que, cuando el sistema deja de tener holgura, el tiempo de respuesta de este flujo supera su plazo.

Los siguientes gráficos representan el tiempo de respuesta de los otros dos flujos (el cambio de coordenadas y la gestión de energía). Se obtienen conclusiones similares a los anteriores casos, aunque en estos casos el tiempo de respuesta que indica la técnica holística no es el doble que las técnicas basadas en *offsets.*

Además, se puede apreciar que en cuanto se reduce el *speed factor* más de 0.6 el tiempo de respuesta se incrementa de forma considerable.

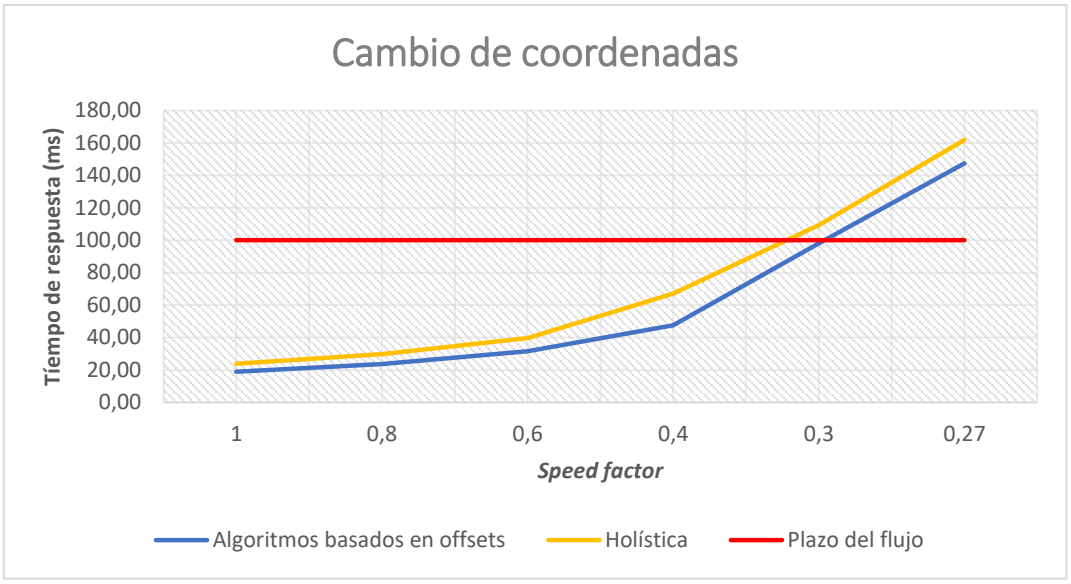

Ilustración 16: Evolución del tiempo de ejecución en el flujo cambio de coordenadas

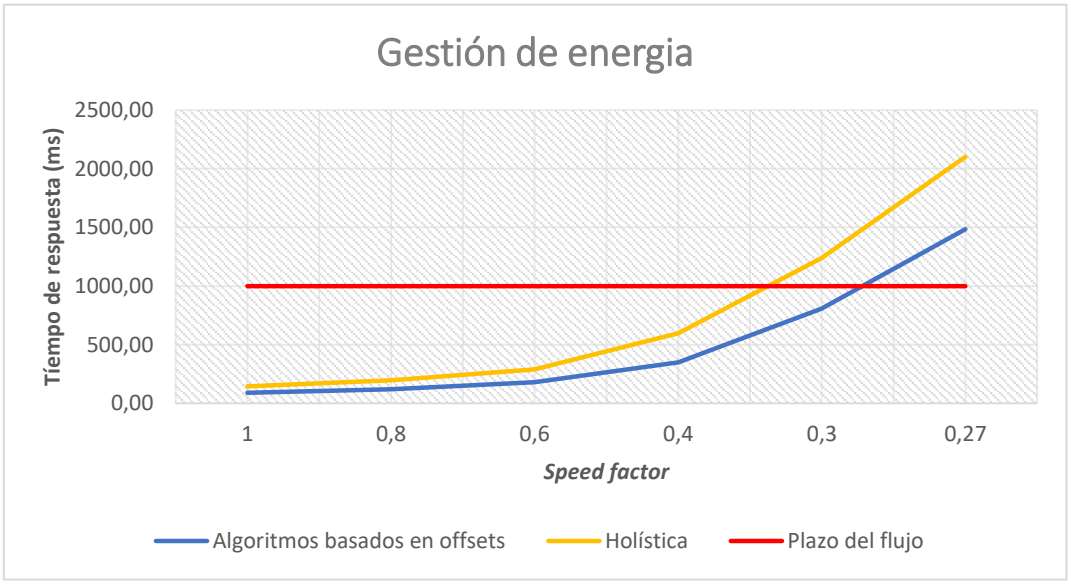

Ilustración 15: Evolución del tiempo de ejecución en el flujo gestión de energía

4 Modelado y análisis del caso redundado 1oo2

En este apartado se va a describir un caso que se basa en la solución anterior. La diferencia entre ambos casos es que los componentes del mismo tipo se comunican entre ellos para contrastar los resultados que generan.

4.1 Arquitectura del sistema

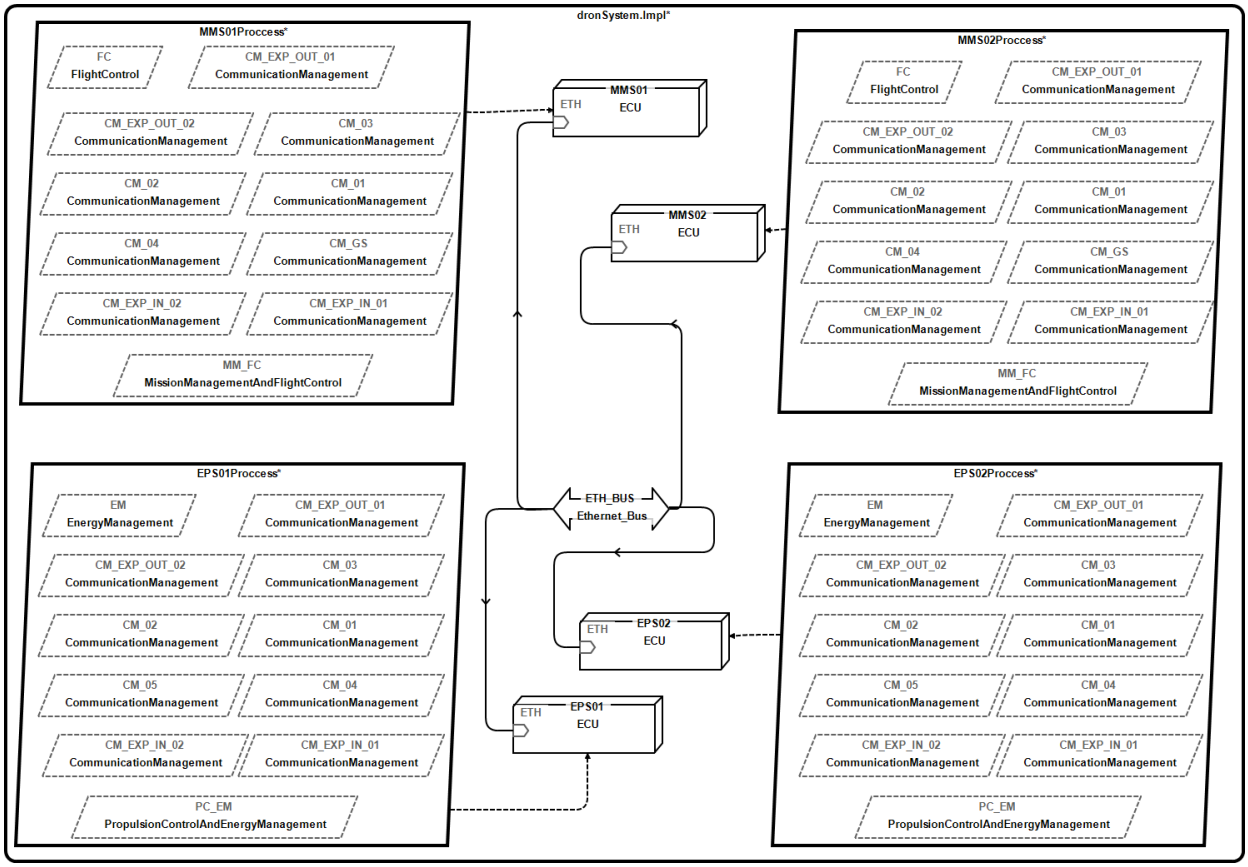

Ilustración 17: Arquitectura del caso redundado 1oo2

El objetivo de este caso no solo es cumplir con el que se comenta en el *challenge* (que un solo fallo en el sistema provoque la pérdida de funcionalidad del dron), sino que también se pretende aumentar el nivel de integridad de seguridad, en inglés *Safety Integrity Level* (SIL) [16] [17].

El SIL es el nivel relativo de reducción de riesgo que provee una función de seguridad o el nivel objetivo para la reducción de riesgo [21]. Hay cinco niveles de seguridad, siendo SIL0 el más bajo (no hay seguridad) y SIL4 el más alto.

El dron debe asegurar alta disponibilidad, fiabilidad, robustez y eficiencia energética. Por ello, se propone utilizar recursos cuyo nivel de seguridad es SIL0 y hacer que trabajen de forma conjunta utilizando el esquema 1oo2 [16] [17] para incrementar el nivel SIL del sistema.

El propósito de utilizar el esquema 1oo2 es que los subsistemas del mismo tipo se comuniquen entre ellos para comparar los resultados que generan. En principio los subsistemas que son del mismo tipo tendrán la misma implementación y recibirán los mismos parámetros. Por lo tanto, retornarán el mismo resultado. En el caso de que no sea así, significará que uno de ellos no funciona correctamente y será necesario utilizar algún tipo de votación para determinar qué resultado utilizar.

Para establecer la comunicación entre los subsistemas del mismo tipo, se han implementado dos nuevos tipos de tareas: CM_EXP_OUT (envía el resultado al otro subsistema) y CM_EXP_IN (recibe el resultado del otro subsistema).

En total, cada subsistema tiene 11 tareas. Las tareas que no se han mencionado tienen las mismas funciones que se describen en el apartado 3.1 del caso redundado lineal.

4.2 Flujos end to end

Para implementar el esquema 1oo2 y conseguir que dos subsistemas del mismo tipo se comuniquen, es necesario utilizar dos tipos de manejadores de eventos (*event handlers*) que permiten que un modelo tenga flujos *multipath* [22]:

- *Fork*: Provoca que una tarea o evento active varias tareas de forma simultánea.
- *Join*: Una tarea solo se ejecuta cuando recibe todos los eventos de llegada.

A continuación se muestra un ejemplo de un flujo *multipath* en el que se utiliza un *fork* y un *join*:

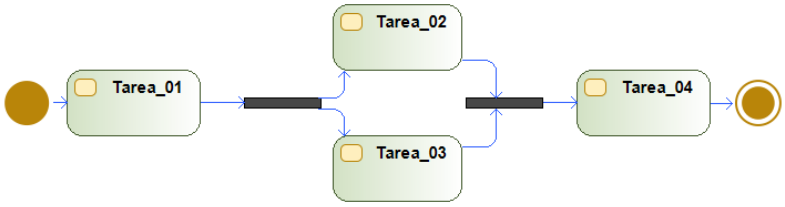

Ilustración 18: Ejemplo flujo *multipath*

Como se puede observar en la ilustración 18, cuando 'Tarea_01' termina, se utiliza un *fork* para activar las tareas 'Tarea_02' y 'Tarea_03'. Y 'Tarea_04' solo se activa cuando 'Tarea_02' y 'Tarea_03' terminan dado que, se está utilizando un *join*.

Como se ha comentado anteriormente, los flujos de este caso se basan en los flujos del caso redundado lineal (3.2). La diferencia que hay es que estos flujos son *multipath* y utilizan *fork* y *join* para establecer una comunicación entre los subsistemas del mismo tipo. A continuación se muestran los modelos de estos flujos *multipath* para nuestro sistema.

4.2.1 Flujo principal

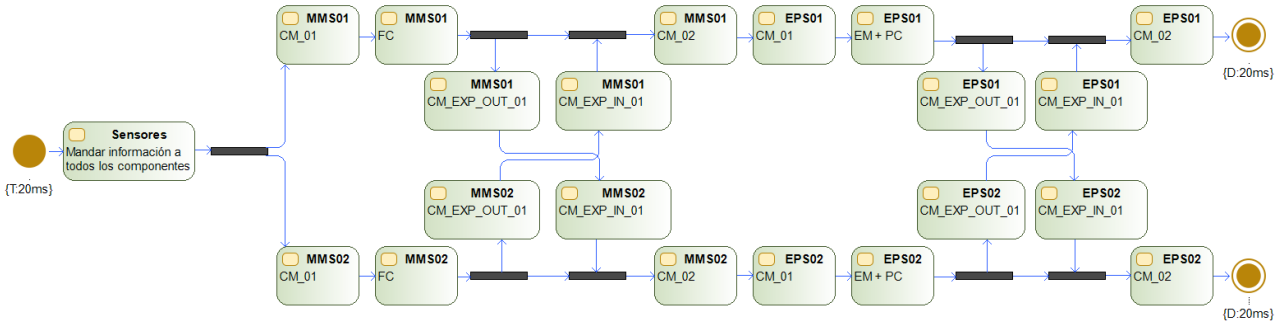

Ilustración 19: Flujo principal utilizando *fork-join*

4.2.2 Cambio de coordenadas

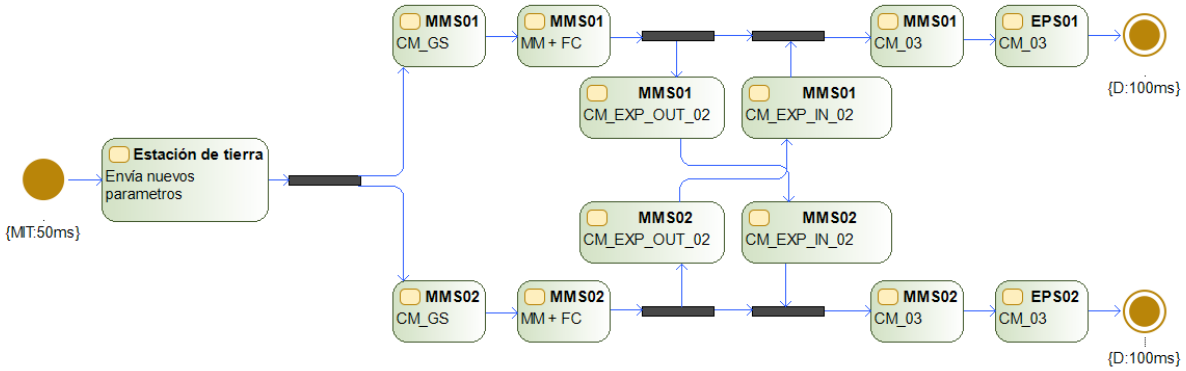

Ilustración 20: Flujo de cambio de coordenadas utilizando *fork-join*

4.2.3 Gestión de energía

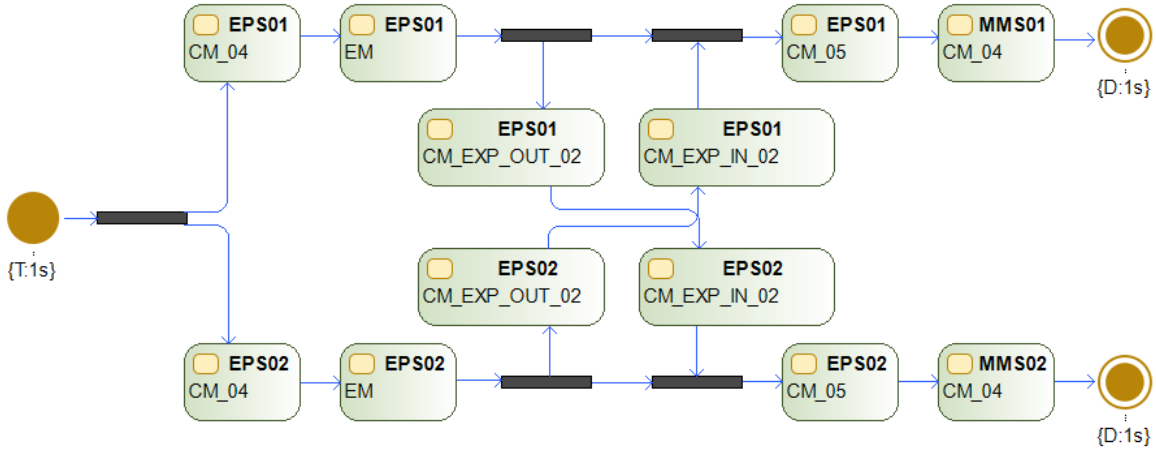

Ilustración 21: Flujo de la gestión de energía utilizando *fork-join*

4.3 Tiempos de ejecución de las tareas

Los WCET y las prioridades de las tareas que ya existen en el caso redundado lineal son las mismas que aparecen en el apartado 3.3. A continuación, se detalla la información de las tareas nuevas que se han implementado, en este caso, correspondiente a las comunicaciones.

Subsistema	Flujo e2e	Tarea	WCET (ms)	Prioridad
MMS	**Flujo principal**	CM_EXP_OUT_01	0.01	25
		CM_EXP_IN_01		24
	Actualización de coordenadas	CM_EXP_OUT_02		13
		CM_EXP_IN_02		12
EPS	**Flujo principal**	CM_EXP_OUT_01		25
		CM_EXP_IN_01		24
	Gestión de energía	CM_EXP_OUT_02		8
		CM_EXP_IN_02		7

Ilustración 22: Tiempos de ejecución y prioridades de las nuevas tareas

4.4 Análisis temporal

En este apartado se describe el análisis temporal del caso utilizando flujos *multipath*. Para realizar el análisis utilizaremos la herramienta MAST, la cual te permite desarrollar modelos con flujos *multipath*.

La última versión pública de MAST (1.5.1.0) sólo permite aplicar la técnica holística de análisis a modelos *multipath*, por lo que se ha utilizado una versión de desarrollo (*beta*) que no es pública pero implementa las nuevas técnicas basadas en *offsets* para flujos *multipath* [16]. Este trabajo sirve así como test de la implementación de estas técnicas en MAST.

De hecho, durante el análisis de este sistema se detectó que algunos resultados de tiempos de respuesta no parecían correctos y fue necesario revisar el código de MAST para corregir el problema que efectivamente existía.

4.4.1 Resultados obtenidos

Speed factor	Técnica utilizada	*Slack* del sistema	Utilización MMS	Utilización EPS	Flujo Principal	Cambio Coordenadas	Gestión energía
1	Algoritmos basados en *offsets*	184.77%	26.14%	24.10%	6.04 ms	25.14 ms	101.2 ms
	Holística	-2.73%			20.5 ms	38.77 ms	266.24 ms
0.8	Algoritmos basados en *offsets*	127.73%	32.67%	30.13%	7.55 ms	31.43 ms	134.05 ms
	Holística	-22.27%			25.63 ms	49.21 ms	362.94 ms
0.6	Algoritmos basados en *offsets*	70.70%	43.57%	40.17%	10.07 ms	41.9 ms	201.52 ms
	Holística	-41.80%			34.18 ms	74.98 ms	551.25 ms
0.4	Algoritmos basados en *offsets*	13.67%	65.35%	60.25%	15.1 ms	81.45 ms	424.35 ms
	Holística	-61.33%			70.35 ms	145.83 ms	1173.22 ms
0.355	Algoritmos basados en *offsets*	0.78%	73.63%	67.89%	17.01 ms	91.77 ms	550.06 ms
	Holística	-65.63%			79.27 ms	175.04 ms	1535.94 ms
0.27	Basada en *offsets* y basada en *offsets slanted*	-23.44%	96.81%	89.27%	22.37 ms	198.26 ms	2422.33 ms
	Basada en *offsets* por fuerza bruta	-23.44%			22.37 ms	198.26 ms	2381.33 ms
	Holística	-73.83%			131.07 ms	511.11 ms	6309.41 ms

Tabla 4: Resultados del análisis temporal del caso redundado 1oo2

Como se puede observar, se han utilizado las técnicas de análisis basadas en *offsets* y la técnica holística para obtener los resultados. Además, se ha ido reduciendo el *speed factor* de los procesadores para analizar cómo se comporta el sistema.

Analizando los datos, lo que más llama la atención es que la técnica holística indica desde el primer momento que el sistema no es planificable. Inicialmente se indica que la causa está en el tiempo de respuesta del flujo principal (que supera su plazo), pero cuando el *speed factor* baja de 0.6, el resto de los flujos superan el plazo establecido.

Respecto a las técnicas de análisis basadas en offsets, las tres técnicas dan los mismos resultados excepto cuando el sistema deja de ser planificable (*speed factor* = 0.27). En este caso, la técnica de análisis basada en offsets por fuerza bruta obtiene un mejor resultado para el flujo de la gestión de energía.

4.4.2 Representación de los resultados

Al igual que en el caso anterior, para evaluar la evolución del sistema se han generado los siguientes gráficos.

4.4.2.1 Slack del sistema

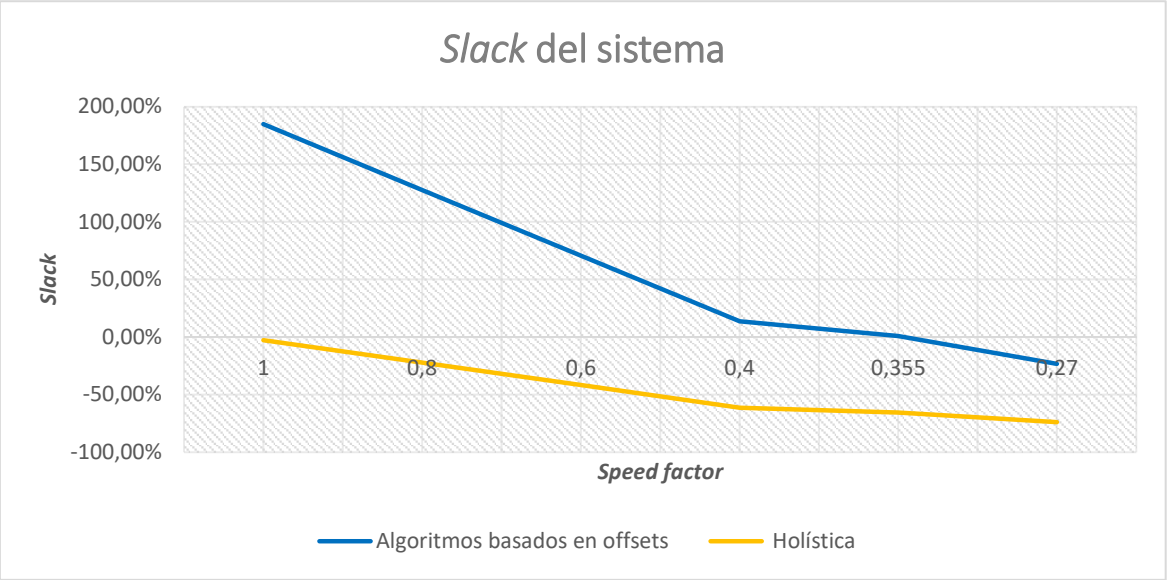

Ilustración 23: Evolución del *slack* del sistema

Inicialmente, las técnicas basadas en *offsets* indican que el sistema tiene un *slack* del 184.77%. A medida que se reduce el *speed factor*, el *slack* se va reduciendo de forma constante hasta que la utilización del sistema supera el 100%. Este hecho ocurre cuando *speed factor* está por debajo de 0.27.

Por otro lado, tal como se ha comentado anteriormente, los resultados obtenidos con el algoritmo holístico indican que el sistema no es planificable.

4.4.2.2 Tiempos de respuesta

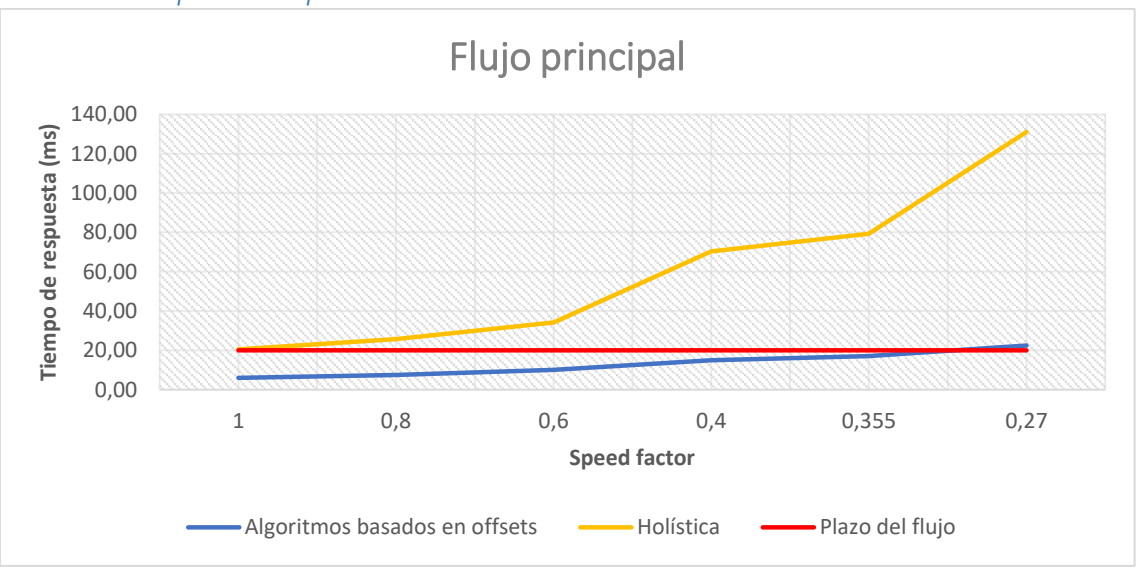

Ilustración 24: Evolución del tiempo de respuesta en el flujo principal

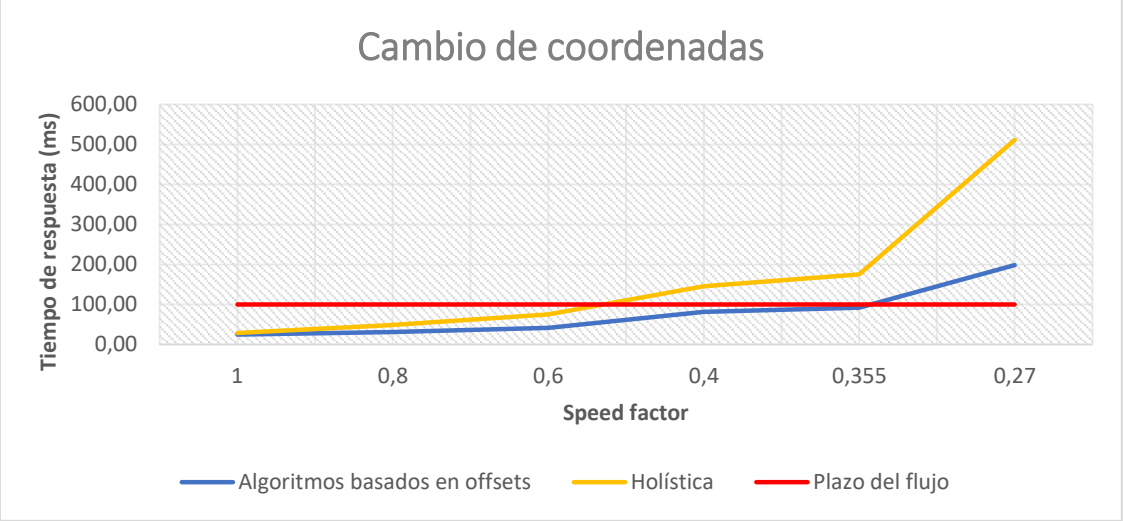

Ilustración 25: Evolución del tiempo de respuesta en el flujo cambio de coordenadas

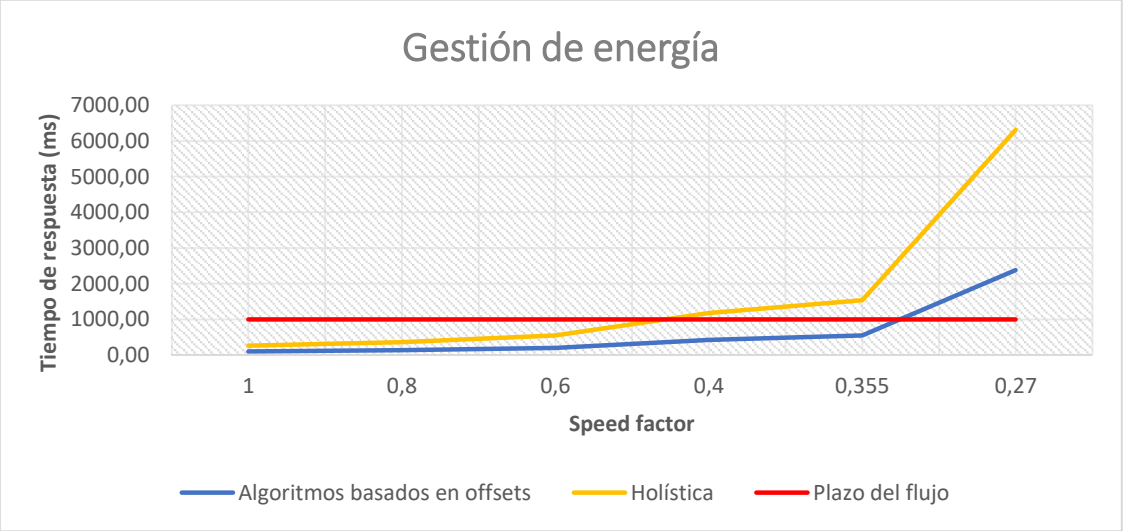

Ilustración 26: Evolución del tiempo de respuesta en el flujo gestión de energía

Al examinar las gráficas que representan los tiempos de respuesta de los flujos e2e se puede apreciar un patrón. Cuando el *speed factor* pasa de ser 0.6 a 0.4, los tiempos de respuesta aumentan el doble (aproximadamente). En el caso del flujo principal, utilizando la técnica holística, el tiempo de respuesta pasa de 34.18 ms a 70.35 ms. Otro ejemplo puede ser utilizando las técnicas basadas en *offsets* en el flujo de cambio de coordenadas (de 41.9 ms a 81.45 ms).

Hay que destacar que este efecto se puede observar mejor en la gráfica que muestra los resultados del flujo principal. Probablemente se deba a que este flujo es más crítico que los otros dos y su plazo es mucho menor.

Por último, se puede determinar que en cuanto el sistema deja de ser planificable, los tiempos de respuesta aumentan de forma considerable.

4.4.3 Comparación de resultados con el caso redundado lineal

Tras realizar el análisis temporal de este sistema y comprobar que el sistema es planificable (al menos según las técnicas de análisis basadas en *offsets*), vamos a proceder a comparar ambos casos, ya que, realizan las mismas funciones. La diferencia entre ambos casos es que este utiliza flujos *multipath* para establecer una comunicación entre los subsistemas del mismo tipo.

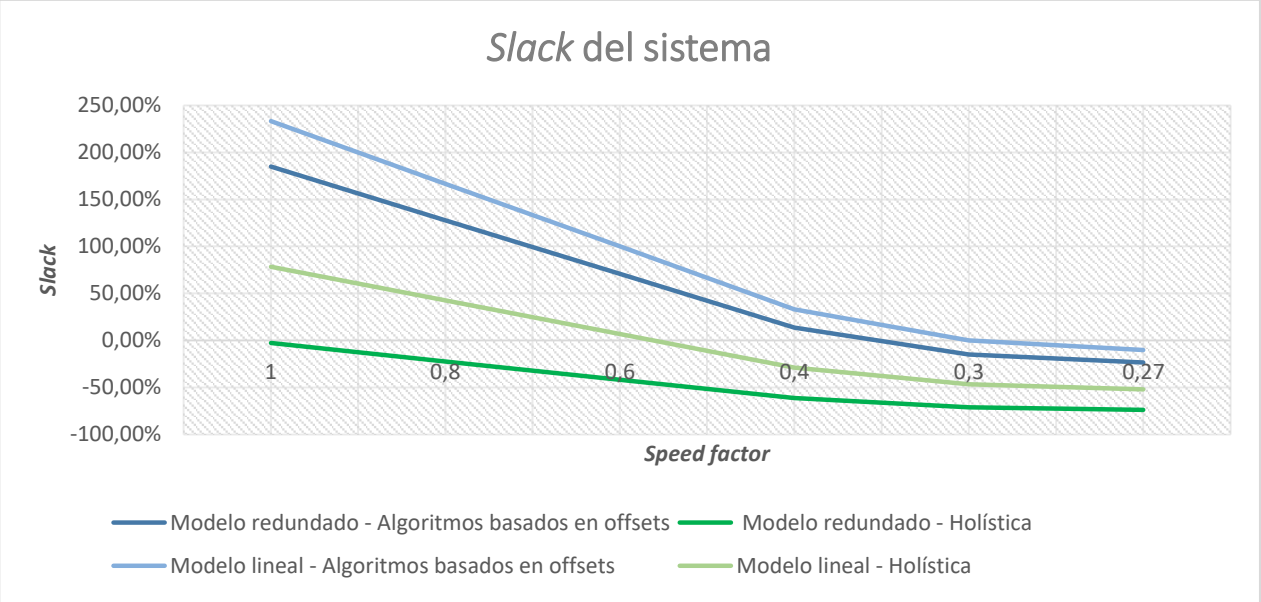

Ilustración 27: Comparación de los *slacks* de ambos casos

Como se puede apreciar, cuando el *speed factor* es igual a 1 hay bastante diferencia entre ambos casos (sobre todo si se tiene en cuenta los resultados de la técnica de análisis holística). Pero a medida que se reduce el *speed factor*, los *slacks* se empiezan aproximar.

El hecho de que el caso redundado 1oo2 tenga menor *slack* que el lineal es normal, teniendo en cuenta que este caso tiene más tareas y utiliza una arquitectura de redundancia modular dual.

Además hay que tener en cuenta que utiliza el esquema 1oo2 para comunicar los subsistemas del mismo tipo. Esto no solo permite cumplir con los objetivos *challenge* (disponibilidad, utilización,…) sino que además aumenta la seguridad del dispositivo, aumentando a su vez, el nivel SIL del sistema.

Teniendo en cuenta todos los datos que se han obtenido hasta el momento. Se seleccionaría el caso que utiliza una arquitectura de redundancia modular, con el esquema 1oo2, para presentar una solución al *challenge* WATERS 2018. Esto es así, no solo por el hecho de que este modelo es el que más se asemeja al sistema propuesto por el *challenge*, sino porque también cumple con todos los objetivos que se proponen.

5 *Benchmarks* para los casos de *fork, join* y *merge* en la herramienta MAST

Las versiones antiguas de MAST permiten modelar sistemas con flujos *multipath*. Una característica de estos flujos es que suelen utilizar alguno de estos manejadores eventos:

- *Fork*: Provoca que una tarea o evento active varias tareas de forma simultánea.
- *Join*: Una tarea solo se ejecuta cuando recibe todos los eventos de llegada.
- *Merge*: Una tarea se ejecuta cada vez que recibe uno de los posibles eventos de llegada.

Aunque en estas versiones puedan modelar flujos *multipath*, solo se podía aplicar la técnica de análisis holística. Como se ha comentado anteriormente, sobre la última versión de MAST (1.5.1.0), se han actualizado las técnicas de análisis basadas en *offsets* modificando las funciones que se encargan de calcular los *offsets* y *jitters* de las tareas (dando lugar a la versión *beta* de MAST). Estás técnicas solo están disponibles para los sistemas que utilizan la política de planificación basada en prioridades fijas.

Sin embargo, a la hora de desarrollar el análisis temporal del caso redundado 1oo2 se detectó que las técnicas de análisis no funcionaban correctamente. Este bug se detectó porque al realizar el análisis temporal, la herramienta muestra el tiempo de respuesta en cada evento de salida, y los eventos de salida finales no tenían el mismo valor.

Hay que tener en cuenta que los flujos *multipath* tienen un nodo de entrada y dos de salida. Los dos *path* de los flujos son idénticos, por lo tanto, los eventos finales deberían mostrar el mismo tiempo de respuesta. Y en este caso, no era así.

Al detectar este bug se informó al equipo de la Universidad de Cantabria para que lo corrigieran. Tras corregirlo y disponer de la nueva versión *beta*, se procedió a ejecutar los *benchmarks* relacionados con los flujos *multipath* para comprobar que el funcionamiento de los eventos *fork*, *join* y *merge* no se había visto afectado.

5.1 Casos de uso de *fork, join* y *merge*

En la documentación de la herramienta MAST se comentan las diferentes formas de usar los manejadores de eventos *fork, join* y *merge* para generar flujos *multipath*:

- *Fork* al principio del flujo

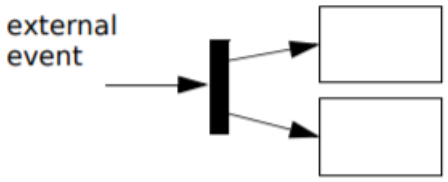

Ilustración 28: *Fork* al principio de un flujo

- *Fork* seguido de un *join*

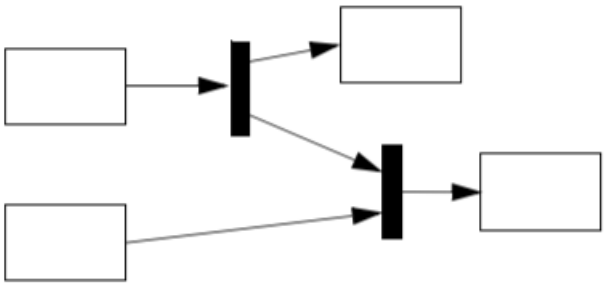

Ilustración 29: *Fork* seguido de un *join*

- *Fork* seguido de un *merge*

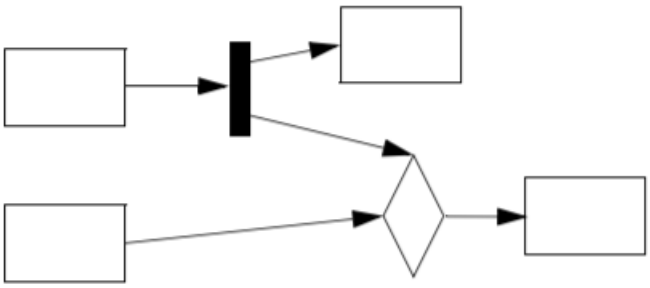

Ilustración 30: *Fork* seguido de un *merge*

- Merge seguido de un fork

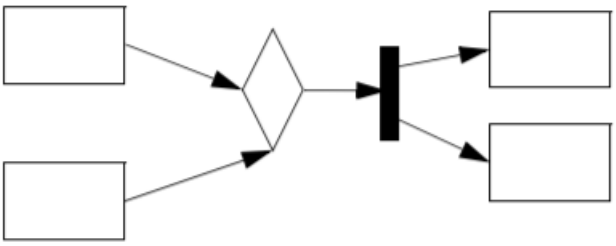

Ilustración 31: *Merge* seguido de un *fork*

Además de estos casos, también se describen algunos usos que no están permitidos. La mayoría de estos usos son casos cuya implementación podría realizarse de una forma más sencilla. Algunos de estos casos son utilizar un *fork,* un *join* o un *merge* al final del flujo, lo cual no tiene mucho sentido.

Sin embargo, hay un caso cuyo uso sí que puede ser útil pero implementarlo en la herramienta MAST es bastante complejo, por lo que se decidió no permitir su uso. Este caso es utilizar un *join* seguido de un *fork*.

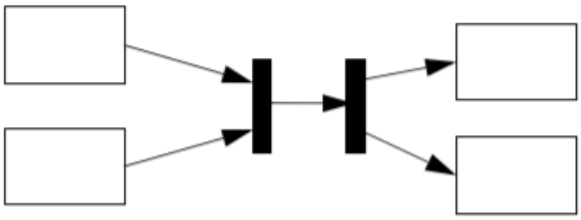

Ilustración 32: *Join* seguido de un *fork*

Para resolver este inconveniente, se ha propuesto utilizar la siguiente solución: implementar un procesador *dummy* (el tiempo por el cambio de contexto es cero) que tenga una tarea *dummy* (el tiempo de ejecución es cero).

Al utilizar la tarea *dummy* entre el *join* y el *fork* aseguras que el tiempo de respuesta de los eventos de salida del *join* y del *fork* tenga el mismo valor, ya que tanto el tiempo de ejecución como el cambio de contexto del procesador son cero.

5.2 Resultados de los *benchmarks*

Para comprobar que la versión *beta* funciona correctamente, se han comparado sus resultados para los *benchmarks* con la técnica holística con los obtenidos por la versión pública actual (1.5.1.0). También se han ejecutado los *benchmarks* para las tres técnicas de *offsets* implementadas en la versión *beta*. A continuación, se describen los diferentes *benchmarks* con los resultados obtenidos.

Benchmark: simple_fork.txt

En este caso, el *benchmark* consiste en un flujo e2e con tres actividades. Tras finalizar la ejecución de la primera actividad se utiliza un *fork* para generar dos caminos. En cada camino se ejecuta una de las actividades restantes.

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	511.72%	19.3 ms	11.5 ms
Técnicas basadas en *offsets*	511.72%	16.3 ms	8.5 ms

Tabla 5: Resultados *benchmark* simple_fork.txt

Benchmark: simple_fork_2proc.txt

Este *benchmark* se basa en el *benchmark* 'simple_fork.txt'. La diferencia está en que las dos últimas actividades del flujo las ejecutan dos procesadores diferentes.

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	649.61%	16.3 ms	8.5 ms
Técnicas basadas en *offsets*	649.61%	16.3 ms	8.5 ms

Tabla 6: Resultados *benchmark* simple_fork_2proc.txt

Benchmark: simple_fork_at_the_start.txt

En ese caso el *benchmark* consiste en un flujo e2e con dos actividades. Al principio del flujo se utiliza un *fork* para que en cada camino se ejecute una actividad.

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	649.61%	13.3 ms	5.5 ms
Técnicas basadas en *offsets*	649.61%	13.3 ms	5.5 ms

Tabla 7: Resultados *benchmark* simple_fork_at_the_start.txt

Benchmark: simple_fork_followed_by_join.txt

El objetivo de este *benchmark* es probar que se puede utilizar un *fork* seguido por un *join*. En total, en el flujo se generan tres caminos pero se utiliza el *join* para unir dos de ellos.

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	649.61%	57.4 ms	53.1 ms
Técnicas basadas en *offsets*	649.61%	57.4 ms	53.1 ms

Tabla 8: Resultados *benchmark* simple_fork_followed_by_join.txt

Benchmark: simple_fork_followed_by_merge.txt

Este *benchmark* se basa en el *benchmark* 'simple_fork_followed_by_join.txt' y la diferencia entre ambos es que en vez de utilizarse un *join*, se utiliza un *merge*

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	19.53%	98.5 ms	94.2 ms
Técnicas basadas en *offsets*	19.53%	93.0 ms	94.2 ms

Tabla 9: Resultados *benchmark* simple_fork_followed_by_merge.txt

Benchmark: simple_fork_join.txt

En este *benchmark* se utiliza tanto el evento *fork*, como el *join*. Entre ambos eventos hay una actividad ejecutándose.

Técnica utilizada	*Slack*	Evento final 1
Técnica holística (ambas versiones de MAST)	237.50%	33.2 ms
Técnicas basadas en *offsets*	237.89%	33.2 ms

Tabla 10: Resultados *benchmark* simple_fork_join.txt

Benchmark: simple_fork_join_3proc.txt

Este caso, se basa en el *benchmark* 'simple_fork_join.txt', con la diferencia de que cada procesador se encarga de ejecutar las actividades de un camino.

Técnica utilizada	*Slack*	Evento final 1
Técnica holística (ambas versiones de MAST)	649.61%	17.5 ms
Técnicas basadas en offsets	649.61%	17.5 ms

Tabla 11: Resultados *benchmark* simple_fork_join_3proc.txt

Benchmark: simple_fork_merge_3proc.txt

Se basa en el *benchmark* 'simple_fork_merge.txt' con la diferencia de que cada procesador se encarga de ejecutar las actividades de un camino.

Técnica utilizada	*Slack*	Evento final 1
Técnica holística (ambas versiones de MAST)	649.61%	18.7 ms
Técnicas basadas en *offsets*	649.61%	17.5 ms

Tabla 12: Resultado *benchmark* simple_fork_merge_3proc.txt

Benchmark: simple_merge_followed_by_fork.txt

El objetivo de este *benchmark* es utilizar un *merge* seguido por un *fork*

Técnica utilizada	*Slack*	Evento final 1	Evento final 2
Técnica holística (ambas versiones de MAST)	110.16%	63.7 ms	18.7 ms
Técnicas basadas en *offsets*	110.16%	58.2 ms	17.5 ms

Tabla 13: Resultado *benchmark* simple_merge_followed_by_fork.txt

Benchmark: simple_join_followed_by_fork.txt

El objetivo de este *benchmark* es utilizar un *join* seguido por un *fork*. Como se ha comentado anteriormente, este caso no está permitido en MAST y por ello se ha propuesto un solución que se basa en utilizar un procesador y una tarea *dummy*.

Para describir este *benchmark* se ha generado la siguiente ilustración:

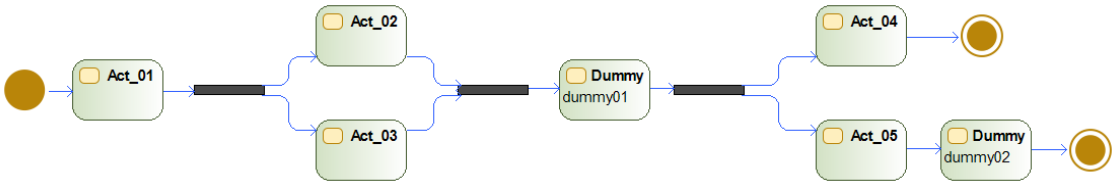

Ilustración 33: Flujo e2e del *benchmark* simple_join_followed_by_fork.txt

Como se puede observar, la actividad 'Dummy' se utiliza dos veces. El propósito de esto es probar que esta actividad se puede utilizar tantas veces como sea necesario ya que no va alterar el resultado.

Test	Evento antes de ejecutar dummy1	Evento después de ejecutar dummy1	Evento final tras ejecutar Act_04	Evento antes de ejecutar dummy2	Evento final tras ejecutar dummy2
1	16.3 ms	16.3 ms	40 ms	17.5 ms	17.5 ms
2	31.3 ms	31.3 ms	60 ms	37.5 ms	37.5 ms
3	31.3 ms	41.3 ms	70 ms	47.5 ms	Holística ➔ 57.5 ms Basadas en *offsets* ➔ 52.5 ms

Tabla 14: Resultados *benchmark* simple_join_followed_by_fork.txt

En total se han realizado tres pruebas. En todas ellas se han utilizado las cuatro técnicas de análisis y han dado los mismos resultados (excepto en un caso). La primera prueba se ha realizado especificando que el tiempo por cambio de contexto en los planificadores de los procesadores fuese 0.

La segunda prueba se ha realizado indicando que el tiempo por cambio de contexto fuese 2.5. Este cambio se ha realizado en todos los planificadores excepto en el del procesador *dummy.*

Por último, en la última prueba se ha indicado que el tiempo por cambio de contexto en el planificador *dummy* fuese 2.5 (igual que en los otros planificadores). La única razón de que se haya realizado esta prueba es para justificar que la tarea y el procesador *dummy* deben tener los valores que tienen por defecto, ya que si se modifican, no cumplirán su propósito de forma adecuada.

Benchmarks de los modelos del dron

Teniendo en cuenta la complejidad de los modelos del dron que se han desarrollado durante este TFM, se han propuesto como *benchmarks* para futuras pruebas. Además, basándonos en el modelo redundado se han generado otros tres *benchmarks*.

En estos *benchmarks*, se han remplazo los eventos *join* (por eventos *merge*) de los flujos que se encargan de gestionar la energía o actualizar los parámetros de vuelo. Dependiendo del *benchmark* se altera el flujo principal de una forma u otra.

Actualmente, en el flujo principal hay cuatro *join* (dos *join* antes de establecer la comunicación con EPS y otros dos antes de enviar los comandos a los motores) y la idea de estos *benchmarks* es remplazar algunos de los *join* por *merge*.

Benchmark: dron-100-Merge.txt

En esta prueba, solo se remplazan los dos últimos *join* (los que hay antes de enviar los comandos a los motores) por dos *merge*.

Técnica utilizada	*Slack* del sistema	Utilización MMS	Utilización EPS	Flujo Principal	Cambio Coordenadas	Gestión energía
Algoritmos basados en *offsets*	162.11%	28.34%	31.90%	6.64 ms	25.74 ms	127.4 ms
Holística (ambas versiones de MAST)	-5.47%			21.1 ms	41.97 ms	324.67 ms

Tabla 15: Resultados *benchmark* dron-100-Merge.txt

Benchmark: dron-100-Merge-Join-Error.txt

En este *benchmark* se ha remplazo los dos primeros *join* (los que hay antes de establecer la comunicación con EPS) por dos *merge*.

Al realizar el análisis la herramienta MAST ha informado de un error debido a una inconsistencia en el modelo. Esta inconsistencia se debe a que los dos *merge* (con varios eventos de entrada), pueden generar eventos de salida a un ritmo superior al periodo de activación del flujo e2e. Y los *join*, que están situados después de los *merge*, requieren que todos sus eventos de entrada tengan el mismo periodo.

Benchmark: dron-100-Merge-Merge.txt

En este caso, se ha remplazado todos los *join* del flujo principal por *merge*.

Técnica utilizada	*Slack* del sistema	Utilización MMS	Utilización EPS	Flujo Principal	Cambio Coordenadas	Gestión energía
Algoritmo basado en *offset*	80.47%	32.34%	50.00%	11.06 ms	33.36 ms	169.2 ms
Algoritmos basados en *offsets slanted* y fuerza bruta	80.47%			11.06 ms	33.36 ms	159.96 ms
Holística (ambas versiones de MAST)	-39.84%			33.76 ms	54.43 ms	465.4 ms

Tabla 16: Resultados *benchmark* dron-100-Merge-Merge.txt

6 Conclusiones

El objetivo de este TFM era modelar y analizar el sistema propuesto por el *challenge* WATERS 2018 pero durante el análisis del sistema se descubrió el sistema propuesto no era viable por varias razones.

Algunas de estas razones son el uso del patrón maestro-esclavo para gestionar todos los subsistemas del dron y la falta de algunos flujos e2e. Se considera que parte del *challenge* consiste en actualizar el diseño que proponen, pero el caso es, que el sistema, con los datos que proponen, no es planificable.

Debido a los requisitos temporales que proponen, el sistema no es planificable y probablemente actualizar los requisitos no forme parte del *challenge*. Por lo tanto, se decidió implementar dos casos redundados: uno lineal y otro utilizando el esquema 1oo2.

Ambos casos se basan en el sistema que se propone en el *challenge*: tienen dos de los tres subsistemas y utilizan tres de los siete flujos que se describen en la documentación del *challenge*.

La diferencia entre estos casos es que uno de ellos utiliza flujos e2e *multipath* para utilizar el esquema 1oo2 y mejorar el nivel SIL del sistema.

Al comparar los análisis temporales de ambos casos se observó que los resultados eran similares, aunque el *slack* del caso que utiliza el esquema 1oo2 es menor. Esto es normal porque la complejidad de este caso es mayor que la del caso lineal.

Teniendo en cuenta esto, se decide que para una futura implementación, la mejor opción es utilizar el caso que utiliza el esquema 1oo2, no solo por los beneficios comentados anteriormente, sino porque también se cumplen los objetivos del *challenge*.

Por último, para finalizar este TFM, se ejecutaron todos los *benchmarks* de la herramienta MAST que tienen relación con los flujos *multipath,* con el objetivo de comprobar que la versión *beta* de MAST funcionaba correctamente. Además, se desarrollaron otros *benchmarks* con el objetivo de cubrir otros casos de usos a la hora de modelar un flujo e2e.

6.1 Trabajos futuros

El objetivo inicial de este TFM era modelar y analizar el sistema propuesto por el *challenge* WATERS 2018. Pero como se detectó que el sistema propuesto no es planificable se desarrollaron dos modelos basándose en este sistema.

A continuación, se comentan los trabajos que se realizarán en el futuro teniendo en cuenta que el sistema utilizará una arquitectura de redundancia dual utilizando el esquema 1oo2.

- Definir la arquitectura hardware del sistema. Este modelo se ha desarrollado teniendo en cuenta los componentes principales del sistema. Habría que diseñar el sistema para su futura implementación teniendo en cuenta la posibilidad de implementar otros componentes como HBS o el panel de control.
- Definir las tareas y flujos *end to end* del sistema. Actualmente se han definido las tareas pero se desconocen sus funciones y su tiempo de ejecución. Además, sabemos que en el *challenge* no se han definido todos los flujos del sistema.

  El objetivo sería definir todas las tareas y los flujos e2e del sistema partiendo desde cero. Además habría que empezar a definir los requisitos temporales del sistema.
- Implementar las tareas del sistema para conocer su tiempo de ejecución real.
- Modelar y analizar el sistema para comprobar si es planificable y realizar cambios en caso de que no sea así.
- Implementar el dispositivo y realizar pruebas reales.

7 Bibliografía

[1] J.A. Stankovic and K. Ramamritham, "Hard Real-Time Systems", IEEE Computer Society, catalog no. EH0276-6, 1988

[2] Object Management Group, "UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems". OMG Document, v1.1 formal/2011-06-02 (2011).

[3] M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake, "MAST: Modeling and Analysis Suite for Real Time Applications", Proc. of the 13th Euromicro Conference on Real-Time Systems, Delft (The Netherlands), págs. 125-134, 2001.

[4] M. González Harbour, J.J. Gutiérrez, J.M. Drake, P. López and J.C. Palencia, "Modeling distributed real-time systems with MAST 2", Journal of Systems Architecture, vol 56, no. 6, Elsevier, págs. 331-340, 2013

[5] MAST, http://www.mast.unican.es

[6] K.W. Tindell and J. Clark, "Holistic Schedulability Analysis for Distributed Hard Real-Time Systems", Microprocessing and Microprogramming, vol. 50, no. 2-3, págs. 117-134, 1994.

[7] J.C. Palencia Gutiérrez, J.J. Gutiérrez García, M. González Harbour. "On the schedulability analysis for distributed hard real-time systems". 9th Euromicro Workshop on Real-Time Systems. Toledo, 1997.

[8] K.W. Tindell, "Adding Time-Offsets to Schedulability Analysis", Department of Computer Science, University of York, Technical Report YCS-221, 1994.

[9] J.C. Palencia and M. González Harbour, "Schedulability Analysis for Tasks with Static and Dynamic Offsets", Proc. of the 19th Real-Time Systems Symposium (RTSS), 1998.

[10] J. Mäki-Turja and M. Nolin, "Efficient implementation of tight response-times for tasks with offsets", Real-Time Systems Journal, vol. 40, no. 1, págs. 77-116, 2008.

[11] WATERS 2018 (Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems), https://w3.onera.fr/waters2018

[12] ECRTS (Euromicro Conference on Real-Time Systems), https://www.ecrts.org

[13] AADL, http://www.aadl.info

[14] OSATE, https://osate.org

[15] Mathieu Bouvier and Ian Yellowley, "International Journal of Manufacturing Technology and Management", 2006, vol. 8 (A distributed reconfigurable architecture for fault-tolerant systems)

[16] Andoni Amurrio, Ekain Azketa, J. Javier Gutiérrez, Mario Aldea y Michael González Harbour; "Response-Time Analysis of Multipath Flows in Hierarchically-Scheduled Time-Partitioned Distributed Real-Time Systems". Documento interno (pendiente de publicación).

[17] IEC, "IEC 61508: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 1: General requirements", 2010

[18] E. Pastor, J. Lopez and P. Royo, "A Hardware/Software Architecture for UAV Payload and Mission Control," 2006 ieee/aiaa 25TH Digital Avionics Systems Conference, Portland, OR, 2006, págs. 1-8

[19] T. Kekec, B. C. Ustundag, M. A. Guney, A. Yildirim and M. Unel, "A modular software architecture for UAVs," IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society, Vienna, 2013, págs. 4037-4042

[20] Juan A. Besada, Luca Bergesio, Iván Campaña, Diego Vaquero-Melchor, Jaime López-Araquistain, Ana M. Bernardos and José R. Casar, "Drone Mission Definition and Implementation for Automated Infrastructure Inspection Using Airborne Sensors", 2018

[21] David John Smith, Kenneth G. L. Simpson, "Functional Safety: A straightforward guide to applying IEC 61508 and related standards", Butterworth-Heinemann, 2004

[22] J.J. Gutiérrez, J.C. Palencia y M. González Harbour, "Schedulability analysis of distributed hard real-time systems with multiple-event synchronization", in Proceedings 12th Euromicro Conference on Real-Time Systems. Euromicro RTS 2000. IEEE, 2000, págs. 15–24.

8 Anexo I: Flujos e2e del *challenge*

En este anexo se describen los flujos e2e del *challenge* WATERS 2018 que no se han utilizado en los modelos que se han desarrollado.

8.1.1 Modo degradado donde MMS falla

Ilustración 34: Flujo principal cuando MMS falla del *challenge*

Este flujo se activa cuando MMS1 y MMS2 han fallado y EPS01 se convierte en el subsistema maestro del dron. Al igual que el flujo principal descrito en el punto 2.2.1, este tiene un periodo de 20 milisegundos. La tarea CM del subsistema EPS01 se encarga de procesar los datos que han enviado los sensores.

A continuación, la FC genera los siguientes comandos de vuelo teniendo en cuenta todos los datos recabados hasta el momento. Después, se analiza la situación actual de la energía del dron (EM) y se transforman los comandos de vuelo en comandos de voltaje (PC) para transmitirlos a los motores (CM).

8.1.2 Comprobación de estado

Ilustración 35: Flujo de comprobación de estado del *challenge*

Todos los subsistemas ejecutan este flujo para testear su funcionamiento y poder detectar posibles fallos. Cada 50 milisegundos, la tarea HM evalúa la situación del subsistema. En el caso de que se detecte un fallo, la tarea FT puede realizar dos acciones: corregir el problema o inhibir el funcionamiento del subsistema.

En el caso de que se inhiba el funcionamiento del subsistema, se transmite un mensaje *broadcast* informando del evento para que el resto de subsistemas puedan registrarlo y realizar las acciones pertinentes.

8.1.3 Reconfiguración en caso de fallo

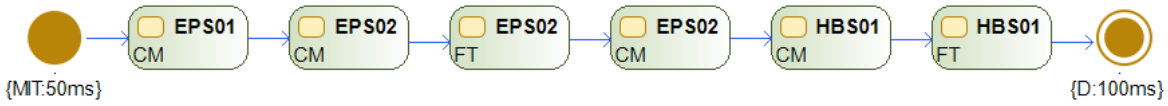

Ilustración 36: Flujo de reconfiguración en caso de fallo del challenge

Todos los subsistemas disponen de un flujo que se encarga de reconfigurar el funcionamiento de su subsistema en caso de que otro componente informe de un error.

Cuando un componente falla, se puede informar del error de dos formas:

- El flujo que comprueba el estado del subsistema envía un mensaje *broadcast* informando del evento.
- El subsistema que le está monitorizando detecta que no funciona correctamente y se encarga de enviar las señales y los mensajes al resto de subsistemas para que realicen las acciones necesarias.

En la ilustración anterior, se muestra el caso en el que EPS01 informa del error y EPS02 reconfigura su funcionamiento. La tarea FT se encarga de realizar la reconfiguración del subsistema y en cuanto finaliza con la reconfiguración, envía un mensaje a HBS01 para informar de los cambios y que se reconfigure.

8.1.4 Aterrizaje

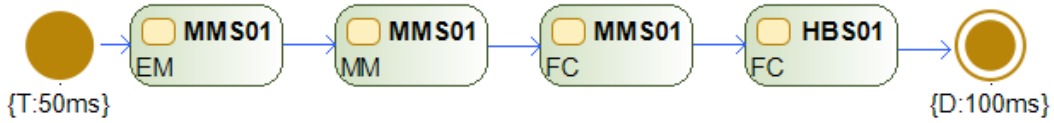

Ilustración 37: Flujo de aterrizaje del challenge

Este flujo se activa cada 50 milisegundos y se encarga de controlar el aterrizaje del dron. En la ilustración se muestra el caso en el que se realiza un aterrizaje en situaciones normales.

Inicialmente, se ejecuta la tarea EM para recopilar todos los datos relacionados con la gestión de la energía. A continuación, la tarea MM evalúa la situación de la misión por si es necesario realizar alguna acción.

- Si hay poca energía y no se ha llegado al destino, se realiza un aterrizaje de emergencia.
- Si hay poca energía y se ha llegado al destino, se realiza un aterrizaje de emergencia.
- Si hay energía suficiente y se ha llegado al destino, se realiza un aterrizaje normal.
- Si hay energía suficiente y no se ha llegado al destino, se continúa con la misión.

En el ejemplo que se muestra en la ilustración, se realiza un aterrizaje normal. Por lo tanto, se generan los comandos de vuelo para realizar el aterrizaje. En la ilustración, las tareas FC de MMS01 y HBS01 se comunican directamente.

Además, no es necesario que se ejecute la tarea FC del subsistema HBS01 porque ya se ha ejecutado en MMS01. Consideramos que es una errata y que la tarea que se debe ejecutar es BC que se encarga de transformar los comandos de vuelo en comandos que los frenos hidráulicos puedan interpretar. Por lo tanto, el flujo quedaría tal como se muestra en la ilustración 38.

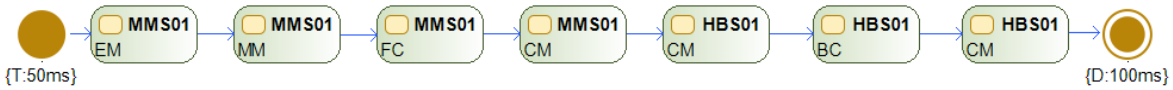

Ilustración 38: Corrección del flujo de aterrizaje del *challenge*