Proceso de verificación y validación de software embarcado en satélite: una estrategia basada en propiedades de composición

Tesis Doctoral

Javier Fernández Salgado

Departamento de Automática
Escuela Politécnica Superior
Universidad de Alcalá

Alcalá de Henares, junio 2016

Proceso de verificación y validación de software embarcado en satélite: una estrategia basada en propiedades de composición

Tesis Doctoral

Javier Fernández Salgado

Directores:

Dr. Oscar Rodríguez Polo, **Dr. Sebastián Sánchez Prieto**

Departamento de Automática
Escuela Politécnica Superior
Universidad de Alcalá

Alcalá de Henares, junio 2016

Agradecimientos

Esta Tesis Doctoral ha supuesto un trabajo intenso en los últimos años de mi vida, en la que he invertido noches sin dormir, viajes por diferentes países —estancias de investigación y conferencias—, preocupaciones, y muchas horas de trabajo con el objetivo de producir una contribución original en un campo de investigación que me apasiona como es el de sistemas de tiempo real. No hubiese conseguido confeccionarla sin la ayuda, el amor y la dedicación de las personas qué me queréis y me habéis ayudado en estos intensos años. Creo que como mínimo es nombraros en estas líneas, ya que parte de este trabajo es vuestro. Siempre os llevaré en mi corazón.

A mis padres y mi familia, un gran motivo de que estas líneas estén impresas es gracias a vuestra dedicación, apoyo económico y cariño incondicional — mi padre: Albino Fernández Romero de Castilla, mi madre: M. José Salgado Granado y mi hermano: Álvaro Fernández Salgado—, ya lo sabéis, pero quisiera repetiros que os quiero incondicionalmente. A Claudia Bidian por tu cariño y paciencia siendo mi pareja en unos años tan duros, nunca lo olvidaré, muchas gracias «Pecas». A la Dra. Pilar Fernández Fernández por tu cariño, ánimo y apoyo. A D. José David Sánchez Melero por tus sabias enseñanzas sobre argumentación y negociación. A Dña. Belinda por tus increíbles revisiones bibliográficas. Al Dr. Raúl Gómez Herrero y al Dr. Juan José Blanco Ávalos, por vuestra comprensión, cariño y apoyo, gran parte del mérito que entregue esta Tesis es gracias a vuestros buenos consejos y ánimos en los peores momentos. Al Dr. Antonio Guerrero Ortega y a la Dra. Judith Palacios Hernández, por todos los ánimos y agradables comidas que hemos tenido en el último año de mi Tesis. A Dña. Marta Rodríguez Peleteiro, por tus sabios consejos sobre publicación y tus revisiones ortográficas. Cuando no contabas con tiempo material sacastes hueco para leer entera la Tesis Doctoral. Al Dr. Guillem Bernat, al Dr. César Martín, al Dr. José María Drake Moyano y al Dr. Michael González Harbour, por vuestras discusiones sobre temas de investigación, vuestra dedica-

ción y compartir conmigo vuestra visión en este campo, los conocimientos que tengo en esta materia os lo debo a vosotros. A mis abuelas Salud y Antonia. Por vuestro apoyo, amor y cariño — y por supuesto a vuestra pequeña pero valiosa financiación económica—. A todos los desarrolladores de software libre, que con su dedicación y solidaridad nos brindan la posibilidad de tener herramientas de un valor incalculable sin coste económico, como son el caso de Emacs, R, Python y otras muchas, usadas en esta Tesis. He querido terminar con estos agradecimientos con dos personas, mis tutores de Tesis Doctoral, Dr. Sebastián Sánchez Prieto y Dr. Óscar Rodríguez Polo. Gracias Sebastián por la financiación, por el apoyo y las sabias palabras que has tenido conmigo, espero, en el futuro, poder volver a compartir experiencias de investigación contigo. Gracias Dr. Óscar Rodríguez Polo por ser mi tutor estos años.

Mis agradecimientos al Ministerio de Ciencia e Innovación. Este trabajo ha sido financiado con los proyectos AYA2009-13478-C02-02, AYA2011-29727-C02-02 y AYA2012-39810-C02-02 teniendo todos ellos como titulado «SOLAR ORBITER: CONSOLIDACIÓN DEL GRUPO EPD Y TECNOLOGÍAS DE SOPORTE». Como becario de formación FPI me fue concedida una beca con código administrativo BES-2010-042797.

Me gustaría concluir con una cita de John Lennon «La vida es aquello qué te pasa mientras estás ocupado haciendo otros planes» vosotros habéis sido mi vida estos últimos años. Se que aveces no ha sido fácil, pero os quiero, muchas gracias por todo lo bueno que me habéis dado.

Índice general

Índice de figuras

Índice de cuadros

Resumen

Resumen en español

Las últimas estimaciones de costes del proceso de verificación y validación (VV) de sistemas de tiempo real críticos están entorno al 50 % de coste del total economico de los proyectos, siendo incluso mayor en algunos casos. Esta fase es tan costosa debido a las extensas pruebas, que tienen como objetivo comprobar que el sistema cumple con todos sus requisitos. Entre estos requisitos en sistemas de tiempo real, están aquellos que dependen de parámetros cuya naturaleza es de carácter extrafuncional. Uno de estos parámetros es el tiempo de ejecución. [1] Por este motivo se realizan diferentes análisis, como son el análisis de planificabilidad, análisis de rendimiento y la caracterización del *Worst-Case Execution Time Profile* (WCET). Todos estos análisis se encuadran dentro del ciclo de vida software en la fase de VV. Veremos en este trabajo que una aproximación de generación manual de modelos de análisis en la fase de VV conlleva varios riesgos y costes asociados. Riesgos que pueden ser atenuados gracias al uso de técnicas basadas en el paradigma *Model-Driven Engineering* (MDE). Esta técnica permite que cada elemento del proceso de VV se integre de una manera coherente y cohesionada permitiendo reducir su coste económico.

En el presente trabajo hemos planteado la siguiente hipótesis de trabajo: los elementos de análisis imprescindibles en la fase de VV generados de una manera automática reducen el coste económico de esta fase. Entre estos elementos están el análisis de planificabilidad, el análisis de rendimiento y la caracterización de los tiempos de ejecución. La calidad de estos modelos de análisis depende del proceso que realiza la transformación entre los diferentes modelos. Para poder cuantificar la calidad de estos modelos hemos propuesto un proceso de verifica-

[1] El correcto funcionamiento de sistemas de tiempo real no sólo depende del resultado lógico que devuelve la computadora, también depende del tiempo en que se produce ese resultado.

ción que comprueba que los modelos de análisis corresponde con el modelo de diseño.

La hipótesis planteada ha sido desarrollada usando el paradigma MDE. Para ello ha sido usado el *framework* MICOBS, éste nos permitió definir con mayor comodidad el proceso propuesto. Parte del presente trabajo ha consistido en la definición de modelos intermedios en MICOBS que permiten integrar con mayor facilidad nuevos modelos de análisis. Esta aproximación permite cohesionar los modelos que constituye el proceso propuesto.

Como parte de los resultado se genera un modelo de planificabilidad, un modelo de rendimiento y un modelo de verificación. Cada uno de los modelos generados corresponde a una herramienta que es la encargada de realizar dicho análisis. El modelo de análisis de planificabilidad corresponde a la herramienta MAST. El modelo de análisis de rendimiento, corresponde a la herramienta PCM. El modelo de análisis de verificación corresponde a la herramienta RapiCheck. Todos estos modelos son generados a partir de un modelo de diseño denominado EDROOM el cual está basado en el paradigma CBSE.

Este proceso ha sido probado gracias a su uso en la descripción del ICUSW del instrumento EPD. Este instrumento va a bordo de la misión Solar Orbiter, una aventura conjunta entre las agencias espaciales *European Space Agency* (ESA) y *National Aeronautics and Space Administration* (NASA). En esta Tesis mostramos los modelos generados de análisis a partir de la descripción del modelo de diseño. Estos son: el modelo de análisis de planificabilidad, el modelo de análisis de rendimiento y el modelo de verificación. Estos resultados son usados para realizar tareas de VV bajo el estándar ECSS-E-ST-40C.

Abstract

Experts in real time of VV process estimate its economic cost near to 50 % of overall cost of the project. Even in under some conditions the costs can be increased. The cause of this economic cost is the intensive tests which aim is to probe the system convert their requirements. In real time system a set of properties are extra-functional type, the most prominent of this group is the execution time. For that reason in real time system the requirements are satisfied when the system respond to the proper action in the proper time. A set of analysis are required in VV form, where the timing requirements are checked. Usually this analysis are: scheduling analysis, performance analysis and WCET analysis. In this thesis we starting of the premise that the manual generation of analysis have a group of risks and associated economic costs. The disadvantages would have been mitigate thankfully to the use of MDE paradigm. This paradigm enables the cohesion among the elements in the VV process, there results is reduce the economic cost.

This thesis has set out the following hypothesis: The analysis elements which are essential in VV process, they are generated in automatic manner, reducing the economic cost. The essential analysis elements are: scheduling analysis, performance analysis and WCET analysis. In addition a verification process is added, its aim is boosting the reliability of the generated analysis elements.

The hypothesis suggested has been developed under the MDE paradigm. The framework MICOBS has been used with this intention, as well as it afford to define easier a VV process of real time system then other approach. A set of models are described in MICOBS to generate a analysis models from design models. The aim is cohesion among models and remove the their gaps. The proposed analysis models are decorated with extra-functional properties than being analyzed. The analysis models proposed in this thesis require the WCET then they are decorated with it. The annotation could be partial thankfully the proposed process has the properties of *composability* y *compositionality*.

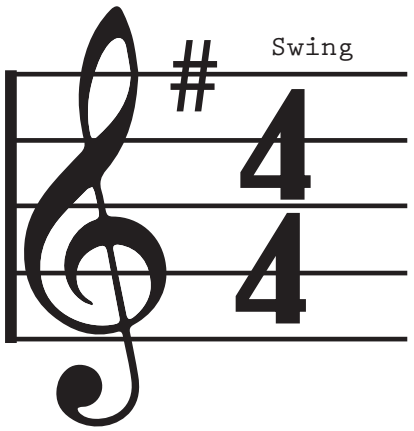
As part of the results a set of models are generated: a scheduling model, a performance model and verification model. These models are the entry of a tool. The scheduling analysis model is the entry model of MAST tool. The performance model is the entry of the PCM model. The verification model is the entry of `Rapicheck` model. The initial model is the design model which is EDROOM and it is based on CBSE paradigm.

The process has been proved in the ICUSW. It manages the the control unit of the instrument EPD which is on board of Solar Orbiter mission. The mission is coordinate adventure between ESA and NASA. This Thesis shows

the scheduling analysis, the performance analysis and the verification analysis. The results, which have been obtained, are used as part of the documentation of the process ICUSW VV.

Capítulo 1

Introducción



Dios no juega a los dados.

A. Einstein

Ground Control to Major Tom
Ground Control to Major Tom
Take your protein pills
and put your helmet on
Ground Control to Major Tom
Commencing countdown,
engines on
Check ignition
and may God's love be with you.

D. Bowie

«Suficientemente seguro» se ve
diferente a 35.000 pies de altura.

Libro de Douglass, regla 198

1.1. Contextualización

Los procesos de desarrollo de software pueden dividirse principalmente en dos fases: la fase de diseño e implementación y la fase de verificación y validación (VV). Existen metodologías de proceso de desarrollo de software, como la metodología en V, que representan esta división de una manera explícita. Durante la fase de VV se comprueba, por un lado, que los requisitos iniciales son factibles y, por otro, que están presentes en el producto resultante. La fase de VV tiene un impacto significativo en el coste del producto en los sistemas de tiempo real. Siendo difícil dar una estimación concreta (McDermid y Pumfrey, 2001), algunos autores apuntan a un coste de en torno al 50 % sobre el coste total del proyecto (Leveson, 1995).

En el presente trabajo de Tesis Doctoral se desarrollan los procesos de VV que comprueban los requisitos relativos a las restricciones de tiempo de respuesta y rendimiento en sistemas de tiempo real. En el contexto del proceso en «V», la verificación consiste en revisar y asegurar que cada etapa del proceso de desarrollo es consistente por sí misma. Puede ser ilustrado mediante una simple pregunta: ¿Estamos construyendo el proceso de una manera correcta? Por su parte, la validación es el proceso que comprueba que el sistema cumple los requisitos. Esto también puede ser ilustrado mediante una pregunta: ¿Estamos construyendo el producto correcto?

La definición de procesos software, MDE (Kent, 2002a), viene aplicándose con éxito durante la última década (Hugues et al., 2008; Montecchi y Lollini, 2011; Dearle, 2001; Crnkovic et al., 2009a). Una de sus propiedades más destacadas es la de fomentar la continuidad de las etapas que conforman los procesos de software (Grassi et al., 2007). Para ello utiliza la definición de modelos y las transformaciones de modelo a modelo. Esta continuidad tiene el efecto de atenuar casos perniciosos, como es la programación a la carrera (*rush to code*) (Selic et al., 1994).

Por otra parte, la ingeniería de software basada en componentes (*Component-Based Software Engineering* o CBSE) (Heineman y Councill, 2001) es usada frecuentemente junto con procesos MDE. El beneficio de este enfoque se debe a que el diseño y la construcción de sistemas se realizan mediante el empleo de elementos aislados denominados componentes, que se comunican a través de interfaces bien definidas. El empleo de componentes permite aumentar la cohesión y reducir el acoplamiento de los elementos que componen el sistema, facilitando de esa manera su reutilización.

En los últimos años han surgido nuevas líneas de investigación dentro del CBSE relacionadas con el análisis de *Extra-Functional Properties* (EFP) de los

componentes, y la introducción del paradigma de *correctness by construction* en las metodologías de desarrollo. En los procesos de VV de sistemas de tiempo real, su corrección no sólo viene dada por la comprobación de los requisitos funcionales, sino que se tienen en cuenta las EFP relacionadas con el cumplimiento de las restricciones temporales. La metodología de construcción por corrección, combinada con el paradigma CBSE, ha dado buenos resultados en el campo de sistemas de tiempo real. Por ejemplo, mediante su integración en *frameworks* (López et al., 2008) como en el estudio de su uso y el impacto en la certificación de componentes (Stafford y Wallnau, 2003; Hissam et al., 2003). También hay estudios más académicos como, por ejemplo, la definición de taxonomías de CBSE de las propuestas surgidas (Lau y Wang, 2007), su categorización en función de sus propiedades (Crnkovic et al., 2005), y cuáles son los nuevos retos que conlleva el uso de este tipo de tecnología (Crnkovi, 2003). Los principios de *composability* y de *compositionality* (Mazzini et al., 2008; Matic, 2008; Gössler y Sifakis, 2005) son especialmente interesantes para analizar las EFP en los sistemas construidos bajo el paradigma CBSE. Para que un sistema cumpla el principio de *composability*, los componentes que lo forman no pueden ver alteradas sus EFP por la acción de otros componentes. Asimismo, el principio de *compositionality* se cumple cuando las propiedades del sistema, en su conjunto, se puede obtener a partir de las propiedades de sus partes, esto es, de los componentes que lo forman.

Los procesos de VV de los sistemas de tiempo real en aplicaciones espaciales que siguen la norma de desarrollo software *European Cooperation for Space Standardization, Space Engineering, Software* (ECSS-E-ST-40C) son críticos dentro del proceso de desarrollo. Actualmente, el esfuerzo requerido para proveer suficientes evidencias es costoso en tiempo y dinero, motivado por el incremento continuo de la complejidad funcional, que se refleja en el tamaño del software. Esto conlleva un aumento de la criticidad de la fase de VV. En este contexto, aproximaciones que provean una validación basada en la generación automática de modelos de análisis, como el análisis de planificación y análisis de rendimiento, y procesos de verificación automática de evidencias, implican de por sí una reducción de los costes del desarrollo de las aplicaciones para el espacio. El objetivo del proceso de verificación es evitar aquellos casos que dan lugar a que los modelos de análisis y el sistema final no concuerden, como, por ejemplo: errores de inversión de prioridad, cambios de contexto superiores a las especificadas, *mutex* ocultos, etc.

El trabajo desarrollado en esta Tesis Doctoral tiene aplicación para procesos de desarrollo de software para sistemas empotrados. Presenta una solución general al proceso de VV de este tipo de sistemas desarrollados bajo los paradigmas

de CBSE y MDE, y aplica los principios de *compositionality* y *composability* para analizar las EFP de los sistemas construidos. Como ejemplo de uso se ha utilizado el software de control del instrumento EPD (Sánchez et al., 2012), que forma parte de la misión Solar Orbiter, en la que participan tanto la ESA y la NASA. Este software ha sido desarrollado por el grupo *Space Research Group* (SRG) de la Universidad de Alcalá. El software encargado de gestionar el flujo de datos y comportamiento del instrumento tiene que ser validado respecto al estándar ECSS-E-ST-40C (ECSS Secretariat, 2009). El proceso presentado en este trabajo tiene como objetivo ser usado en el proceso VV de este software de aplicación.

1.2. Objetivos

El objetivo general de este trabajo es presentar una solución al proceso de VV de sistemas software desarrollados bajo los paradigmas de CBSE y MDE que automatice las principales actividades de este proceso, y que permite así reducir el coste económico.

El trabajo se diferencia de otros que siguen el mismo enfoque en que la incorporación de la información requerida para el análisis depende de la plataforma de despliegue. Esto implica que el modelo de diseño no es anotado con las EFP, sino que un modelo de análisis generado automáticamente a partir de éste —el cual depende de la plataforma— es anotado. Éste se anota con la información de traza obtenida a partir de la ejecución de las pruebas unitarias para caracterizar los componentes. Este proceso se lleva a cabo basándonos en medidas obtenidas en los escenarios de prueba sobre cada plataforma, y evaluando el comportamiento de un sistema completo sobre distintas alternativas de despliegue. Además los productos resultantes en forma de modelos de análisis deberán ser verificados en las plataformas de despliegue donde han sido caracterizados. Con este propósito la verificación se realiza en las pruebas de integración. Partiendo de este punto, los objetivos específicos que se plantean son los siguientes:

- Generación automática, a partir de un modelo formal de diseño, del análisis de planificabilidad y del modelo de análisis de rendimiento en conformidad con el estándar ECSS-E-ST-40C.
- Asignación de los tiempos de ejecución a cada uno de los elementos que conforman el sistema.
- El tiempo de ejecución se analiza usando técnicas de Peor Tiempo de Ejecución WCET.

- La caracterización del peor tiempo de ejecución se realiza en función del análisis por separado de cada uno de los elementos que conforman el sistema.

- Las propiedades anotadas en el modelo de análisis se categorizan en función de la plataforma de despliegue.

- Los modelos de análisis son comprobados de una manera automática en el sistema final mediante un análisis de verificación de evidencias (modelo de vuelo o modelo de ingeniería).

1.3. Hipótesis

La hipótesis planteada es: la definición de un proceso de VV mediante técnicas MDE —que habilita la generación automática de los modelos de análisis de planificación, rendimiento partiendo del modelo de diseño, así como su verificación— conllevará la reducción de recursos y tiempos en el desarrollo de software empotrado, con especial atención en el software embarcado en satélite.

Para este último caso, el estándar de la Agencia Espacial Europea ECSS-E-ST-40C define como obligatoria la generación de los siguientes aspectos: análisis de planificabilidad, análisis de rendimiento, y el uso de técnicas WCET para caracterizar los tiempos de ejecución.

El uso de técnicas MDE permite que se puedan especificar transformaciones que habilitan la generación de los modelos de planificación, rendimiento, a partir del modelo de diseño, así como la verificación de que las asunciones hechas para el análisis son correctas.

El uso del paradigma CBSE permite desacoplar las funcionalidades del sistema y cohesionar la arquitectura. Gracias a este paradigma y al cumplimiento de un conjunto de restricciones de diseño, se pueden incorporar los principios de *composability* y *compositionality*, que permiten que el sistema sea analizable a partir de la caracterización de sus partes. Con el fin de obtener el análisis de planificabilidad y rendimiento empleando estos principios, se han incorporado herramientas que caracterizacion el WCET de cada una de las partes del sistema. El proceso se ha integrado en un *framework* que permite la indexación de las EFP en función de la plataforma. Finalmente, el modelo de verificación por generación de evidencias permite corroborar que las asunciones en los modelos de planificabilidad y rendimiento son las observadas durante las pruebas de integración.

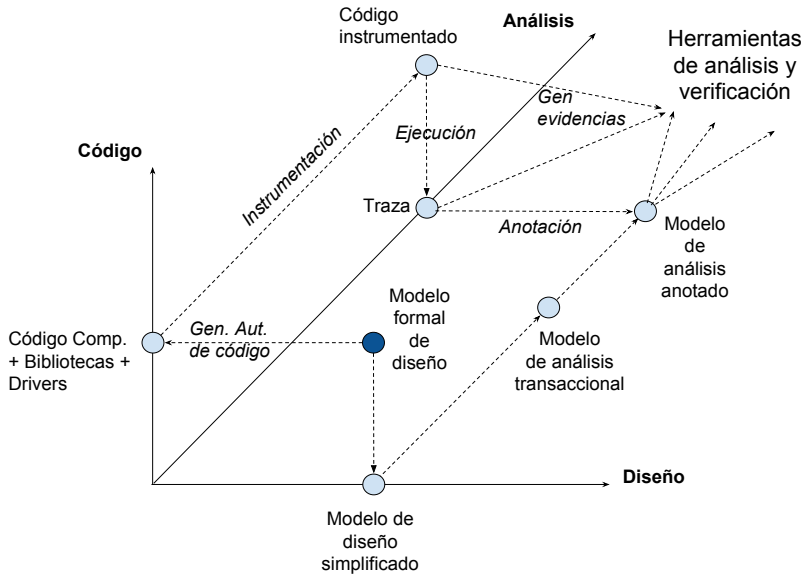


Figura 1.1: En la imagen se describen las transformaciones automáticas y sus resultados en forma de modelos. El resultado final de la generación de los modelos de análisis da lugar a los objetivos marcados en esta Tesis Doctoral.

En la figura 1.1 mostramos el proceso general de transformaciones automáticas. El proceso propuesto tiene como meta satisfacer los objetivos marcados en esta Tesis Doctoral. Las transformaciones entre los modelos transcurren entre un espacio de tres dimensiones: código, diseño y análisis. Tenemos como punto de partida el Modelo Formal de Diseño, que permite, por un lado, generar el código de los componentes y, por otro, resolver la inclusión de las distintas bibliotecas y *drivers*, de forma que lo que se obtiene sea un código compatible con la plataforma de despliegue elegida. El código es instrumentado para poder obtener una traza de la ejecución, a partir de las pruebas unitarias, de integración o funcionales. Del Modelo Formal de Diseño se obtiene también un Modelo de Diseño Simplificado que reduce la descripción del comportamiento para facilitar la transformación hacia un Modelo de Análisis Transaccional. En este modelo se hacen explícitas las transacciones del sistema, entendiendo por transacción el flujo end-to-end que define todos los pasos que dan respuesta a un evento recibido. Estos pasos comprenden la ejecución de operaciónes por parte de los componentes, así como el envío de mensajes entre ellos y el acceso a los diferentes recursos compartidos. Este modelo incorpora posteriormente la información de traza de ejecución de las distintas pruebas y de esa forma se obtiene un Modelo de Análisis Anotado. Éste habilita la utilización de distintas herramientas de análisis de planificabilidad, rendimiento y verificación, tal como se pretende demostrar en esta Tesis Doctoral. Un valor añadido de este enfoque es el de utilizar datos reales para anotar los elementos del modelo, obtenidos mediante una ejecución en la plataforma concreta sobre la que se pretende hacer el análisis. El Modelo de Análisis Anotado es utilizado para generar las entradas que requieren las distintas herramientas de análisis y verificación. Finalmente, el Modelo de Diseño desde el que se generaron el resto de modelos de la cadena es contrastado con un análisis de verificación.

El trabajo se diferencia de otros que siguen el mismo enfoque en que la incorporación de la información requerida para el análisis depende de la plataforma de despliegue y no se añade al modelo de diseño, sino que se utiliza la información de traza obtenida a partir de la ejecución de las pruebas unitarias para caracterizar los componentes. Esto permite caracterizar fielmente los componentes basándonos en medidas obtenidas en los escenarios de prueba sobre cada plataforma, y evaluar el comportamiento de un sistema completo sobre distintas alternativas de despliegue. Para dotar al trabajo del enfoque multiplataforma, se ha integrado en el *framework Multi-platform Multi-Model Component-Based Software Development Framework* (MICOBS), que expresamente incluye la plataforma de despliegue como una dimensión que permite indexar la información de análisis. MICOBS, además, modela fielmente los elementos que forman parte

del software basado en componentes, distinguiendo entre los componentes, que se ubican en el nivel de abstracción más alto del sistema y definen las tareas que cooperan para atender los distintos eventos, las bibliotecas de servicio, localizadas en el nivel intermedio y dedicadas al procesamiento de la información y la implementación de los algoritmos, y los *drivers*, que habilitan el acceso a los elementos hardware de la plataforma.

1.4. Descripción del proceso de VV

Como ya se ha comentado en la sección anterior, el proceso de VV propuesto en esta Tesis Doctoral ha sido integrado en el *framework* MICOBS, basado en los paradigmas MDE y CBSE. MICOBS se centra en la definición de los procesos de desarrollo de software para sistemas empotrados, como son los sistemas de tiempo real embarcados en satélite. La idea central de este *framework* es la definición del vector plataforma para categorizar los elementos que son definidos en el proceso, facilitando el análisis de los sistemas desarrollados, utilizando los principios de *composability* y *compositionality*. Esto significa que es posible anotar los valores de las EFP para las potenciales plataformas de despliegue. En la figura 1.2 mostramos cómo encaja el proceso que desarrollamos en esta Tesis Doctoral en un típico proceso en V.

MICOBS está implementado en *Eclipse Modelling Framework* (EMF), el cual se integra en *Eclipse Framework* (Eclipse). Esto implica que se usan los mismos mecanismos para definir los modelos y sus transformaciones. Los nuevos modelos que han sido integrados para definir el proceso son descritos bajo las definiciones de EMF. En el caso de las transformaciones se usa el lenguaje *Query, View, Transformation Language* (QVTO), teniendo la propiedad de definir transformaciones de una manera cómoda (Object Management Group, 2008).

Como modelo formal de diseño ha sido escogido el modelo de componentes EDROOM (Viana et al., 2006; Polo et al., 2002). Se trata de una herramienta de diseño y generación automática de código que ha sido utilizada con éxito por el grupo SRG en distintos proyectos de desarrollo de software embarcado en satélite (Polo et al., 2012a). El modelo de componentes permite describir su comportamiento a partir de máquinas de estados cuyas transiciones están disparadas por la recepción de mensajes.

El proceso propuesto empieza una vez definido el diseño del sistema con EDROOM. El modelo es transformado a un modelo orientado al análisis, definido como parte de este trabajo de Tesis Doctoral, denominados TSAM. Este

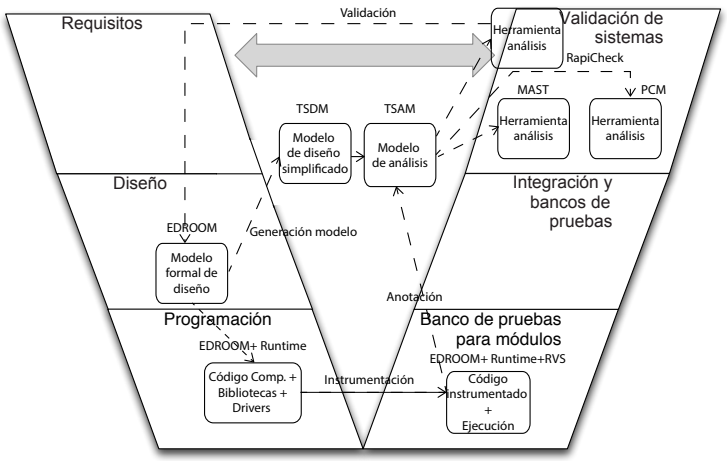


Figura 1.2: En esta figura mostramos el proceso de desarrollo propuesto bajo el enfoque en V. Los modelos superpuestos corresponden a la cadena de modelos y a sus transformaciones. El proceso en V es una típica representación gráfica del ciclo de vida del software. Resume los pasos principales que hay que tomar en conjunción con las correspondientes entregas de los sistemas de validación. La parte izquierda de la V representa la corriente donde se definen las especificaciones del sistema (etapas de requisitos y diseño). La parte derecha de la V representa la corriente donde se comprueba el sistema (contra las especificaciones definidas en la parte izquierda, etapas de validación del sistema e integración y banco de pruebas). La parte de abajo, donde se encuentran ambas partes, representa la corriente de desarrollo.

modelo permite definir las transacciones reactivas que especifican la respuesta del sistema. Además, permite describir los distintos patrones de activación de los eventos que puedan tener lugar dentro del sistema. Este modelo se ha definido de forma que no dependa del modelo de diseño, siempre que éste cumpla con un conjunto de restricciones definidas. El modelo, además, permite obtener distintos modelos de análisis, por lo que hace el papel de modelo pivote (Balp et al., 2008).

Una vez descritos el comportamiento reactivo de los componentes (mediante el modelo transaccional), su WCET tiene que ser caracterizado y anotado como EFP. Para poder realizar esta caracterización, es necesario integrar una herramienta de análisis que permita obtener los datos correspondientes. Esta herramienta es *Rapita Verification Suite* (RVS)(RapiTime, 2007), que permite realizar las medidas sobre la propia plataforma o *target* en la que los componentes se despliegan. Para realizar la anotación de propiedades se pueden emplear las mismas pruebas unitarias y de integración empleadas para la validación funcional de los componentes. El propio proceso de anotación ha sido automatizado mediante el uso de transformaciones de modelos. La transformación extrae de una manera automática la información de las bases de datos generada por las herramientas RVS.

Tras ser anotado el TSAM, las transformaciones a los modelos de análisis de planificabilidad MAST, análisis de rendimiento PCM y análisis de verificación RapiCheck (Ltd., 2014) son habilitadas. El modelo de análisis de planificabilidad permite validar las restricciones temporales asociadas a los eventos del sistema. El análisis de rendimiento nos permite validar la sensibilidad del sistema y los tiempos de respuesta. Por último, se ha planteado un análisis de verificación que tiene como objetivo comprobar que en el sistema se cumplen las asunciones de las que parte el TSAM, como la frecuencia de los eventos, y tiempos de respuesta relativos a la plataforma. La verificación pretende comprobar que no existen condiciones comunes de error que invalidan los modelos de análisis. El objetivo es que la generación de evidencias y su comprobación sean automáticos, ya que un proceso manual sería extremadamente costoso y tendería a generar fallos inherentes. Sobre todo son de especial interés aquellos errores difíciles de reproducir. Éstos, por naturaleza, son extremadamente engorrosos de encontrar y replicar.

Este proceso está siendo usado para el proceso de VV del software de aplicación de la unidad de control del instrumento EPD. Los modelos de análisis obtenidos se entregarán a la ESA como parte de la documentación requerida.

1.5. Cronografía de la Tesis Doctoral

Esta Tesis Doctoral ha sido financiada por el Ministerio de Economía y Competitividad, mediante el proyecto con código «AYA2009- 13478-C02-02». De manera periódica, como requisito para obtener la ayuda, se fueron entregando los hitos del trabajo de investigación propuestos y su estado.

Como parte de la documentación entregada caben destacar los informes de los primeros 24 meses —dos años— y el informe de los 48 meses —cuatro años—. Además, con cada estancia realizada en el extranjero, financiadas con los fondos para ayudas de «Estancias Breves en el Extranjero», fueron entregados objetivos de la estancia y, al retorno de la misma, se elaboró un informe de objetivos cubiertos y nuevas propuestas, en caso de haberse generado.

Las publicaciones relativas a la Tesis Doctoral también forman parte de los hitos. Por el momento han sido publicadas tres comunicaciones en congreso en forma de *Proceedings*, donde en (Fernández-Salgado et al., 2013a) y (Fernández-Salgado et al., 2013b) se presentó el modelo transaccional y su transformación al modelo de análisis de planificabilidad, y en (Fernández-Salgado et al., 2015) se presentó el modelo de verificación. Además, cabe destacar una publicación en revista (Fernández-Salgado et al., 2016) en la cual se describió una extensión en PCM (éste formará parte del proceso como herramienta de análisis de rendimiento) para que sea compatible con la definición de sistemas de tiempo real. Un total de dos publicaciones y una comunicación a congreso como coautor también forman parte de los resultados (Sánchez et al., 2013; Sánchez et al., 2012; Polo et al., 2012a).

Con la información de la documentación aportada al ministerio y con las publicaciones ha sido elaborado un cronograma con los hitos. En la figura 1.3 se muestra en forma de gráfico este cronograma, el cual detallamos en las siguientes secciones, dividido por años.

1.5.1. Fase exploratoria (primer año)

En esta primera fase se revisó la literatura relacionada con el tema de investigación a fin de disponer de un marco teórico sólido sobre el que se sustentó el trabajo. Esta primera fase estuvo marcada por la estancia en una *spin-off* de la Universidad de York denominada Rapita System Ltd. Esta empresa es uno de los entes promotores observadores que se incluyó en la petición del proyecto. El objetivo marcado fue la exploración de la viabilidad del análisis del tiempo de respuesta y de peor caso para la validación del software embarcado en instru-

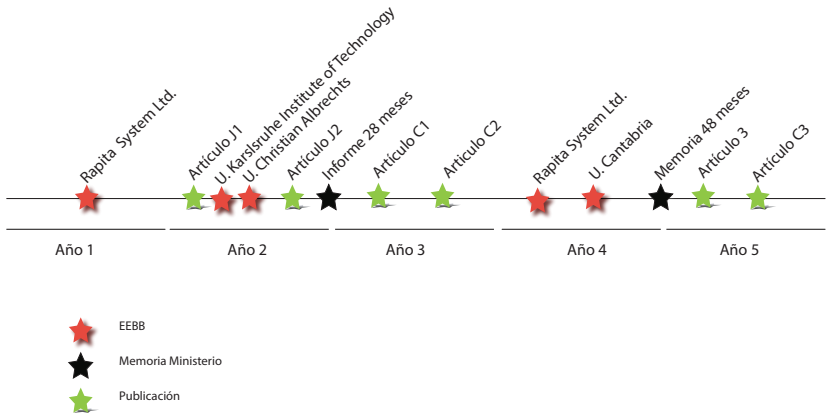


Figura 1.3: Hitos de la Tesis Doctoral. Se incluyen las estancias realizadas en centros de investigación, informes de estado al ministerio y publicaciones.

mentos espaciales, y en concreto, en el caso de estudio de la unidad de control de EPD. Para realizar el análisis se utilizó la herramienta RVS.

Este primer año fue una fase de iniciación en técnicas de WCET con el Dr. Guillem Bernat, experto en la materia. Para cubrir este objetivo se desarrolló un caso de uso para la *Instrument Control Unit* (ICU) de EPD. Éste consistió en el desarrollo de una biblioteca que cubre el protocolo de comunicación *Consultative Committee for Space Data Systems* (CCSDS). De este caso de uso se caracterizaron sus tiempos de ejecución sobre la plataforma Leon. Este estudio permitió conocer los indicadores y las diferentes técnicas de análisis de WCET.

También fue definida la arquitectura multitarea del software de la ICU, utilizado para definir el caso de uso de esta Tesis Doctoral, a partir de las especificaciones. Se usó una notación *Unified Modeling Language* (UML) y la herramienta de modelado basada en componentes EDROOM. El documento *Architecture Design Document* (ADD) se entregó como parte del paquete de documentación del PDR del software de la ICU de EPD, que se superó con éxito.

1.5.2. Definición de modelos transaccionales, definición de la hipótesis (segundo año)

En este segundo año se definió la hipótesis de trabajo de integración de técnicas de desarrollo y análisis de software para la validación automática de EFP de sistemas construidos a partir de componentes. La integración de estas técnicas permitiría la caracterización de la respuesta temporal del sistema software completo, a partir de las medidas obtenidas por separado de cada componente. La integración se realiza sobre el *framework* MICOBS, siguiendo un enfoque MDE, y requiere la definición de una cadena de transformación modelo a modelo.

Otro de los hitos de esta fase es la estancia en el centro tecnológico de Karlsruhe (`KIT`). El objetivo fue la integración de la herramienta PCM con los desarrollos llevados a cabo en 2011 de EPD, además de adaptar la herramienta al paradigma de sistemas de tiempo real. Esto dotó al proceso de una nueva herramienta que permitió realizar el análisis de rendimiento de sistemas software de tiempo real mediante el simulador SimuCom que proporciona la herramienta PCM. De una manera más específica los objetivos fueron:

- Crear una extensión de PCM habilitando para realizar el análisis de rendimiento de sistemas de tiempo real, basados en planificación de prioridades fijas con desalojo y el acceso a recursos compartidos bajo protocolos de herencia de prioridad o de techo de prioridad inmediata.
- Validar la extensión mediante diversas pruebas que permitieron comprobar el correcto funcionamiento del nuevo planificador.
- Realizar el análisis de rendimiento preliminar del software de la unidad de control de EPD. Para ello se usó una estimación de tiempos.

En este año también se realiza una estancia en la Universidad Christian Albrecht, ubicada en Kiel, Alemania. Éste es uno de los grupos que constituyen el equipo de EPD. En ese momento el instrumento acababa de superar el *Preliminary Design Review* (PDR), lo que significaba que habían sido entregados a la Agencia Espacial Europea los modelos preliminares de arquitectura del software de aplicación, el documento consolidado de requisitos (como ya se vio en el primer año), modelos térmicos, etc, del instrumento, cumpliendo con las especificaciones con las que se aprobó EPD como carga útil para Solar Orbiter.

Los trabajos realizados durante la estancia se centraron en las actividades relacionadas con los documentos de requisitos y diseño de la arquitectura del software de aplicación de la unidad ICU. En concreto, se trabajó en la definición de las *interfaces software* con los sensores desarrollados en Kiel, los cuales

son: *SupraThermal Electrons, Ions, and Neutrals* (STEIN), *High Energy Telescope* (HET) y *Electron Proton Telescope* (EPT). Este trabajo se basó en una definición previa de las interfaces en la que ya estaba especificado el nivel físico (UART/LVDS) y el formato básico de trama (Sánchez et al., 2012). Durante esta estancia se definieron con mayor profundidad algunos aspectos relativos al protocolo de comunicación y a la definición del formato del paquete. Con esta información se diseñaron casos de uso de la ICUSW.

A partir de las especificaciones acordadas, se abrió la posibilidad de completar los casos de uso para los modelos de análisis de software desarrollados. Esto permitió definir los eventos asociados al procesamiento de la información de los sensores.

Por último, este año se participa en dos publicaciones: una ponencia en congreso como coautor, en la cual se describe el modelo formal de diseño EDROOM, titulándose «Component-based Engineering and Multi-Platform Deployment for Nanosatellite On-Board Software», y presentada en la conferencia «DASIA»; y un artículo cuyo título es «Instrument Control Unit for the Energetic Particle Detector on-board Solar Orbiter», que presenta las soluciones que han sido adoptadas en el diseño de la ICU de EPD. Las soluciones presentadas en el artículo sobre la descripción de la ICUSW han sido usadas como caso de uso de la presente Tesis Doctoral.

1.5.3. Definición del modelo transaccional de diseño y anotación automática del WCET (tercer año)

En este año se publica el trabajo logrado en los dos años anteriores. Se elaboran ponencias para congresos: el modelo transaccional de análisis y la generación a partir de este del modelo de análisis de planificabilidad MAST. El título del artículo es «Schedulability Analysis of On-board Satellite Software Based on Model-Driven and Compositionality Techniques». En la misma línea, y también como autor, se publica en congreso el artículo «Propuesta de cadena de herramientas para la automatización del análisis de planificabilidad del software del instrumento EPD a bordo del Solar Orbiter». También se participa como coautor en la publicación «HW/SW Co-design of the Instrument Control Unit for the Energetic Particle Detector on-board Solar Orbiter», donde se presentan cuáles son los retos del proceso de desarrollo de la ICU y su diseño.

Este mismo año se define el modelo transaccional de diseño TSDM. Se trata de un modelo de diseño simplificado, el cual es independiente de la tecnología subyacente y permite integrar nuevos modelos de diseño. La integración se facilita, ya que este modelo aplana la jerarquía del comportamiento del modelo de

diseño, lo que implica que sea más cercano a los modelos de diseño que el modelo transaccional de análisis TSAM. Se crea una transformación a este modelo desde el modelo EDROOM. Desde este modelo se habilitan las transformaciones pertinentes al modelo transaccional de análisis TSAM.

Los TSAM son anotados de manera automática con la herramienta RVS. Se crean transformaciones desde los modelos provistos por RVS al modelo TSAM, las cuales fueron realizadas en el contexto de un proyecto fin de carrera (Nilas, 2014). Con este propósito la herramienta RVS se integra en el ciclo de vida en la etapa de pruebas unitarias.

1.5.4. Integración del proceso automático de verificación por evidencias y demultiplexación del modelo transaccional de análisis en situaciones de tiempo real (cuarto año)

Durante la revisión de la hipótesis de la Tesis Doctoral en la estancia breve en el extranjero en Rapita System Ltd. (York, Reino Unido), se incorporan algunas mejoras de la hipótesis de Tesis Doctoral, como es añadir un proceso de verificación. Éste se centra en la verificación de un conjunto de propiedades del sistema, dando lugar al objetivo de verificación del análisis.

La segunda estancia se realizó en la Universidad de Cantabria, en el Grupo de Computadores y Tiempo Real (CTR). Este grupo ha sido el responsable del desarrollo de la herramienta de análisis de planificabilidad MAST (González Harbour et al., 2001). Esta herramienta tiene una serie de restricciones relativas a los operadores de los que se componen las transacciones. Una de estas restricciones afecta al operador *branch*, que está soportado en el análisis por simulación, pero no en el análisis formal de planificación. Como en los modelos transaccionales puede definirse más de una reacción al mismo mensaje —esto depende del estado interno del sistema—, la transformación equivalente al modelo MAST es un operador *branch*, donde cada rama es una de las respuestas.

En esta estancia se abordó el problema, se definieron un conjunto de patrones y se introdujeron, como parte de los modelos transaccionales, las situaciones de tiempo real. Esto es parte del objetivo de la categorización de las reacciones en función de la situación de tiempo real y su estado interno.

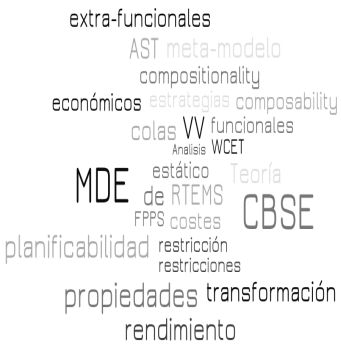
1.5.5. Caso de uso y redacción de la Tesis Doctoral (quinto año)

En 2015 fue publicado el trabajo relacionado con la extensión del análisis de planificabilidad «Integration of a preemptive priority based scheduler in the Palladio Workbench».

Además el proceso propuesto ha sido probado con el software de aplicación de la unidad de control de Solar Orbiter. Esto nos ha permitido comprobar la efectividad del proceso.

Capítulo 2

Estado del arte



There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.

C. A. R. Hoare

El peor enemigo del elefante selvático es el elefante domesticado.

D. Papanikas

Entre las diferentes fases que componen las metodologías de desarrollo de software siempre están presentes los procesos de verificación y validación (VV). Éstos son inherentes a todas las metodologías de diseño de software.

En los sistemas de tiempo real críticos, el proceso de VV cobra una relevancia mayor que en los sistemas de propósito general, ya que, por su propia naturaleza, un fallo puede ser catastrófico (Lions, 1996; Leveson, 1995; Accident, 1996). Esto da lugar a procesos de desarrollo costosos económicamente.

En esta Tesis Doctoral usamos una clara distinción entre verificación y validación (Boehm, 1984). En el contexto de un proceso en forma de V (Storey, 1996):

- **Verificación** es la fase de revisión de que cada paso del proceso de desarrollo es consistente y completo, o dicho de otra manera más simple, **¿estamos construyendo el sistema de una manera correcta?**
- **Validación** es la comprobación de que el sistema de desarrollo reúne los requisitos o, dicho de otra manera, **¿estamos construyendo el producto correcto?**

En este contexto, verificación es la fase inherente de desarrollo que cumple con los elementos marcados por el estándar, garantizando que cada paso del proceso es correcto, mientras que validación es la fase de producción de evidencias que comprueban que el software concuerda con los requisitos, las restricciones de diseño y el análisis de la hipótesis.

El proceso de VV no está desligado de las etapas de diseño e implementación. La complejidad del software es directamente proporcional al coste del proceso de VV. Por eso, en la fase de diseño se buscan propiedades intrínsecas en elementos de construcción o integración de restricciones que faciliten las posteriores fases. También se buscan perspectivas más empíricas, donde los elementos que debemos verificar y validar son menos costosos de observar, mediante bancos de pruebas y técnicas de observación.

Esto significa que el proceso de desarrollo de sistemas de software en tiempo real debe buscar el equilibrio entre el poder expresivo de los modelos utilizados para el diseño y los costes económicos del proceso de VV. Este equilibrio se puede lograr mediante el ajuste de la potencia expresiva usado para el diseño de la aplicación, definiendo restricciones en los modelos. Estas restricciones facilitan la reducción de los costes económicos del proceso de VV e incluso permite la definición de nuevas estrategias que no serían posibles si no se aplicasen estas restricciones.

Las estrategias de VV pueden ser de dos clases. La primera es la relativa al uso de técnicas basadas en la aplicación de métodos formales como, por ejemplo, el uso de *model-checkers* (Holzmann, 1997) y *reasoning frameworks* (Shaw, 1989), todos ellos ligados al mundo académico, el cual busca nuevos caminos que mejoren las actuales propuestas basadas en bancos de pruebas. El segundo tipo está basado en procesos empíricos, partiendo de bancos de pruebas; este tipo de estrategia es más típico de la industria, y está aceptado como el método de facto.

Ahora vamos a definir los conceptos que son usados en esta Tesis Doctoral para los términos de sus *Functional Properties* (FP), y sus EFP.

- **FP** son aquellas propiedades que definen el sistema en función de la relación entre sus salidas y sus entradas.
- **EFP** son aquellas propiedades que definen cómo el sistema proporciona las salidas a partir de las entradas, y se centran en aspectos tales como el tiempo de ejecución en el peor caso, el tiempo medio de respuesta, el espacio de pila ocupado por las tareas, etc.

Esta categorización es ampliamente usada por la comunidad de sistemas de tiempo real. Por ejemplo, (Chung y Prado Leite, 2009) fundamentan esta categorización con el objetivo de extenderla a la reutilización de software. Podría decirse que la especificación funcional de un elemento está libre de sus connotaciones extrafuncionales cuando conocemos la relación del impacto de las propiedades no funcionales, por lo que se podría reutilizar en cualquier otro sistema.

Sea cual sea el tipo de estrategias que hemos visto anteriormente, éstas tienen el mismo objetivo, que es la caracterización de las propiedades del software, tanto funcionales como extrafuncionales. Éstas pueden ser caracterizadas o bien por las mismas propiedades intrínsecas de los axiomas de construcción (proceso deductivo) o porque han sido observadas en un banco de pruebas (procesos inductivos). Incluso el uso de estrategias mixtas puede ser interesante. Por ejemplo, la empresa SymTa Vision (Racu et al., 2006) usa un método híbrido para la caracterización del WCET, usando modelos de memoria caché y métodos basados en trazas de ejecución para realizar el análisis (White et al., 1997).

2.1. CBSE

En los últimos años, dentro del paradigma de *Component-Based Software Engineering* (CBSE) han surgido nuevas propiedades intrínsecas. Estas propiedades pivotan alrededor de la definición de componentes, los cuales permiten aumentar la cohesión y reducir el acoplamiento de los elementos que componen el sistema (Crnkovic et al., 2009). Parte de estas investigaciones están relacionadas con el análisis de EFP de los componentes y la introducción de la metodología de desarrollo de corrección-por-construcción (*correctness by construction*) (López et al., 2008; Stafford y Wallnau, 2003; Lau y Wang, 2007; Hissam et al., 2003; Crnkovic et al., 2005; Crnkovi, 2003). Dentro de ésta última se definen dos propiedades asociadas a los modelos de componentes como

son las propiedades de *composability* y de *compositionality* (Mazzini et al., 2008; Matic, 2008; Gössler y Sifakis, 2005). Para que un sistema cumpla el principio de *composability*, los componentes que lo forman no tienen que ver alteradas sus propiedades intrínsecas por la acción de otros componentes. Asimismo, el principio de *compositionality* se cumple cuando las propiedades del sistema se pueden expresar como la composición de las propiedades de sus partes, esto es, de sus componentes.

Otras restricciones formales son relativas a la descripción de la comunicación entre componentes. Como, por ejemplo, la eliminación de llamadas cíclicas entre componentes o la definición de contratos software, pudiendo éstos ser relativos a propiedades funcionales o no funcionales (Hawkins, 2006). En el trabajo propuesto por (Mangeruca et al., 2013), estos contratos definen qué elementos son provistos o requeridos por el sistema y se comprueba si son satisfechos cuando el sistema es compuesto. Otras propuestas como, por ejemplo, (De Alfaro y Henzinger, 2001; Cancila, 2008), sólo permiten usar elementos en el sistema que cubren los requisitos de las interfaces, de tal manera que sólo se deja conectar aquellos elementos que satisfagan todas las propiedades especificadas. Estas propuestas son interesantes, ya que aumentan la calidad del sistema resultante del proceso.

Las técnicas basadas en CBSE pueden ser clasificadas como abstractas o constructivas (Carloni et al., 2015). Los enfoques abstractos a menudo se basan en álgebras abstractas y modelan sistemas muy generales (Bruni et al., 2007), los cuales utilizan, por ejemplo teoría de grafos, para modelar la estructura arquitectónica de un sistema o la adición de nuevas maneras de expresión de máquinas de estados que permitan dotarlas de mejoras relativas al poder de expresión (Harel, 1987). Ejemplos típicos son los estándares de la *Object Management Group* (OMG) (OMG, 2015), como es el caso de *Unified Modeling Language 2* (UML-2) (Miles y Hamilton, 2006), *Systems Modeling Language* (Sysml) (Friedenthal et al., 2008) y *Real-Time Object-Oriented Modeling* (ROOM) (Selic et al., 1994). Por otro lado, los modelos de componentes constructivos tienen como objetivo la generación de sistemas basados en componentes. Definen estrategias y elementos de sistema concretos para un área industrial específica, por ejemplo en sistemas empotrados existen herramientas como EDROOM (Polo et al., 2001) y *Modeling and Analysis of Real-Time and Embedded Systems* (MARTE) (Object Management Group, 2009), y algunas implementaciones de Sysml como son el caso de (SCADE, 2015). Siempre los modelos constructivos se basan en los modelos abstractos.

2.2. Procesos de verificación y validación

Tradicionalmente la industria usa procesos de VV basados en bancos de pruebas. Estos procesos validan los requisitos marcados y los verifican mediante la definición de pruebas a partir de los mismos. En los últimos tiempos se ha visto un incremento de la complejidad del software. Esto implica que las técnicas de análisis y de VV del software se estén quedando atrás debido al aumento del coste (McDermid y Pumfrey, 2001). Los sistemas empotrados críticos cada vez tienen mayor peso en la industria y el número de líneas por sistema está creciendo exponencialmente. Una estimación del coste del proceso de VV usando pruebas es difícil de determinar (McDermid y Kelly, 2006). Depende de varios factores, como el sector industrial, el estándar de facto, etc. Cada sector tiene diferentes estándares, por ejemplo, automoción, *Road Vehicles – Functional Safety 2626* (ISO-2626), aviación, *Software Considerations in Airborne Systems and Equipment Certification 178-C* (DO-178-C) y *European Cooperation for Space Standardization, Space Engineering, Software* (ECSS-E-ST-40C). La diferencia entre zonas económicas también afecta, como es el caso de Europa, donde es más común el uso de técnicas de análisis más formales, estando en contraposición, por ejemplo, con Estados Unidos, donde es más común el uso de técnicas más experimentales (Feldt et al., 2010). Todo esto implica que una estimación del coste de la fase de VV es difícil de cuantificar. Aunque podemos afirmar que su coste no será menor del 50 % del presupuesto del proyecto (McDermid y Pumfrey, 2001).

El coste es elevado y la complejidad es creciente. La pregunta que se hace la comunidad investigadora es: ¿cómo podemos optimizar el proceso de VV? En el caso concreto de esta Tesis Doctoral, ¿cómo podemos optimizarlo y respetar los procesos de facto de la industria? Hay líneas de investigación que tienen como objetivo poder reducir el porcentaje de este coste económico.

Existen trabajos recientes que eliminan la definición de los bancos de pruebas parcialmente, como la verificación de restricciones, o totalmente, como el análisis estático de código (Sauser et al., 2009). También hay líneas de investigación de generación automática de bancos de prueba (Niemann y Haubelt, 2006) y generación de infraestructuras de bancos de pruebas unitarias (Ltd., 2015; VectorCast, 2015). A día de hoy los procesos industriales requieren el proceso VV basado en bancos de pruebas, por lo que estrategias estáticas, cuya ventaja es el no uso de estos bancos, están fuera del contexto industrial actual.

Una propuesta de verificación interesante es la verificación por asunciones (Ltd., 2014). Al contrario que las propuestas estáticas, está basada en la comprobación de determinados hechos, los cuales son observados en el software de

despliegue. Para ello, mediante técnicas de instrumentación, observa diferentes parámetros, los cuales serán verificados posteriormente mediante restricciones especificadas a través de Automata Finito Temporal (AFT). De esta manera se comprueba en los bancos de pruebas que el sistema cumple con diferentes requisitos, los cuales pueden ser de naturaleza extrafuncional o funcional. La utilización de este tipo de técnicas permiten verificar que los modelos de análisis utilizados son válidos.

2.3. MDE

Otra forma de reducir el coste económico consiste en el uso de mecanismos en forma de herramientas que la industria ha desarrollado, las cuales permiten automatizar fases o elementos del proceso software de acuerdo con los estándares. Éstas permiten automatizar parte del proceso de VV, como por ejemplo: análisis de planificabilidad, análisis de pila, análisis de dependencias, etc. Otras herramientas permiten automatizar la verificación de métricas, como son WCET (Wilhelm et al., 2003), *performance* (Rathfelder et al., 2013), caracterización QoS (Cisco, 2003) o cobertura (Marick, 1999). En los procesos software estas herramientas pueden ser usadas conjuntamente para cubrir requisitos de los estándares. Estas herramientas normalmente se articulan formando cadenas (Perrotin et al., 2012). La vertebración se realiza en torno a la definición del proceso de desarrollo software, el cual, como hemos visto, es estratégico para la empresa, sector, nación, etc.

La definición de procesos software mediante técnicas de Modelo MDE (Kent, 2002b) permite integrar los diferentes modelos del proceso software de una manera coherente, eliminando las discontinuidades entre modelos y vertebrando el proceso software mediante la definición de los mismos y sus transformaciones. Esta técnica define cada uno de los elementos que conforman el proceso como modelo, donde pueden definirse diferentes niveles de abstracción o, incluso más interesante, diferentes perspectivas del sistema, como son las basadas en las acciones que deben llevarse a cabo por diferentes roles (Reussner et al., 2007). MDE define el concepto metamodelo, el cual especifica los elementos y el poder expresivo del mismo. Éste puede ser restringido semánticamente. De hecho, podrían ser expresadas restricciones relativas a la recursividad, anidamiento de bucles y complejidad ciclomática. Cuando procesos complejos son descritos se necesitan definir distintos metamodelos que permiten describir distintas vistas de los mismos. Para evitar discontinuidades semánticas MDE soporta la definición automática de transformaciones entre modelos. La OMG definió pa-

radigmas en este sentido, como QVTO (OMG, 2015). Estas transformaciones eliminan las discontinuidades entre modelos, permitiendo definir procesos, estrategias y mecanismos de una manera coherente.

Parte de las restricciones propuestas en la fase de diseño, como las relativas a la recursividad, anidamiento de bucles y complejidad ciclomática, permiten generar modelos de análisis con una serie de propiedades. Unas de las características resultantes es la habilitación de los principios de *composability* y *compositionality*. Ambas propiedades permitirían realizar caracterizaciones parciales del sistema, y una vez hechas, podremos inferir las mismas propiedades a nivel de sistema. Por ejemplo, (Gössler y Sifakis, 2005) proponen la caracterización de las propiedades funcionales de un sistema, y a partir de éstas se infieren las propiedades del sistema. Para esto se propuso una serie de restricciones, como la eliminación de posibles efectos, como los *deadlocks*. En este trabajo estamos interesados en la eliminación de estos *deadlocks*, de tal manera que se facilita la inferencia de las propiedades que deben ser provistas para realizar un análisis de planificabilidad y de rendimiento.

2.4. Análisis de planificación y rendimiento

Para aplicar una serie de restricciones válidas a los modelos de diseño, antes hay que entender las restricciones de los modelos de análisis en sistemas de tiempo real. En especial nos centraremos en los análisis de planificación y de rendimiento. Estos análisis permiten caracterizar el uso de los recursos que forma parte de la plataforma, el análisis de las restricciones temporales y los tiempos de respuesta a los eventos externos. Este tipo de análisis es necesario debido a que los sistemas de tiempo real no pueden corroborar ciertas propiedades —por ejemplo, los tiempos de respuesta— con la asignación dinámica de recursos, debido al indeterminismo que se introducen en los tiempos de ejecución. Por este motivo se debe fijar de manera estática el ratio de procesamiento de la CPU, el tamaño de memoria y el tiempo necesario para gestionar las demandas de entrada/salida. De esta forma es posible cuantificar y verificar sus requisitos no funcionales.

Las técnicas de análisis de planificación y rendimiento nos permiten identificar aquellos elementos que no cumplen con los requisitos de tiempo de respuesta y tienen que ser optimizados. Cuando un producto está en fase de integración, y las optimizaciones son relativas a la arquitectura del software, da lugar a sobrecostes, poniendo en peligro el proyecto (McDermid y Kelly, 2006). Por este motivo, es de especial interés que los análisis de planificabilidad y de rendimien-

to estén presentes a lo largo de las diferentes fases del ciclo de vida, y no sólo acotados a una última fase de pruebas de integración. De hecho, las únicas optimizaciones que deberían ser realizadas en las pruebas de integración no deberían tener un alcance más lejano a un módulo software, y su comportamiento a nivel arquitectónico no debería verse alterado (Laplante, 1992).

Las medidas realizadas en la fase de pruebas sobre la plataforma de despliegue son las más fructíferas, ya que resuelven las incógnitas relativas a la plataforma y a la implementación detallada del software. Sin embargo, es durante el diseño cuando queremos definir aspectos de la arquitectura que afectan a los tiempos de respuesta, por ejemplo, que los *deadlines* se cumplan. En el caso de que la plataforma no esté disponible es posible inferir las propiedades del software mediante estimaciones basadas en la experiencia en el desarrollo de software similar o utilizar la simulación de aquellos elementos que son críticos. Estas estimaciones se pueden combinar con análisis teóricos, como *Rate Monotonic Analysis* (RMS), que nos permiten determinar el tiempo de respuesta.

Todas las técnicas de análisis software aquí presentadas están limitadas por la complejidad computacional. Hasta que lleguen otros modelos computacionales con relaciones distintas de longitud del problema/tiempo de respuesta, tenemos que trabajar con las limitaciones de las máquinas deterministas (máquinas de Turing). En el caso del análisis de planificabilidad y rendimiento, el problema que queremos resolver es la asignación de recursos en un sistema multitarea. Este problema está relacionado con algoritmos de satisfacción booleana o N-sat. Aunque los problemas 2-Sat tienen una complejidad polinómica, N-Sat —resolución de los problemas más interesantes— es no polinómico (NP) completo. La asignación de recursos, en el caso que nos ocupa, incluye elementos como multicores, multiprocesadores y algunos elementos relacionados con sincronización (*mutex*), es un problema NP completo. Sin embargo, si se aplica un conjunto de restricciones como las anteriormente citadas, el problema puede pasar de NP completo a NP, de tal manera que pueden usarse algoritmos heurísticos que permiten una asignación de recursos que cumple con las restricciones, aunque esta asignación no sea la óptima. Por ejemplo, una de estas estrategias es el uso de un único procesador usando asignaciones estáticas.

Los tiempos de ejecución que tienen que ser caracterizados están relacionados con el tiempo relativo a las interrupciones/corrutinas, con el software de aplicación reactivo. Dependiendo de la granularidad de la técnica de extracción de medición de tiempos de ejecución, esta información podrá ser desgranada en menor o mayor granularidad de los tiempos de respuesta. Por ejemplo, medir el tiempo de latencia hardware de una interrupción es difícil debido a que es del orden de nanosegundos.

Hay un gran abanico de herramientas de cálculo WCET que se basan en diferentes estrategias de caracterización y con diferentes grados de industrialización —algunas herramientas son industriales y otras, de carácter académico o demostradores relacionados a proyectos de investigación— Un estudio pormenorizado de estas herramientas se presentó en (Wilhelm et al., 2008). Entre las diferentes estrategias nos parecen de gran relevancia las técnicas híbridas, las cuales se basan en medidas *in situ* combinadas con análisis estático. Estas medidas son usadas posteriormente para realizar un análisis WCET. Una de estas herramientas es RVS (Ltd., 2007). Esta herramienta instrumenta el código fuente con elementos inequívocos, de tal manera que tras la ejecución del sistema estos elementos son capturados con alguna clase de infraestructura, como, por ejemplo, un analizador lógico. La herramienta realiza un análisis de WCET, basado en el árbol *Abstract Syntax Tree* (AST) del código fuente. RVS presenta varios perfiles de tiempo de ejecución, entre ellos algunos de carácter analítico, como es el caso del WCET, y otros de observación, como son el tiempo de ejecución máximo y el tiempo de ejecución mínimo. Entre sus debilidades se encuentra que los tiempos de las piezas software son medidos. Esta cualidad implica que no son deducidos por parte de ningún modelo detallado, como sí sucede en otros trabajos (Gustafsson et al., 2003; Jin y Li, 2008; Lundqvist, 2002; Sandberg et al., 2006). Esta debilidad comparativa con análisis estáticos se reduce, ya que estos últimos, tienen como contrapartida errores inherentes a todo proceso software (Black y Shen, 1998), donde para poder corregirlos se deberá realizar un conjunto de bancos de pruebas. Estos bancos de pruebas se denominan bancos de prueba de calibración (*calibration test*). Otras de las limitaciones de RVS son las relativas al uso de una estructura como los árboles para el análisis de WCET. Aunque son fáciles de generar a partir del AST, estos árboles realizan una serie de sobreestimaciones, como es el caso de los bucles triangulares, y otras deficiencias (Li et al., 2014), elemento que podría de una manera ideal subsanarse con propuestas de uso de grafos, como es el caso presentado por (Betts y Bernat, 2006).

Creemos que la significatividad de los tiempos de ejecución, los cuales formarán parte del análisis de WCET, dependerá del estándar aplicable al proceso de desarrollo del software. El tipo de estrategias que puede usarse sería relativo a la cobertura del código. Por ejemplo, en el caso de un sistema muy crítico, se deberá comprobar, mediante pruebas, todas las combinaciones de posibles caminos y todas las combinaciones en elementos de decisión (MC/DC) (ECSS Secretariat, 2009), esto implica estresar cada elemento del software.

El análisis de planificabilidad nos permite corroborar de una manera analítica con los valores del WCET si las restricciones temporales a nivel de sistemas

son satisfechas. Hay varias técnicas (Sha et al., 2004), pero las que están extensamente usadas en la industria son las estrategias basadas en un *Fixed Priority Preemptive Scheduling* (FPPS). Esta conclusión la hemos extraído de un estudio de mercado del año 2014 realizado por la empresa UBM Tech (Study, 2014), donde en la cima de la clasificación de los sistemas operativos de tiempo real que fueron más utilizados por la industria se encuentran: *Free Real-Time Operation System* (FreeRTOS), *Controller Operation System - II* (UC/OS-II) o *VxWorks Real Time Operating System* (VxWorks), los cuales se basan en un planificador FPPS. Otra evidencia a favor de su importancia, esta vez en el contexto de esta Tesis Doctoral, es que el software embarcado en satélite, el sistema operativo de tiempo real recomendado por la ESA para sus misiones es *Real-Time Executive for Multiprocessor Systems* (RTEMS), también implementa un planificador FPPS. Por último, vale la pena mencionar que la asignación de prioridades fijas en las tareas es la base del algoritmo de planificación RMS, que a su vez se utiliza ampliamente en el diseño de sistemas críticos de tiempo real para ofrecer garantías de plazos de respuesta.

Con la información del análisis de planificabilidad se pueden identificar los elementos que no cumplen con las restricciones de tiempos de respuesta. Por lo que se pasará a realizar un conjunto de optimizaciones en el código fuente. En este punto creemos que algunas recomendaciones realizadas por Laplante (1992) merecen ser al menos enumeradas:

- **Escalas aritméticas**, eliminar el uso de operadores de coma flotante por tablas con factores precalculados. Con esta operación cambiamos el tiempo de cálculo de operaciones de coma flotante por un acceso a memoria donde se extraerá el valor precalculado.

- **Medidas del momento angular**, similar a la escala aritmética, reduce el uso de elementos en coma flotante, en este caso relativos a momentos angulares.

- **Tablas del tipo Look-up**, similar a las anteriores propuestas, precalcula porciones de una ecuación, por ejemplo, en el caso del cálculo *Cyclic Redundancy Check* (CRC) es común su uso.

- **Funciones intrínsecas**, siempre que se pueda, en piezas de código muy sensibles se deberán usar llamadas a funciones de este tipo (en C se denomina funciones INLINE o macros) en vez de las ordinarias. Nos permite ahorrar la sobrecarga de llamadas a funciones.

- **Eliminación de invariantes en bucles**, aquellos elementos que no dependen de ningún parámetro que puede cambiar con cada iteración. La mayoría de optimizadores de compiladores suelen mover estas instrucciones fuera del bucle de una manera automática. En algunos casos, el uso de optimizaciones por parte de los compiladores no está permitido. Esto implica que deberán tenerse en cuenta como optimización de código manual.

- **Memorias cachés y registros**, este uso no se refiere a la optimización de tiempo medio, como es el caso de las memorias caché en sistemas de propósito general, sino memorias que almacenan valores usados con mucha asiduidad. Por ejemplo, direcciones de memoria que son especialmente usadas pueden almacenarse en un registro del microprocesador o parte de la memoria caché, especificando al compilador que éste quedará congelado.

- **Eliminación de código muerto**, si se eliminan caminos de ejecución, se reducirá la complejidad, y puede ser que se elimine un camino patológico. En caso de uso pueden ser utilizadas directivas al compilador para incluir código dependiendo de parámetros de configuración.

- **Optimización de los caminos de ejecución**, puede realizarse mediante la reducción de puntos de decisión.

- **Desenrollado de bucles**. Los bucles pueden ser desenrollados por un determinado factor, lo que nos permitirá, dependiendo de este factor, ahorrarnos las instrucciones relativas a la condición de iteración del bucle.

Hay más técnicas de optimización, incluso se propuso que este tipo de optimizaciones relativas a peores casos se integrasen en las fases de *Back-End* de los compiladores de tiempo real (Plazar et al., 2013).

Finalmente vamos a ver las ventajas de un análisis de rendimiento en sistemas de tiempo real. En el análisis del sistema es necesario tener en cuenta que no todos los eventos de un sistema de tiempo real responden a un patrón de tipo periódico, algunos elementos son esporádicos. Estos elementos pueden ajustarse a un modelo *rate-monotonic* ajustando su *inter-arrival time* a un periodo constante, aunque en algunos casos, cuando el caso es muy patológico, esto da lugar a niveles de rendimiento pésimo (Liu, 2000). Por este motivo, los análisis estadísticos, como es el caso del análisis de rendimiento mediante simulación, son especialmente interesantes, ya que permiten identificar qué elementos son los que amplían el coeficiente de dispersión de los tiempos de ejecución (Laplante,

1992). Entre este tipo de análisis estadísticos destacan los basados en teoría de colas (Brémaud, 1999).

La teoría de colas nos permite modelar los eventos del sistema y el tiempo de respuesta a los eventos en función de distribuciones de probabilidades. Estamos especialmente interesados en el modelo M/M/1, donde la tupla está formada por:

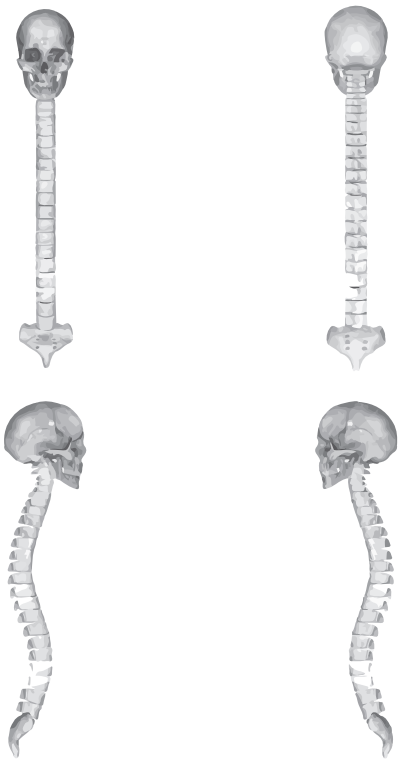
- M, es la función exponencial de Poisson del tiempo medio de llegada del evento.
- M, es la función de distribución exponencial de Poisson del tiempo para satisfacer la demanda.
- 1, es el número de unidades de CPU para procesar las demandas.

Como restricciones para que el análisis se lleve a cabo, se asume que la cadencia de producción de eventos es menor que el de consumo, esto es: las funciones de distribución $\frac{1}{\lambda}$ y tiempo $\frac{1}{\mu}$, siendo $\frac{1}{\lambda} < \frac{1}{\mu}$, que la longitud de la cola es infinita, y que el promedio de llegada de clientes es menor al promedio al que son atendidos por los servidores. Con este modelo podemos calcular el tiempo medio de llegada, el tamaño máximo de los *buffers*, y el tiempo de respuesta de los eventos.

Cuando el software que se pretende analizar es complejo, generar modelos basados en teoría de colas con cierto nivel de detalle es una tarea propensa a errores. Esta motivación es el punto de partida de varias propuestas; como por ejemplo, PCM (Becker et al., 2009). Palladio propone describir el modelo a analizar usando modelos similares a los usados en la etapa de diseño. Estos modelos de diseño se transforman en complejos modelos de análisis basados en teorías de colas. Finalmente, y tras ejecutar herramientas de análisis usando los modelos generados, los resultados son anotados en los modelos de diseño.

Capítulo 3

Modelo transaccional



If you change the way you look at things, the things you look at change.

W. Dyer

Medir el progreso del desarrollo de software por líneas de código es como medir el progreso de la construcción de un avión por su peso.

B. Gates

¿Los índices de los arrays deberían comenzar en 0 o en 1? Mi propuesta neutral de usar 0.5 fue rechazada, en mi opinión, sin la debida consideración

S. Kelly-Bootle

3.1. Introducción

En este capítulo abordamos la descripción de los modelos denominados transaccionales, llamados así porque permiten hacer explícitas las transacciones que implementa el sistema. Estos modelos son una parte fundamental de la hipótesis planteada en esta Tesis Doctoral, siendo los encargados de habilitar la generación de modelos de planificación, rendimiento y verificación a partir del modelo de diseño. También se plantea un conjunto de elementos que tienen que estar presentes para que un modelo de diseño sea compatible con el proceso.

Los modelos transaccionales propuestos han sido integrados en el *framework* MICOBS. Este *framework* nos da soporte durante las definiciones de transformaciones y la categorización de las EFP que deben ser anotadas para realizar los análisis de planificabilidad y rendimiento. Por un lado nos facilita la implementación del proceso, ya que hay elementos del propio *framework* que son usados como soporte evitando así tener que añadirlos, y por otro lado añade al paradigma MDE el concepto de plataforma, permitiendo independizar plataforma y software, y actuando como vector para obtener los productos del proceso (los modelos de análisis y rendimiento) a partir de la plataforma seleccionada.

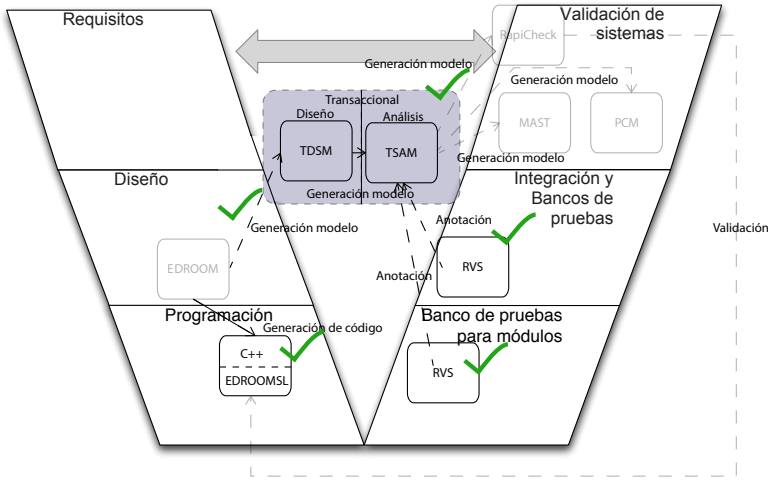


Figura 3.1: Contextualización de la funcionalidad de los paquetes de modelos transaccionales en el proceso mostrado en la Tesis Doctoral.

La funcionalidad principal del modelo transaccional es la de actuar como pivote, siendo en esta Tesis Doctoral el elemento vertebrador del proceso. De esta manera cumple la función de interfaz entre la descripción del modelo formal de diseño y los modelos de análisis. La transformación hacia el modelo transaccional se ha dividido en dos fases. La primera fase consiste en obtener un Modelo de Diseño Simplificado, que denominamos TSDM, mientras que la segunda permite obtener el modelo transaccional orientado al análisis que denominamos TSAM. Este enfoque facilita la cadena completa de transformaciones ya que el TSDM es cercano al origen de la transformación (Modelo Formal de Diseño), mientras que el TSAM lo es al destino final, es decir, a los modelos de Análisis. Esto reduce las complejidad de las transformaciones desde distintos modelos formales de diseño habilitando transiciones desde modelos basados en diagrama de secuencia (Grønmo y Møller-Pedersen, 2010) o de actividad (Börger et al., 2000), y hacia nuevos modelos de análisis. De esta manera no hay que definir una transformación completa desde un nuevo modelo formal de diseño a cada uno de los modelos de análisis. Sólo se definirán nuevas transformaciones a su relativo cercano.

El encaje en el proceso de los modelos transaccionales se muestra en la imagen 3.1. En ella vemos que, partiendo del modelo de diseño EDROOM, se define una transformación al modelo TSDM, desde éste se define una transformación al modelo TSAM y desde el modelo TSAM se describe un número de transformaciones que dependen de los modelos de análisis que son definidos en el proceso.

Los costes se reducen gracias a la automatización de los elementos que tienen que ser generados de acuerdo a los estándares típicos de cada industria, como el ECSS-E-ST-40C, aplicable al software embarcado en satélite.

Los modelos transaccionales son definidos para la anotación de EFP. Esto significa que el modelo de diseño no es anotado. El modelo anotado es un modelo equivalente orientado al análisis que permite especificar las propiedades en función de la plataforma. Esta cualidad nos permite desvincular la plataforma de sus propiedades.

Gracias a la topología de modelos utilizados el número de transformaciones se reduce. Si se quiere añadir un nuevo modelo de análisis o de diseño, sólo deberá definirse una transformación desde o hacia el modelo que le corresponda. Por ejemplo, si se quiere añadir un modelo formal de diseño se deberá generar una nueva transformación hacia el modelo TSDM, si en cambio se quiere añadir un nuevo modelo de análisis se deberá añadir una nueva transformación desde el modelo TSAM a dicho modelo.

En resumen, hemos definido dos modelos, uno de ellos es el relativo a una descripción cercana a los modelos de diseño y el otro es cercano a la descripción de los modelos de análisis. Ello permitirá reducir los costes como hemos visto en los párrafos anteriores.

- Denominaremos al paquete de modelo transaccional de diseño simplificado como TSDM. El TSDM nos permite definir un modelo intermedio e independiente de la tecnología de componentes subyacente, de tal manera que la transformación desde un modelo formal de diseño a éste es cercana.
- El paquete que describe el modelo transaccional de análisis se denomina TSAM. El TSAM permite integrar con facilidad diversas herramientas de análisis, ya que este modelo contiene información anotada de las EFP y una descripción reactiva del sistema en secuencias de acciones lineales similar a las cadenas de Markov (Brémaud, 1999).

Para poder ilustrar mejor los elementos que son descritos en esta sección en la imagen 3.2 mostramos los modelos y sus transformaciones. De esta manera el capítulo queda organizado de la siguiente forma: la sección 3.2 trata sobre qué requisitos tienen que tener los modelos de diseño para ser integrados en el proceso, en la sección 3.3 se describen las modificaciones realizadas al modelo de diseño EDROOM para que la información de análisis sea completa y un resumen en la sección 3.4.3 de su proyección al modelo TSDM, en la sección 3.5 se detallan los modelos TSAM que actúan como lenguaje intermedio de análisis. Además, en esta sección se describe cómo son anotados los modelos TSAM. En la sección 3.5.6 se presenta un resumen de la transformación automática desde el modelo TSDM al modelo TSAM.

3.2. Restricciones del modelo de diseño

Un modelo de diseño para ser integrado en el proceso tiene que cumplir una serie de patrones y restricciones de diseño. Las restricciones que proponemos están relacionadas con la propuesta de desarrollo conocido como *correctness by construction* (Chapman, 2006). En nuestro caso en particular, definimos que un sistema es correcto si su análisis de planificabilidad es factible, y además, demuestra que se cumplen las restricciones temporales. Lo que significa que el objetivo de estas restricciones es que el modelo de diseño tenga la propiedad de ser analizable. En el caso particular de esta Tesis Doctoral queremos que los modelos de diseño basados en componentes sean analizables por composición.

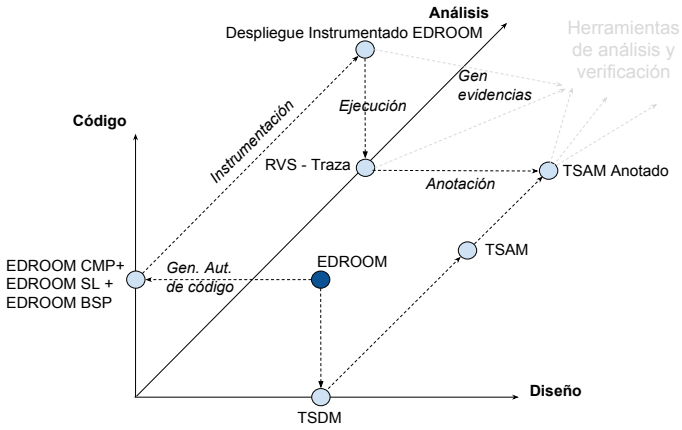


Figura 3.2: Se muestra la transformación desde el modelo de diseño al modelo transaccional y su anotación. Desde el modelo de diseño EDROOM se genera el código a desplegar sobre la plataforma y el modelo TSDM, que a su vez se transforma en el modelo TSAM. Este modelo servirá para generar a partir de él distintos modelos de análisis requeridos por el estándar ECSS-E-ST-40C.

Hemos dividido las restricciones en dos grupos por claridad en su exposición. Por un lado tenemos las restricciones a nivel de sistema y por el otro, restricciones a nivel de componente.

- Las restricciones a nivel de sistema, donde seguimos una estrategia de caja negra y definimos elementos de planificación del sistema, eventos y comunicación de componentes.

- Las restricciones a nivel de componente, donde se sigue una estrategia de caja blanca, y nos centramos en qué elementos tienen que estar definidos de una manera explícita en las reacciones.

3.2.1. Restricciones a nivel de sistema

La restricción propuesta más relevante es la utilización de un patrón de planificación basado en prioridades fijas con desalojo, junto con el uso de protocolos herencia de prioridad o techo de prioridad inmediato para evitar fenómenos de inversión de prioridad. Se recomienda que la gestión de las interrupciones se realice mediante una estrategia de `Top-half` y `Bottom-half` , ya que ajusta los tiempos de respuesta asociados a las interrupciones.

Además del patrón citado, el conjunto completo de restricciones a nivel sistema es el siguiente:

- El sistema debe estar formado por un conjunto de componentes con prioridad fija que se ejecutan sobre un único procesador.
- La prioridad de un componente sólo deberá poder ser modificada de una manera temporal cuando se accede a un recurso compartido. El cambio de prioridad será realizado en función de los protocolos de herencia (Sha et al., 1990) o de techo de prioridad (Goodenough y Sha, 1988).
- Los recursos compartidos no deben estar anidados. Esto permite eliminar la posibilidad de fenómenos relacionados con los abrazos mortales.
- Los componentes deberán comunicarse entre sí a través del paso de mensajes que serán almacenados en colas *First In First Out* (FIFO) hasta ser atendidos. Esto permite el desacoplamiento del comportamiento de los componentes, lo que implica fomentar en el modelo de diseño propiedades *composability* y *compositionality*.
- El *System Run-time* debe proporciona un planificador de prioridades fijas con desalojo, de tal manera que se determina en cada momento qué componente debe ejecutarse.
- El *System Run-time* debe proporcionar manejadores de eventos de excepción y de interrupción. Éstos pueden estar asociados a controladores de entrada/salida y temporizadores.
- El *System Run-time* debe ser configurado para que los manejadores de interrupciones conviertan los eventos en mensajes, enviando éstos a los componentes que están suscritos a eventos. De esta forma, los componentes pueden suscribirse a servicios de temporización e interrupción.

- El *System Run-time* puede personalizar la gestión de las interrupciones generadas mediante mecanismos de `Bottom-half`. Este requisito no es obligatorio, ya que es sólo una medida de eficiencia. Esta eficiencia se ve plasmada en análisis relativos a tiempos de respuesta. ¿Cómo funciona este mecanismo? Añadiendo sólo código correspondiente a la captura de la excepción en el elemento `Top-half`, y dejando la gestión del evento en el elemento `Bottom-half`. De esta forma el `Bottom-half` gestionará la parte más pesada, por ejemplo, conversión de evento a mensaje, de esta manera el sistema tendrá deshabilitadas las interrupciones menos tiempo, disminuyendo la latencia global de las interrupciones menos prioritarias.
- Los eventos a los que responde el sistema deben seguir unos patrones de activación acotados. Los patrones son los siguientes:
  - Periódico.
  - Esporádico.
  - Ráfaga.

3.2.2. Restricciones a nivel componente

Introducimos un conjunto de restricciones de diseño sólo aplicadas a los componentes. En nuestro caso el uso de los patrones de componentes que proponemos, junto con las anteriores restricciones, garantiza que el sistema sea analizable y, además, permite la construcción de modelos de análisis a través de transformaciones modelo a modelo. Estos son los tipos de componentes que proponemos:

- **Componente proactivo**, puede enviar mensajes asíncronos, suscribirse a eventos, iniciar una comunicación síncrona y acceder a recursos compartidos. Por contra, no puede responder a una comunicación síncrona.
- **Componente reactivo**, puede enviar y recibir mensajes asíncronos de manera idéntica a los componentes proactivos. En la comunicación síncrona, sin embargo, este tipo de componentes puede también responder a los componentes del tipo proactivo. Además, puede estar suscrito a eventos y acceder a recursos compartidos.
- **Recurso compartido**, sólo puede responder a mensajes síncronos. Esto permite predecir la respuesta e impide anidamientos de recursos compartidos. Estos elementos no pueden estar suscritos a eventos.

Los componentes, además de pertenecer a uno de los tres tipos citados, deben tener las siguientes propiedades:

- La reacción de un mensaje por parte de cualquier tipo de componente debe tener una respuesta finita y predecible, lo que implica que no está permitida la recursividad.
- La respuesta tiene que ser definida como una secuencia de acciones básicas. La naturaleza de estas acciones tiene que expresarse de una manera explícita, las cuales pueden disparar tanto el envío de mensajes predeterminados, como la invocación de funciones. En este sentido se propone el siguiente conjunto de elementos que pueden formar parte de una acción:
  - **MsgDatahandler**, acción que gestiona la información adjunta a un mensaje.
  - **Action**, acción que desempeña una función que no supera el alcance del componente.
  - **Send**, acción que envía un mensaje de naturaleza asíncrona.
  - **Invoke**, acción que envía un mensaje de naturaleza síncrona.
  - **Reply**, acción que define la reacción de un envío de un mensaje síncrono.
- El orden de ejecución de las acciones básicas también tiene que ser explícito. La priorización es la siguiente: `MsgDataHandler` → `Action` → `Send` → `Invoke`.
- En el caso de la respuesta a un mensaje síncrono, debe incluirse una única acción de tipo `reply` intercalada en la secuencia de acciones básicas.
- Las funciones que un componente puede invocar deben ser previamente y explícitamente declaradas. Por ejemplo, las llamadas a *bibliotecas* externas tienen que formar parte del modelo de diseño y no encontrarse de una manera implícita.

3.3. Adaptación del modelo de diseño EDROOM

EDROOM está compuesto por diferentes modelos:

- Modelo de definición de comportamiento.

- Modelo de estructura de componentes.
- Modelo de definición de protocolos.

EDROOM permite la descripción de un sistema de acuerdo con las restricciones a nivel de sistema y de componente que hemos planteado en la sección anterior (véase 3.2.1).

En el aspecto de las restricciones a nivel de sistema EDROOM cumple con todas las restricciones. De hecho, la recomendación del uso de mecanismos del tipo `Bottom-half` fue incluida en la última versión, la cual está siendo utilizada en el desarrollo del software de la ICU del Solar Orbiter (Sánchez et al., 2012). Sin embargo esto no sucede en el caso de las restricciones a nivel de componente.

EDROOM no define una taxonomía de componentes como las presentadas en las restricciones. Por ello hemos propuesto una taxonomía idéntica a la explicada en las restricciones, lo que implica que hemos definido los tres tipos de componentes, proactivo, reactivo y recurso compartido. A esto sólo hay que añadir que tanto los componentes del tipo proactivo como los componentes del tipo reactivo tienen asociados hilos de ejecución mientras que los recursos compartidos, no. Este matiz es una cuestión de eficiencia, debido a que los recursos compartidos sólo pueden aceptar mensajes síncronos, por lo que no tiene sentido que tengan un hilo de ejecución asociado, lo que significa un mayor gasto de recursos.

EDROOM define el comportamiento de cada uno de los componentes mediante máquinas de estado. Define un conjunto de estados y transiciones. Cada transición tiene asociado un conjunto de acciones en forma de código empotrado (*place-holder*). Este código de programación es código C++, el cual se ejecutará cada vez que es ejecutada la transición. El problema de esta aproximación es que no cumple la restricción relativa a la definición explícita de la naturaleza de las reacciones de los componentes que propusimos en la sección anterior. La modificación llevada a cabo para habilitar esta restricción es la definición de un elemento que denominamos `msgHandler`. Entonces las transiciones ya no tienen asociado de una manera directa el código, sino un `msgHandler`. Los `msgHandler` están compuestos por un conjunto de elementos que denominamos `msgHandlerItem`, los cuales por un lado tienen asociados código C++ y por otro lado definen una determinada naturaleza. Los tipos de naturaleza de los `msgHandlerItem` son los mismos que definimos en las restricciones. Éstos son: 1) MsgDataHandler, 2) Action, 3) Send, 4) Invoke y 5) MsgReply.

La lista con los elementos `msgHandler` está enumerada porque el orden de ejecución de los `msgHandlerItem` depende de su naturaleza. Este orden de ejecución es debido al mismo motivo por el que definimos las diferentes naturalezas

de las reacciones de los componentes. Esto es, conocer de una manera explícita la reacción de un componente.

La inclusión de una taxonomía de componentes y la redefinición de los *place-holders* mediante los `msgHandler` habilitan la integración de EDROOM en el proceso propuesto en esta Tesis Doctoral. Con estas restricciones aplicadas los *end-to-end* podrán ser reconstruidos y los sistemas resultantes de esta herramienta estarán libres de *deadlocks*. Gracias a esta propiedad, el proceso presentado en esta Tesis Doctoral tiene la propiedad de *correctness by construction*.

3.4. Modelo transaccional de diseño TSDM

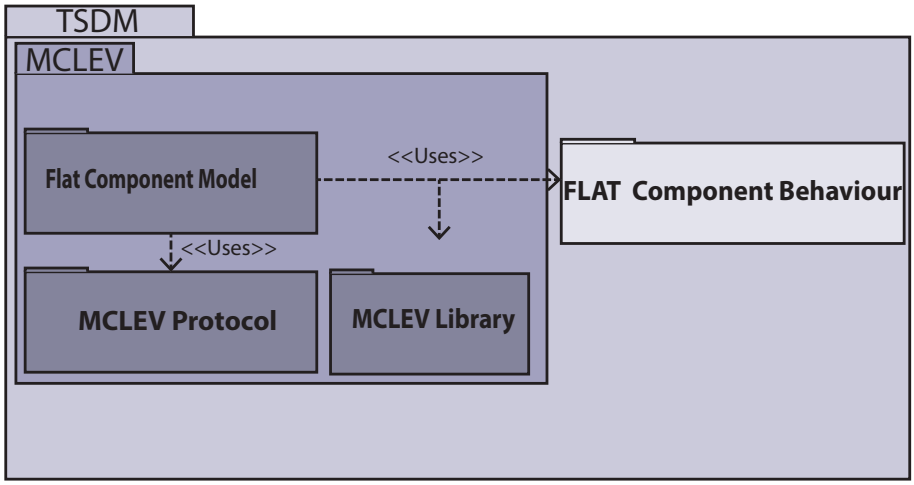


Figura 3.3: Se muestran los modelos que conforman el modelo TSDM. Está formado por modelos MCLEV, los cuales forman parte de la infraestructura de MICOBS. Además ha sido definido un nuevo modelo, que se denomina FCM.

El modelo TSDM describe un modelo simplificado cercano a los modelos formales de diseño, siendo una de sus características no depender de ninguna tecnología de componentes subyacente. Esto lo convierte en un modelo pivote para la integración de modelos formales de diseño en el proceso.

El modelo TSDM describe un sistema mediante:

- Componentes.
- Bibliotecas de servicio.
- Protocolos.

En la imagen 3.3 mostramos todos los modelos que conforman el modelo TSDM. En las próximas subsecciones definimos sus funcionalidades.

3.4.1. Elementos que son usados de la infraestructura MICOBS

En la definición del modelo TSDM hemos usado elementos de la infraestructura de MICOBS que nos permitió definir con mayor facilidad el modelo TSDM. En especial, el *framework* MICOBS contiene un modelo denominado MCLEV (Parra et al., 2011). Los modelos de los cuales está compuesto MCLEV son mostrados en la imagen 3.3. A continuación pasamos a describirlos:

- **MCLEV Flat Component System**, modelo que describe la composición del sistema mediante componentes. El atributo *Flat* indica que este modelo no expresa jerarquías de componentes. La descripción de los componentes se define mediante una estrategia de caja negra, esto significa que define los puertos por los cuales los componentes se suscriben o demandan servicios, pero no el detalle de su comportamiento. Estos puertos están asociados a unos determinados protocolos. Además, también definen la composición del sistema mediante las relaciones entre los diferentes componentes a través de sus puertos.
- **MCLEV Protocol**, modelo que describe los protocolos. Éstos son definidos en función de los mensajes que pueden ser enviados o recibidos por los componentes.
- **MCLEV Library**, modelo que describe las bibliotecas que dan un soporte funcional a los componentes. Describe las interfaces públicas de cada una de las bibliotecas.

Todos los modelos del paquete forman parte del *framework* MICOBS a excepción del modelo FCM, del cual hablaremos extensamente en la siguiente sección.

3.4.2. Descripción del FCM

Hemos visto que el modelo MCLEV no contiene ningún modelo que permita describir el comportamiento de los componentes. Como ya vimos, las respuestas de los componentes tienen que ser explícitas para que los modelos de análisis puedan ser generados, confiriendo al modelo la propiedad de ser analizable. Por lo que hemos definido un nuevo modelo que junto con los modelos provistos por MCLEV completa el modelo TSDM.

Este nuevo modelo es el FCM. El modelo FCM describe las reacciones de los componentes. La descripción de las reacciones se realiza mediante una taxonomía que hemos propuesto, la cual tiene como objetivo facilitar la identificación de los tipos de transiciones que pueden darse en la definición de las máquinas de estado de Harel. Una vez que un mensaje es recibido por parte del componente, las reacciones son definidas en función de cadenas de transiciones que son ejecutadas.

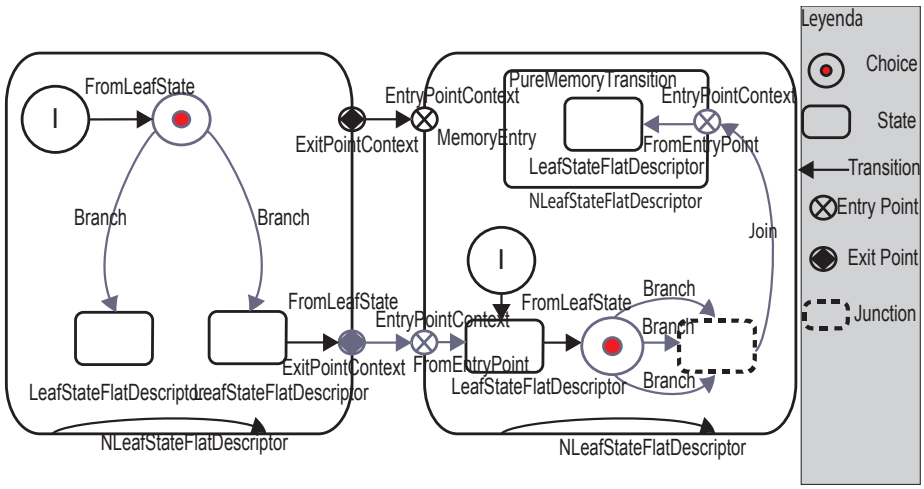


Figura 3.4: Mostramos un ejemplo de una máquina de estado típica representada con el formalismo de Harel. El ejemplo consta de una máquina de estados que presenta cada uno de los elementos que son categorizados en el modelo FCM. Podemos ver la categorización de elementos relativos a estados, transiciones, cambio de contextos y elementos de decisión.

Hemos definido una taxonomía basada en las máquinas de estado de Harel —en la figura 3.4 se muestra un ejemplo de una máquina de estado típica representada por el formalismo de Harel—. Ésta tiene como propósito categorizar las reacciones mediante sus construcciones sintácticas, ampliamente usadas en modelos constructivos, como es el caso de ROOM o UML-2. Una abstracción importante incluida en este modelo es la que se ajusta al de *Run-To-Completion* (RTC). Esta abstracción implica que se ejecute el mensaje recibido sin que pueda interrumpirse por la llegada de otro, puesto que sólo hay un hilo de ejecución. La ejecución implica una secuencia de pasos (*steps*) hasta alcanzar un estado destino. Algunos *steps* son pseudoestados que están explícitos en el modelo como los *choice* y las *juntion* (no confundir con los *join*, puesto que no hay varios hilos en un componente, sólo uno). Ambos pseudoestados están definidos en UML y en el *StateChart* de Harel, así como en otros modelos de diseño, como los *activity diagrams*. El modelo también incluye *steps* asociados a tipos de pseudoestados más complejos como son las transiciones con historia, o las entradas (*entry*) y salidas (*exit*) de contextos, que permiten definir una jerarquía de estados.

Elementos FCM

En esta sección definiremos la taxonomía de máquina de estados basada en el formalismo de Harel, y una abstracción paralela que tiene el objetivo de componer cadenas de `msgHandler`. En la imagen 3.5 mostramos un resumen con cada uno de los elementos que son usados en la taxonomía.

Primero definimos la taxonomía de transiciones que hemos propuesto. Ésta se divide en varios grupos, permitiéndonos ilustrar con mayor facilidad. Estos grupos son:

- **Estados**, categorizan los estados.
- **Transiciones de decisión**, categorizan los puntos de decisión.
- **Elementos de cambio de contexto**, categorizan los elementos que producen cambios de contexto.
- **Transiciones de memoria**, categorizan las transiciones de memoria.
- **Transiciones reactivas a mensajes**, categorizan los tipos de transiciones en función de si son reactivos a mensajes.

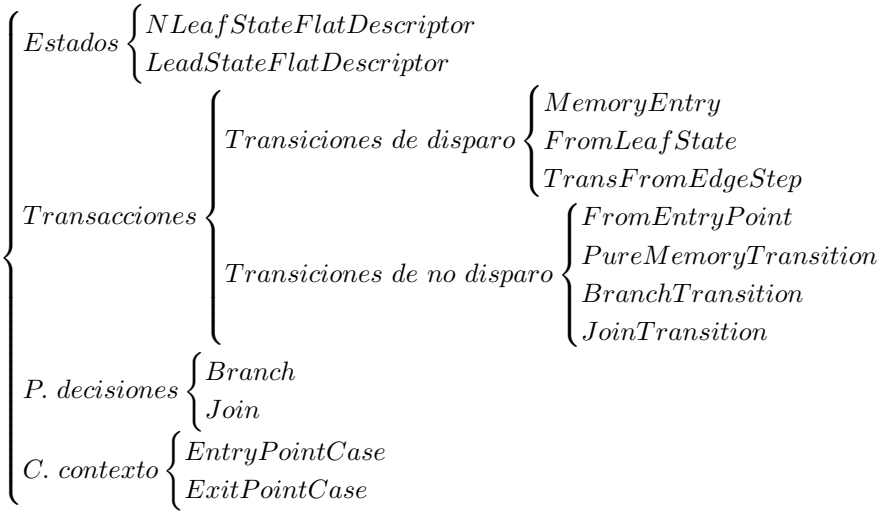


Figura 3.5: Taxonomía propuesta para categorizar los elementos de una máquina de estados, basada en el formalismo de Harel.

Estados

Diferenciamos dos tipos de estados. Éstos son:

- `NLeafStateFlatDescriptor`, corresponde a los estados que contienen en su interior más estados, a éstos se pueden acceder mediante elementos de cambio de contexto.
- `LeafStateFlatDescriptor`, corresponde a los estados que no contienen más estados en su interior.

La diferencia entre ambos tipos de estado es interesante por varios motivos. El primero de ellos es que todas las transiciones que partan del tipo `LeafState` serán transiciones reactivas a mensajes, mientras que para el tipo `NLeafState` no todas las transiciones serán reactivas, ya que este tipo de estados se pueden comportar como un pseudoestado en algunos casos. En este aspecto profundizaremos más adelante, viendo los tipos de transiciones. La segunda es que en caso de ser un estado `NLeafState` este elemento contendrá más contextos de una manera recursiva, lo que permite diferenciar de una manera explícita la jerarquía de estados.

Elementos de decisión

Hay dos tipos de elementos de decisión.

- `Branch`, corresponde a las transiciones que dependen del resultado de un elemento de guarda.
- `Join`, corresponde a las transiciones que convergen desde un conjunto de transiciones a una sola transición.

Esta categorización permite distinguir entre los dos posibles elementos que pueden modificar el flujo del comportamiento de un componente.

Transiciones de memoria

Las transiciones de memoria permiten retornar a la última configuración que había dentro de un contexto.

Hay dos tipos de transiciones de memoria, las cuales son:

- `MemoryEntry`, corresponde a todos los `EntryPoint` que no están conectados a una transición.

- `PureMemoryTransition`, para cada contexto se crea un elemento de este tipo que modela el hecho de alcanzar este contexto fruto de una transición con memoria definida en un contexto superior. Este elemento permite componer las transiciones con memoria.

Elementos de cambio de contexto

Estos elementos permiten comunicar transiciones entre estados de diferentes niveles de jerarquía.

Diferenciamos dos tipos de elementos de cambio de contexto, los cuales son:

- `EntryPointContext`, corresponde a la entrada en un contexto fruto de la llegada de una transición definida en el contexto superior.
- `ExitPointContext`, corresponde a la salida de un contexto fruto de una transición de partida definida en el contexto superior.

Transiciones

Diferenciamos dos tipos de transiciones reactivas a mensajes, las cuales son:

- **Transiciones de disparo (TriggerStep)**, corresponden a las transiciones disparadas por la recepción de mensajes.
- **Transiciones de no disparo (NoTriggerStep)**, corresponde a las transiciones que son ejecutadas a instancia de su relación con una transición de disparo, formando parte de las respuestas explícitas de un mensaje. Las que parten de un punto de entrada, las que definen un punto de salida de un contexto al que llega una transición y las transiciones con memoria `PureMemoryTransition`.

Esta categorización de las transiciones es bastante interesante. Nos permite identificar cada una de las transiciones iniciales que serán ejecutadas cuando se reciba un mensaje en forma de transiciones de disparo. Esto permite identificar rápidamente los primeros elementos de respuesta.

Cada una de las categorías anteriores de transiciones, a su vez, se categorizan de nuevo. El objetivo es tener una representación detallada de los casos en los que una transición es del tipo de disparo o no. El elemento que hemos usado en este caso es el relativo a los elementos de origen o destino de una transición para identificar a qué conjunto van a pertenecer.

- `Transiciones de disparo (TriggerStep)`.
  - **`FromExitPointFree`**, corresponde a las transiciones que tienen como origen un `ExitPoint`, no siendo éste el destino de ningún tipo de transición.
  - **`MemoryEntry`**, corresponde a las transiciones del tipo memoria.
  - **`FromLeafState`**, corresponde a las transiciones que tienen como origen un `LeafStateFlatDescriptor`.
  - **`TransFromEdgeStep`**, corresponde a las transiciones de borde, asociadas a la membrana de los contextos de un estado del tipo `NLeafStateFlatDescriptor`.
- **Transiciones de no disparo (NotTriggerStep)**.
  - **`FromexitPointNotFree`**, corresponde a las transiciones que tienen como destino un `ExitPoint`, siendo éste el origen de otra transición.
  - **`FromEntryPoint`**, corresponde a las transiciones que tienen como origen un `EntryPoint`.
  - **`PureMemoryTransition`**, corresponde a cada estado `NLeafState`, el cual tiene asociado un elemento de este tipo.
  - **`BranchTransition`**, corresponde a las transiciones que tienen como origen un elemento del tipo `Branch`.
  - **JoinTransition**, corresponde a las transiciones que tienen como origen un elemento del tipo `Join`.

`MsgHandler`

Estos elementos definen las acciones asociadas a las transiciones. Éstas serán ejecutadas cuando la transición tome lugar. Diferenciamos dos tipos de `MsgHandler`, los cuales son:

- **`MsgHandlerWithTrigger`**, corresponde a los `MsgHandler` que están asociados a transiciones de disparo.
- **`MsgHandlerWithOutTrigger`**, corresponde a los `MsgHandler` que están asociados a una transición de no disparo.

Esta categorización permite comprobar directamente si un `MsgHandler` reacciona a la recepción de un mensaje. De esta manera no hace falta comprobar la transición a la que está vinculada para saber si es reactivo o no. Gracias a esta cualidad se pueden conocer de una manera explícita las acciones que se ejecutarán.

Cadenas de `MsgHandler`

Las cadenas de `MsgHandler` describen todas las acciones que serán ejecutadas en una determinada secuencia cuando un mensaje es recibido. Estas cadenas de `MsgHandler` se componen de dos tipos:

- El primer elemento de la cadena de `MsgHandlers` es del tipo `MsgHandlerWithTrigger`.
- Los consecuentes `MsgHandlers` tras el primero son del tipo `MsgHandlerWithOutTrigger`.

También hay que tener en cuenta que dependiendo del tipo de transición asociada al `MsgHandlerWithTrigger` los siguientes `MsgHandlerWithOutTrigger` deberán estar restringidos en función de su tipo de transición asociadas. Éstas se enumeran a continuación:

- **`FromExitPointFree`**, son válidos todos los `MsgHandlerWithOutTrigger`, excepto los asociados a transiciones de memoria, como son los elementos `PureMemoryTransition`.
- **`FromLeafState`**, son válidos todos los `MsgHandlerWithOutTrigger`, excepto los asociados a transiciones de memoria, como son los elementos `PureMemoryTransition`.
- **`MemoryTransition`**, sólo son válidos los `MsgHandlerWithOutTrigger` que están asociados a transiciones de memoria, como son los elementos `PureMemoryTransition`.
- **`TransFromEdgeStep`**, sólo son válidos los `MsgHandlerWithOutTrigger` que están asociados a transiciones de memoria, como son los elementos `PureMemoryTransition` y los elementos `FromEntryPoint`.

Analogías con el modelo EDROOM

El FCM categoriza la semántica de las máquinas de estado de Harel. Estas máquinas de estado son usadas por el modelo de diseño EDROOM que integramos en este proceso. Ambas sintaxis son equivalentes y la información no se pierde en ninguno de los sentidos.

Abstracciones como los `MsgHandler` del modelo FCM eliminan el alcance (jerarquía) entre estados, los cuales son descritos en el modelo EDROOM. Este tipo de estrategia está motivada por el divide y vencerás, donde el objetivo es descomponer las funcionalidades del sistema. Pero en el caso de los modelos de análisis los alcances y sus descomposiciones no son los mismos. Por este motivo, las cadenas de `MsgHandler` del modelo FCM son introducidas con el objetivo de hacer visibles las reacciones a los mensajes por parte de un componente y eliminar el alcance de los estados y pseudoestados (jerarquía). En la imagen 3.6 se muestra, en forma de ejemplo, una comparativa entre el formalismo de Harel, un modelo intermedio que identifica las transacciones disparadas por mensajes y el formalismo usado para la descripción de los FCM.

La taxonomía de los elementos sintácticos por el modelo TSDM permite facilitar la transformación hacia cualquier otro modelo. Pueden ser descritas reglas de transformación para cada uno de los elementos. Las transiciones son descritas teniendo en cuenta estados o pseudoestados, o puede usarse la relación de las cadenas de `MsgHandler`. Ésta última permite conocer la secuencia de acciones del modelo EDROOM que se ejecutan cuando un mensaje es recibido.

3.4.3. Transformación desde el modelo EDROOM al modelo TSDM

La transformación ha sido dividida en cuatro pasos. El `Modelado de componentes` es el primer paso, en el que se asocia un componente EDROOM a un componente del TSDM. En el segundo paso `Comunicación entre componentes`, se especifican los protocolos de comunicación que permiten la comunicación entre componentes. En el siguiente paso `Transformación de bibliotecas de servicio` se transforman los tipos de datos definidos en EDROOM y las bibliotecas de servicio que son usadas por las mismas. Para finalizar, en el último paso, se transforma el comportamiento de los componentes desde el modelo EDROOM al modelo FCM. En el cuadro 3.1 se muestra un resumen de las transformaciones propuestas.

La transformación desde EDROOM a MICOBS ya forma parte de la propia infraestructura de MICOBS.

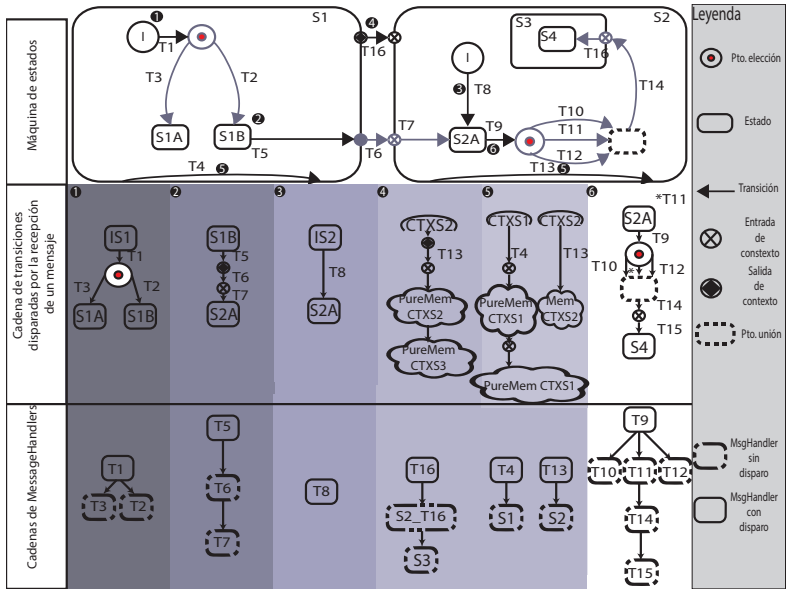


Figura 3.6: Ejemplo de las composiciones de las cadenas de `MsgHandler` del modelo FCM. La imagen está compuesta por tres filas, donde la primera muestra una máquina de estado, la siguiente la composición de las transiciones que toman lugar cuando un mensaje llega y la última muestra la composición de los `msgHandlers`. Cada número corresponde con cada una de las cadenas que se pueden formar. Vemos, en la última fila, cómo en cada cadena el primer elemento es un `MsgHandlerWithTrigger`, seguido de un `MsgHandlerWithOutTrigger`.

La integración de EDROOM en MICOBS ya aporta los modelos MCLEV, por lo que sólo ha sido necesario definir la transformación relativa al modelo FCM.

Transformación desde la descripción de comportamiento de EDROOM al modelo FCM

Cuadro 3.1: Resumen de la transformación desde EDROOM a FCM.

Elementos de EDROOM.	Detalle EDROOM.	Elemento de FCM.
Estados.	Estado: E_i.	Un estado del algún tipo por cada: E_i.
	Estado E_i con contexto.	Definimos un `NLeafStateFlatDescriptor`.
	Estado E_i sin contexto.	Definimos un `LeafStateFlatDescriptor`.
Elementos de decisión.	Elemento de decisión: Dcs_i.	Se define algún tipo elemento de decisión.
	Dcs_i `BranchPoint`.	Se define un `Branch`.
	Dcs_i `JoinPoint`.	Se define un `Joint`.
Transiciones de memoria.	Transición de memoria: Trm_i.	Se define algún tipo de transición de memoria.
	Por cada E_i del tipo `NLeafStateFlatDescriptor`.	Define un `PureMemoryTransition`.
	Por cada Trm_i.	Se define un `MemoryEntry`.
	Por cada Trm_i.	Se resuelve la cadena de `PureMemoryTransition`.
Elementos cambio de contexto.	Cambio de contexto: Ctx_i.	Se define algún elemento de cambio de contexto.
	Por cada Tr_i de entrada a un E_i con contexto.	Se define un `EntryPointContext`.
	Por cada Tr_i de salida a un E_i con contexto.	Se define un `ExitPointContext`.
	Por cada Trm_i.	Se resuelve la cadena de `PureMemoryTransition`.
Transiciones de contexto.	Transiciones: $TrCtx_i$.	Se define algún tipo de transición de contexto.
	Por cada $TrCtx_i$ elemento de origen, un Ctx_i de salida.	Se define `FromExitPointFree`.
	Por cada $TrCtx_i$ elemento de origen, un E_i del tipo `LeafStateFlatDescriptor`.	Se define un `FromLeafState`.
	Por cada $TrCtx_i$ con origen y destino, el mismo contexto E_i.	Se define `TransFromEdgeStep`.
Transiciones no de contexto.	Transiciones: Tr_i.	Se define algún tipo de transición de no contexto.
	Por cada Tr_i con destino, un Ctx_i no libre.	Se define `FromExitPointNotFree`.
	Por cada Tr_i con un origen Ctx_i.	Se define `FromEntryPoint`.
	Por cada Tr_i que tiene como origen un Dc_i tipo `Branch`.	Se define `BranchTransition`.

continúa

continúa

Elementos de EDROOM.	Detalle EDROOM.	Elemento de FCM.
	Por cada Tr_i que tiene como origen un Dc_i del tipo `Join`.	Se define `JoinTransition`.
`MsgHandlers`.	`MsgHandler`: Msg_i.	Se define algún `MsgHandler`.
	Por cada Msg_i asociado a una Tr_i del tipo Disparo.	Se define `MsgHandlerWithTrigger`.
	Por cada Msg_i asociado a una Tr_i del tipo No Disparo.	Se define `MsgHandlerWithOutTrigger`.
Cadenas de `MsgHandler`.	`MsgHandler`: Msg_i.	Se define una cadena de $m\vec{s}g_i$.
	Por cada Msg_i tipo asociado `MsgHandlerWithTrigger`.	Se resuelve su Tr_i asociada hasta el siguiente Msg_i tipo asociado `MsgHandlerWithTrigger`, define $m\vec{s}g_i$.

Los modelos que participan en esta transformación son por un lado la definición del comportamiento de componentes de EDROOM y, por otro lado, el modelo FCM. La transformación de los diferentes elementos del modelo EDROOM se describe a continuación.

Estados

Por cada estado con contexto del modelo EDROOM se genera un estado del tipo `NLeafStateFlatDescriptor`. Por cada estado EDROOM sin contexto se genera un componente del tipo `LeafStateFlatDescriptor`.

Elementos de decisión

Por cada `BranchPoint` y `JoinPoint` del modelo EDROOM se generan un `Branch` y un `Join` del modelo FCM.

Transiciones de memoria

Esta transformación se divide en tres pasos. En el primero se definen todos los elementos del tipo `PureMemoryTransition`, en el segundo se resuelve la cadena de dependencia entre todos los `PureMemoryTransition` y el tercero genera los elementos `MemoryEntry`. Más en detalle:

1. Los elementos `PureMemoryTransition` son generados por cada estado del tipo `NLeafStateFlatDescriptor`.

2. La dependencia entre los diferentes elementos del tipo `PureMemoryTransition` se resuelve con la relación de los `NLeafStateFlatDescriptor`. Si un `NLeafStateFlatDescriptor` contiene otro `NLeafStateFlatDescriptor`, éstos tienen una relación de dependencia de elementos `PureMemoryTransition`. De esta manera se crea la cadena de propagación entre contextos de los `PureMemoryTransition`.

3. Se comprueban los elementos en el modelo EDROOM que corresponden a los `MemoryEntry` generando este tipo de elementos y relacionándolos con el `PureMemoryTransition` correspondiente al `NLeafStateFlatDescriptor` donde éste se encuentra definido (Son los puntos de entrada sin transición de partida) —recordemos que los `MemoryEntry` son aquellos `EntryPointContext` que no tienen transiciones de partida—.

Elementos de cambio de contexto

Por cada estado en el modelo EDROOM del tipo `NLeafStateFlatDescriptor` se comprueban todas sus transiciones. Todas las transiciones de llegada generan elementos del tipo `EntryPointContext` y todas las transiciones de salida generan elementos del tipo `ExitPointContext`. Estas transformaciones son descritas en el cuadro 3.1.

Transiciones

Todas las transiciones del modelo EDROOM se transforman en sus elementos correspondientes del modelo FCM. Las transiciones presentadas en 3.4.2 son transformadas según lo descrito en el cuadro 3.1. Este cuadro agrupa las transformaciones en los siguientes grupos:

- **Transiciones de memoria**, son transformadas todas aquellas transiciones relativas a transiciones de memoria ya sean bien los elementos `PureMemoryTransition` o `MemoryEntry`.
- **Transiciones de contexto**, son transformadas todas aquellas transiciones que provienen del alcance de este contexto. Por ejemplo entre ellas están las transiciones del tipo `FromExitPointFree` y `FromLeafState`.
- **Transiciones de no contexto**, son transformadas todas aquellas transiciones que provienen del alcance de otro contexto, ya que están relacionadas con **Elementos de cambio de contexto**. Por ejemplo entre estas transiciones están `FromExitPointNotFree` y `FromEntryPoint`.

`MsgHandler`

Se transforman todos los `MsgHandler` del modelo EDROOM a un elemento del tipo `MsgHandlerWithTrigger` o un elemento del tipo `MsgHandlerWithOutTrigger`, dependiendo de si está o no vinculado a una transición del tipo de disparo. El detalle de la transformación se muestra en en el cuadro 3.1.

Composición de transiciones

Por último, se resuelven las cadenas de `MsgHandler` que son ejecutadas cuando un mensaje se recibe. Como ya vimos, su primer elemento es del tipo `MsgHandlerWithTrigger` y el resto de elementos es del tipo `MsgHandlerWithOutTrigger`. Estas cadenas se componen de la siguiente manera:

1. Cada `MsgHandlerWithTrigger` tiene que ser resuelto y formar una cadena.

2. Son resueltos mediante la transición asociada a cada `MsgHandlerWithTrigger`. Desde esta transición se resuelven los restantes elementos recorriendo las diferentes transiciones de una manera recursiva, sólo se detendrá en el caso de no encontrar una transición que no sea del tipo No disparo, profundiza siempre que no encuentre un estado hoja.

3. Por cada elemento que ha sido recorrido se extrae su `MsgHandlerWithOutTrigger`, el cual se asocia a la cadena de `MsgHandler`.

3.4.4. Resumen

Hemos descrito un modelo orientado al diseño y basado en el paradigma CBSE, denominado TSDM. Una de sus principales características es ser independiente de una determinada tecnología de componentes subyacente, pero es compatible con herramientas basadas en UML. Su principal objetivo es facilitar la transformación desde los modelos de diseño a los modelos de análisis. Para ello define un modelo basado en componentes, donde la distancia de los elementos es cercana, define componentes, protocolos, librerías de servicio y las reacciones de los componentes se expresan mediante máquinas de estado. Al mismo tiempo categoriza la semántica de las máquinas de estado de Harel, con el objetivo de facilitar las analogías entre el modelo de diseño y el modelo resultante de análisis. Por este motivo este modelo forma parte de la espina dorsal del proceso junto con el modelo, que veremos a continuación, que denominamos TSAM.

El modelo TSDM está formado por modelos provistos por la infraestructura de MICOBS y un modelo que hemos creado para la definición del comportamiento de los componentes FCM. Ambos elementos completan un modelo pivote de diseño.

En esta sección hemos visto en detalle el modelo FCM. Han sido presentadas una taxonomía de la sintaxis de las máquinas de estado de Harel y una transformación desde el modelo EDROOM al modelo TSDM. Con estos elementos, por un lado, se presenta un modelo que facilita la integración de modelos basados en el paradigma CBSE, y por otro, se presenta un caso práctico de la integración de la herramienta EDROOM mediante su transformación a su equivalente TSDM.

3.5. Transactional System Analysis Model

3.5.1. Introducción

El modelo TSAM es un modelo orientado al análisis basado en el paradigma CBSE. La definición del sistema está basada en componentes que se suscriben o demandan servicios a través de sus puertos. Los componentes también pueden demandar servicios mediante dependencias sobre bibliotecas. Cada uno de estos componentes o bibliotecas de servicio definen sus reacciones en forma de anotación de propiedades EFP. Esto permite que se pueda usar este modelo para la generación de modelos de análisis.

El modelo TSAM categoriza las EFP en función de su plataforma. Esto permite desacoplar la plataforma cuando se realiza un determinado análisis.

Este modelo está compuesto por un conjunto de modelos, los cuales descomponen de una manera ortogonal los elementos necesarios para definir un sistema desde la perspectiva del análisis. Estos modelos son mostrados en la figura 3.7, donde cada modelo descrito será explicado de una manera concisa durante el desarrollo de esta sección. Pero podemos decir que básicamente este modelo está compuesto por:

- Componentes, cuyas reacciones están anotadas con propiedades EFP.
- Bibliotecas de servicio, cuyas reacciones están anotadas con propiedades EFP.
- Protocolos de comunicación.
- Patrones de activación, que describen el ratio de suceso de los eventos.

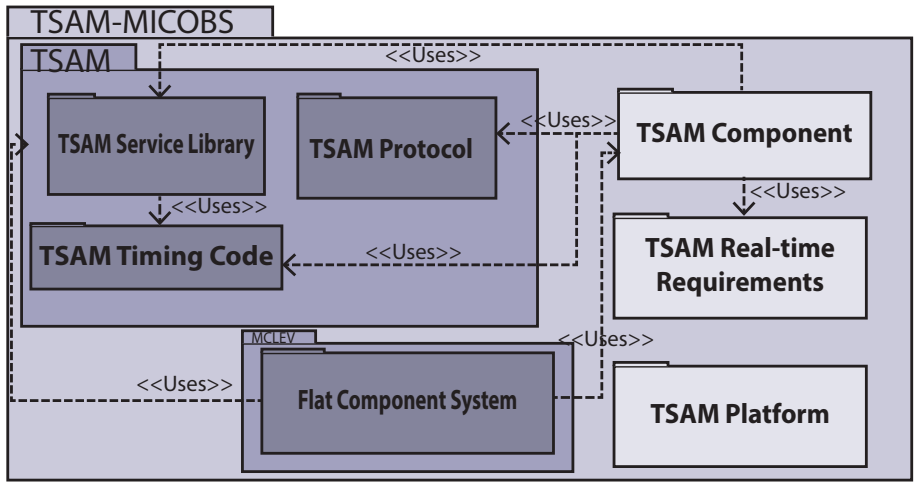


Figura 3.7: Se muestran los modelos que componen el modelo TSAM. Vemos los modelos ya definidos en la versión estándar de MICOBS y los nuevos modelos que han tenido que ser definidos para completar dicho modelo. Por ejemplo, el modelo FCS de MCLEV que forma parte del modelo TSDM es también usado en el modelo TSAM.

3.5.2. Elementos propios de la infraestructura de MICOBS

MICOBS tiene un enfoque multiplataforma. Esto significa que los elementos que forman parte del modelo TSAM deben estar anotados con el vector que define la plataforma. Este vector propuesto por MICOBS está formado por:

- Arquitectura.
- Placa.
- Compilador.
- Sistema operativo.
- *Application Programming Interface* (API) del sistema operativo.

Este vector puede ser anotado con valores explícitos o con un valor comodín. Este valor es `any`, que denota que una de las dimensiones no es significativa del elemento que está siendo anotado.

En la definición del modelo TSAM hemos usado elementos de la infraestructura MICOBS, que nos permitió definir con mayor facilidad el modelo TSAM. En especial, el *framework* MICOBS contiene varios modelos que permiten la anotación de propiedades EFP, la descripción del sistema de componentes y la definición de bibliotecas. Estos modelos son:

- ***Model Flat Model Architecture Desing*** **(FLATMCAD)**, este modelo define el sistema mediante composición de componentes. Define los componentes que conforman el sistema y cómo se relacionan entre ellos mediante la topología de comunicación.
- ***Transactional System Analysis Model Service Library*** **(TSAMSL)**, este modelo contiene la definición de las bibliotecas de servicio usadas en el sistema.
- ***Transactional Analisys Timing Annotated Code*** **(TACode)**, este modelo contiene indicadores relativos al tiempo de ejecución de componentes y librerías de servicio.

Model Flat Model Architecture Design

El modelo FLATMCAD describe las instancias de los componentes y la topología de la comunicación. Este modelo integra todas las instancias de componentes que forman el sistema. Hay que tener en cuenta que este modelo se denomina *Flat* porque no puede presentar ninguna clase de jerarquía de componentes. Esto es debido a que estrategias de jerarquías de componentes son útiles a la hora de describir sistemas de diseño pero no de análisis. Por otro lado, la topología de comunicación resuelve las conexiones entre los puertos de los componentes.

Ya veremos en las transformaciones a modelos de análisis que este modelo es clave en las transformaciones. Nos ayudará a componer reacciones completas del sistemas, más allá del alcance de un solo componente, de tal manera que podremos relacionar todas las reacciones que se llevarán a cabo en el sistema cuando un evento externo sea disparado.

Transacional System Analysis Service Lybrary Model

El modelo TSAMSL describe las bibliotecas de servicio. Su función es describir las propiedades EFP de los métodos públicos y privados de las bibliotecas de servicio. Su anotación está basada en la descripción por parte de cada uno de

los métodos de sus posibles caminos de ejecución (*execution-path*). Cada camino se anota con los tiempos de ejecución mediante una relación con elementos del tipo TACode.

Los elementos que son definidos por cada biblioteca de servicios TSAMSL son:

- ***Service Access Interface* (SAI)**, librería de servicio.
- ***Service Access Point* (SAP) público**, interfaz pública de la librería de servicio.
- ***Service Access Point* (SAP) privado**, todos los métodos que son integrados en la biblioteca.

Transactional Analisys Timming Code

El modelo TACode[1] es el encargado de anotar tiempos de ejecución. Los parámetros de tiempos de ejecución usados son los necesarios para poder realizar análisis de planificabilidad y rendimiento. Los parámetros que se definen son:

- **WCET** , el peor tiempo de ejecución. Es un valor analítico que deberá ser anotado con el «peor tiempo de ejecución posible para una determinada plataforma».
- **Tiempo de ejecución máximo**, el tiempo máximo debe ser anotado con un valor experimental basado en el mayor tiempo observado.
- **Tiempo de ejecución mínimo**, el tiempo mínimo debe ser anotado con un valor experimental con el menor tiempo observado.
- ***High Water Mark Analitical Time Profile* (HVT)**, identifica el camino con el peor tiempo ejecutado. Este valor deberá ser cercano al tiempo de WCET.

Transactional Analysis Protocol Model

Este modelo transaccional es el encargado de la definición de los protocolos de comunicación. Un protocolo puede definir mensajes de entrada y de salida.

[1] Actualmente la herramienta MICOBS cuenta con una integración con la herramienta RVS Nilas (2014). Esta herramienta permite caracterizar todos los parámetros de tiempos de ejecución que hemos descrito anteriormente.

La naturaleza de los mensajes puede ser del tipo síncrono o asíncrono. Por cada mensaje síncrono se define además un mensaje de respuesta que tiene que ser implementado por el receptor. A continuación se describen los elementos que componen un *Transactional System Analysis Model Protocol* (TSAMProtocol):

- **Mensajes de entrada**, conjunto de mensajes de entrada, denominados `TMessage`.
- **Mensajes de salida**, conjunto de mensajes de salida, denominados `TMessage`.

El elemento `TMessage` es la abstracción que se usa en el modelo TSAM para definir los mensajes. Esta abstracción puede ser de tres tipos:

- **TSendMsg**, define un mensaje de naturaleza asíncrono.
- **TInvokeMsg**, define un mensaje de naturaleza síncrona. Tiene asociado un conjunto de mensajes del tipo `TReplyMsg`.
- **TReplyMsg**, define un mensaje de naturaleza de respuesta a un mensaje síncrono.

Descripción de los nuevos elementos

Un conjunto de nuevos modelos fue definido y forma el núcleo del modelo TSAM. Estos modelos describen las siguientes funcionalidades:

- El comportamiento de los componentes.
- Definición de las restricciones temporales.
- Los patrones de activación de los eventos externos.
- Propiedades EFP relativas a la plataforma que tienen que ser tenidas en consideración para el análisis.

Para cubrir estas funcionalidades hemos definido los siguiente modelos:

- **TSAMComponent**, describe las reacciones de un componente cuando recibe un mensaje. La descripción de las reacciones es expresada en función de sus EFP.

- **TSAMRTRequirement**, describe, por un lado, las restricciones temporales en forma de *deadlines*, y por otro lado, define los patrones de activación de eventos externos.
- ***Transaccional System Analisys Platform Model* (TSAMPlatform)**, describe propiedades relativas a la plataforma, las cuales hay que tener en consideración durante el análisis.

Transactional Analysis Component Model

El modelo TSAMComponent es el encargado de describir el comportamiento reactivo de los componentes en términos de los mensajes que pueden recibir —en la figura 3.8 se muestra un ejemplo—. La descripción de la recepción se realiza en términos de sus EFP.

El `TSAMMessageHandler` es el elemento que define una reacción cuando un mensaje se recibe. Este elemento está compuesto por los siguientes parámetros:

- **Puerto y mensaje**, son los elementos que describen el puerto y el mensaje a los cuales el `TSAMMessageHandler` reacciona.
- `TSAMBasicHandler`, una secuencia de elementos que define la respuesta del `TSAMMessageHandler`.

Los `TSAMBasicHandler` a su vez definen una secuencia de elementos que describen diferentes tipos de reacciones. Estas reacciones son denominadas `TSAMMessageHandlerItems` y pueden ser de diferentes tipos:

- `TSAMMHiDataHandler`, define una acción que gestiona los datos adjuntos al mensaje recibido.
- `TSAMMHiAction`, define una acción cuyo efecto se limita al ámbito del `TSAMBasicHandler`. Esta acción no envía ni trata datos adjuntos al mensaje recibido.
- `TSAMMHiSend`, define el envío de un mensaje asíncrono. El puerto y el mensaje son parte de los parámetros de este elemento.
- `TSAMMHiInvoke`, define el envío de un mensaje síncrono. El puerto de origen y el mensaje son los parámetros.
- `TSAMMHiReply`, define la respuesta a un mensaje síncrono (`TSAMMHiInvoke`). Se debe especificar el mensaje de respuesta.

Mediante los elementos que acabamos de citar, los `TSAMMessageHandler`, los `TSAMBasicHandler` y los `TSAMMessageHandlerItems` definen de una manera explícita, las reacciones de tal manera que componen un modelo de análisis. Cuando el sistema reacciona a un evento externo, la cooperación entre componentes mediante paso de mensajes gestiona la reacción, que como hemos visto es descrita con los `TSAMMessageHandlerItems`. Los `TSAMBasicHandler` permiten agrupar secuencias de `TSAMMessageHandlerItems` para dotar a éstos de mayor claridad.

El orden de ejecución de los `TSAMMessageHandlerItems` está preestablecido. De esta manera se conoce explícitamente la secuencia de ejecución de los elementos. Este orden es: 1) TSAMMHiDataHandler, 2) TSAMMHiAction, 3) TSAMMHiSend, y 4) TSAMMHiInvoke. En la imagen 3.8 mostramos un ejemplo que muestra los elementos que acabamos de describir.

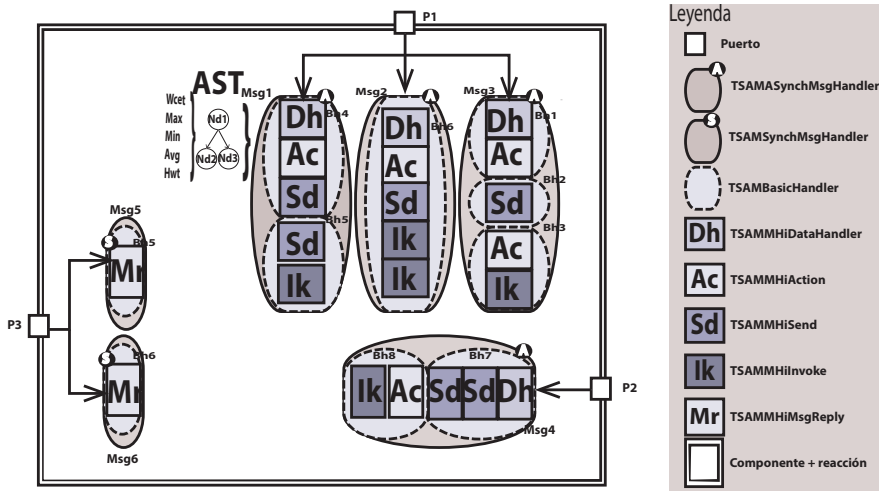


Figura 3.8: Se muestra un ejemplo de descripción de un componente en el modelo TSAMComponent. Este modelo se compone de cadenas de reacciones asociadas a un puerto y mensaje. Las reacciones se componen de `TSAMMessageHandler`, los cuales se componen a su vez de `TSAMBasicHandler`. En éstos últimos se definen las posibles acciones formadas por `TSAMMessageHandlerItem`.

Como hemos visto en el modelo que se ha presentado, hay dos maneras de cooperación entre componentes. Una de ellas, mediante el envío de mensajes asíncronos, y la otra, mediante el envío de mensajes síncronos. El envío asíncrono

no interrumpe el resto de las acciones que debe ejecutar el componente, mientras que el envío de mensajes síncronos detiene su ejecución hasta que el componente destino responda por el mismo puerto con un mensaje. Por este motivo han sido definidos dos tipos de `TSAMMessageHandler`; cada uno de ellos define de una manera explícita la naturaleza de la reacción a la que corresponde. De esta manera son definidos un `TSAMMessageHandler` de naturaleza asíncrona y otro de naturaleza síncrona. Más en concreto estos elementos son:

- **TSAMAsynchMsgHandler**, modela la respuesta a una recepción de mensajes asíncronos. Este tipo de manejador gestiona las reacciones asociadas a puerto y eventos que están suscritos a protocolos asíncronos.
- **TSAMSynchMsgHandler**, modela la respuesta a una recepción de mensajes síncronos. Este tipo de manejador gestiona las reacciones a puertos asociados a protocolos de tipo síncrono.

El tipo de `TSAMMessageHandlerItem` contenido en un `TSAMMessageHandler` está restringido por el tipo de `TSAMMessageHandler`. En concreto, un `TSAMAsynchMsgHandler` contiene todos los tipos de `TSAMMessageHandlerItem`, excepto el `TSAMMHiReply`, y el `TSAMSynchMsgHandler` sólo puede contener un `TSAMMHiReply` como respuesta a un mensaje síncrono.

No todos los componentes pueden definir los dos tipos de `TSAMMessageHandler`. Recuerde que hemos añadido una serie de restricciones para poder garantizar que el sistema está libre de abrazos mortales. Por este motivo hemos definido tres tipos de componentes idénticos a los que definimos en el modelo EDROOM y que forman parte de la hipótesis de la Tesis Doctoral. Estos son:

- Componente proactivo, define sólo `TSAMAsynchMsgHandler`.
- Componente reactivo, define tanto `TSAMAsynchMsgHandler` como `TSAMSynchMsgHandler`.
- Componente recurso compartido, define `TSAMSynchMsgHandler`.

Hay que tener en cuenta que los componentes del tipo proactivo y reactivo definen sus propios hilos de ejecución, donde se ejecutarán sus `TSAMMessageHandlerItems`, mientras que los componentes recursos compartidos, por su parte, no tienen ninguna tarea asociada. Por este motivo hemos categorizado los componentes proactivo y reactivo en un grupo que denominamos componentes tareas. Como los componentes recursos compartidos no tienen ninguna tarea asociada, sus `TSAMMessageHandlerItem` deben ejecutarse en un componente

externo a ellos. Esta tarea externa es la de un componente tarea cuya llamada activa es el `TSAMSynchMsgHandler` del componente recurso compartido. Esta relación no permite la definición de los recursos compartidos anidados, ya que éstos sólo pueden contener `TSAMSynchMsgHandler`, y éstos a su vez sólo pueden definir `TSAMMHiReply`. Estas restricciones son las que hemos visto anteriormente en la sección 3.2.1.

La interfaz entre componentes y bibliotecas de servicios se modela mediante los elementos `TSAMMessageHandlerItems`. Cada `TSAMComponent` determina qué servicios son demandados por parte del componente. Esta declaración se realiza en el modelo mediante una lista de los `TSAMSAI` requeridos. Al igual que las interfaces en otros modelos o lenguajes, el `TSAMSAI` se compone de un conjunto de funciones públicas llamadas `TSAMSAP`. Un `TSAMSAI` puede, a su vez, requerir de otros `TSAMSAI`, de tal manera que describen complejos árboles de dependencias. En el caso de que un `TSAMSAP` tenga una dependencia externa, es necesario especificar el nombre del `TSAMSAP`, el nombre del `TSAMSAI`, y el número máximo de llamadas que se hacen al `TSAMSAP`.

Los *items* básicos de manejo de mensajes, que denominamos `TSAMMessageHandlerItem`, son el equivalente a los `MsgHandlerItem` definidos en el TSDM, pero añadiendo explícitamente información sobre los distintos caminos de ejecución, la dependencia de los SAP de librerías de servicio de cada camino, y la información sobre el tiempo de ejecución. Entonces, los manejadores de mensajes se componen de una secuencia de `TSAMMessageHandlerItem` —hay que recordar que estos elementos se definen en el TSAMComponent—. Estos elementos (*items*) pueden ser de diferente naturaleza, en función de si modelan el tratamiento del dato adjunto de un mensaje, la simple ejecución de código o bien el envío de un mensaje de tipo asíncrono o síncrono. Cuando se modela cada `TSAMMessageHandlerItem` se deben definir un conjunto de caminos de ejecución o *codePaths* implícitos en la definición del elemento. Cada camino tiene asociado un elemento EFP. En nuestro caso cada camino anota las EFP del tipo TACode, que contienen la información relativa al peor tiempo de ejecución, además de otros indicadores como los tiempos mínimos, máximos y medios de ejecución.

Transactional System Analysis Real-Time Requirement Model

El modelo TSAMRTRequirement incluye información de los eventos, unos asociados a interrupciones externas, otros a servicios de temporización gestionados mediante interrupciones programadas. El modelo define los patrones de

disparo de los eventos, las restricciones temporales y las distintas situaciones de tiempo real en las que se disparan los eventos. Sus parámetros son:

- **Eventos externos**, define un conjunto de eventos externos. Cada uno se define mediante un tipo de patrón de activación y la definición del elemento que se activa en el sistema. En este caso la activación se define mediante la tupla componente, puerto y mensaje, que producirán la reacción del sistema.

- ***Deadlines***, define un conjunto de restricciones temporales. Estas restricciones se definen en función del tiempo que puede transcurrir como máximo desde el inicio de un evento hasta un elemento indicado en un componente. Este elemento indicado es un `TSAMMessageHandlerItem`, el cual es usado para la delimitación de las restricciones temporales. Queremos destacar que estas restricciones son definidas una vez que el sistema ha sido construido y no asociado a los componentes.

Las restricciones se expresan en forma de `TDeadlines` sobre la ejecución de los diferentes manejadores de los componentes disparados por un evento a nivel de sistema.

Los tipos de patrones de activación que son definidos están basados en patrones estándares en sistemas de tiempo real (Liu, 2000) —son los mismos que los definidos en el modelo MAST—. Tres tipos de patrones de activación pueden ser definidos, periódico, esporádico y de ráfaga. Éstos se especifican a continuación:

- **Patrón *bursty*/ráfaga**, este patrón nos ayuda a representar conjuntos de secuencias de interrupciones. Una interrupción que denominamos principal es la que modela la interrupción inicial. Una vez que sucede el evento tiene lugar una secuencia de interrupciones. Los parámetros que definen este patrón son el menor tiempo de llegada (*arrival time*) de la interrupción principal, el número máximo de interrupciones de la secuencia y el menor tiempo de llegada entre interrupciones (*inter-arrival time*).

- **Patrón esporádico**, este patrón nos ayuda a representar patrones de activación asociados a elementos de interrupción. El parámetro que forma parte de este patrón es el menor tiempo de suceso (*arrival time*) de la interrupción.

- **Patrón periódico**, este patrón permite modelar los eventos asociados a temporizadores o eventos que se comportan de una manera periódica. El parámetro asociado a este tipo de eventos es el periodo del evento (*period time*).

3.5.3. Transactional Platform Configuration Model

El modelo TSAMPlatform es el encargado de definir las propiedades de la plataforma donde el sistema será desplegado. Estas propiedades son relativas a parámetros como el tiempo de cambio de contexto, el tiempo de acceso y liberación de los *mutex*, gestión de temporizadores y alarmas, manejadores de interrupción y *jitter hardware*. Contiene las características tanto del *Real-Time Operating System* (RTOS) como del *Run-time* de componentes. Estas características son análogas a las que define la propia herramienta MAST en su Ecore (Cantabria, 2015).

Los parámetros presentes en este modelo se dividen en cinco grupos:

- `Procesador`, latencias relativas producidas por el microprocesador. Las latencias que son descritas en este modelo son:
  - **ISR**, latencias de la gestión de interrupciones. Se añaden los elementos máximo, mínimo y tiempo medio.
  - **Rango de prioridades válido**, se define el rango de prioridades válido para la gestión de las interrupciones.
  - **Temporizador del sistema**, se define el temporizador del sistema, el cual define su latencia.
- **Planificador**, latencias relativas al uso del planificador. El modelo define un conjunto extenso de políticas de planificación, de las cuales sólo se encuentra una activa. Éstas son las políticas de prioridades fijas con desalojo. Esto es debido a que uno de los requisitos de la hipótesis planteada era que este tipo de planificación debería estar activada. Los parámetros asociados a este tipo de planificador son:
  - **Rango de prioridades**, el rango de prioridades válidas que puede manejar el planificador.
  - **Latencia de cambio de contexto**, latencia asociada al cambio de contexto de una tarea.
- `Bottom-half`, número de hilos que gestionan los elementos `Bottom-half`. Este elemento es opcional, sólo se deberá configurar en caso de que el *Component Run-Time* (CRT) use este tipo de estrategia. Más en concreto, se debe especificar el número de hilos que gestionará la conversión de los eventos en mensajes.

- ***Mutex***, latencias introducidas por los mecanismos de acceso de exclusión mutua. Además, define el protocolo de acceso a los *mutex*. Hemos definido los dos protocolos de uso más extendido, los cuales son techo de prioridad inmediato y herencia de prioridad. Indistintamente del tipo de protocolo se deben definir dos tipos de latencias. Estas latencias son:
  - **Captura de un mutex**, latencia de la captura de un *mutex*.
  - **Liberación de un mutex**, latencia de la liberación de un *mutex*.
- **Temporizadores**, latencias introducidas por la gestión de temporizadores. Los temporizadores permiten introducir las latencias introducidas por los elementos de temporización. Sus parámetros son:
  - **Sobrecarga de la gestión del temporizador**, latencia producida por la sobrecarga de gestión de latencia del temporizador.
  - **Sincronización local**, temporizador asociado a uno o varios microprocesadores. Se especifica la latencia que introduce la sincronización.
  - **Precisión**, precisión en milisegundos del temporizador.
  - **Eventos**, conjunto de eventos que tienen asociado este temporizador.
- **Alarmas**, latencias introducidas por la gestión de alarmas.

3.5.4. Situaciones de tiempo real

Las situaciones de tiempo real permiten desglosar las reacciones del sistema por posibles categorías. De la misma manera que en los modelos de diseño se desglosan las funcionalidades del sistema mediante componentes en los modelos de análisis, éstos se desglosan en las reacciones válidas por situaciones de tiempo real en el sistema. Estas situaciones de tiempo real engloban las reacciones válidas dependiendo de la situación en la que se encuentre el sistema. Dos tipos de situaciones de tiempo real típicas son el modo nominal de funcionamiento y el modo de error. En ambas situaciones el sistema no reaccionará de la misma manera. Nos permite identificar conjuntos de reacciones válidas del sistema dependiendo de su situación. Podemos identificar escenarios de análisis que deben ser validados.

Las situaciones de tiempo real fueron integradas en el modelo TSAM para definir estrategias de análisis basadas en composición. Se integra por un lado desde una perspectiva de caja negra, mediante su asociación a eventos externos. Estos eventos externos sólo serán validados en análisis relativos a su situación de

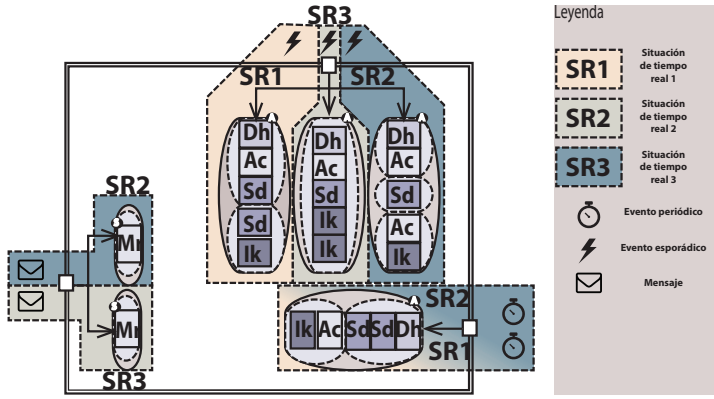


Figura 3.9: Se muestra un ejemplo con una configuración de situaciones de tiempo real. La imagen representa un componente TSAM donde las reacciones se describen mediante `TSAMMessageHandler` y `TSAMMessageHandlerItem`. En este caso observamos tres puertos, dos de ellos suscritos a eventos de temporización e interrupción, y el otro asociado a paso de mensajes. Por ejemplo, hay asociado un total de tres `TSAMMessageHandler` para el puerto asociado a la interrupción. Gracias a las situaciones de tiempo real especificamos bajo qué eventos reacciona con un determinado `TSAMMessageHandler`, en este caso han sido demultiplexados en tres situaciones de tiempo real. Esto significa que se especifican tres `TEvent` que actúan sobre el mismo puerto, con tres reacciones distintas. Un caso opuesto se puede ver en el puerto asociado a un evento de temporización. En este caso sólo hay una reacción, un `TSAMMessageHandler`, el cual tiene asociadas dos situaciones de tiempo real. Éste indica que dos eventos, asociados al puerto de temporización, están asociados al mismo `TSAMMessageHandler`. Por último, las situaciones de tiempo real no tienen por qué estar asociadas sólo a puertos de evento. Vemos un puerto asociado a un protocolo de naturaleza síncrona, sus `TSAMMessageHandler` están etiquetados con situaciones de tiempo real, dependiendo del evento que disparó la primera reacción del sistema.

tiempo real. Por otro lado se integran desde una perspectiva de caja blanca. Ésta anota las reacciones válidas por cada uno de los componentes. No hay que olvidar que la recepción de un mismo mensaje tiene diferentes posibles reacciones que suceden dependiendo de los estados internos del componente. Las situaciones de tiempo real definen qué estados del componente, en definitiva, son válidos para

qué determinadas situaciones. Con ambas estrategias el modelo TSAM dispone de una estrategia de divide y vencerás para vertebrar la generación de modelos de análisis.

El modelado de las situaciones de tiempo real se hace de la siguiente forma. Cada componente puede definir más de un `TSAMMessageHandler` por tupla (puerto, mensajes). Sólo un `TSAMMessageHandler` es ejecutado a la llegada de un mensaje; esta selección dependerá del estado interno del componente, y las situaciones de tiempo real a nivel de sistema. Cada situación de tiempo real determina qué `TSAMMessageHandlers` puede ser ejecutado. Este hecho permite describir bajo qué condiciones y eventos un `TSAMMessageHandler` es ejecutado. En consecuencia, como se mencionó anteriormente, nuestro modelo se limita a una sola `TSAMMessageHandler` mediante los siguientes elementos: el puerto, el mensaje y la situación en tiempo real. En la imagen 3.9 mostramos un ejemplo de cómo se asocian las etiquetas de situaciones de tiempo real tanto a elementos `TEvent` como a elementos `TSAMMessageHandler`. Dependiendo del modelo de análisis estas etiquetas pueden ser interpretadas de diferentes maneras. Por ejemplo, como veremos en el capítulo 4, éstas permitirán ayudar a componer los flujos *end-to-end*, dependiendo de las situaciones de tiempo real que han sido configuradas. Dependiendo de las estrategias de análisis, estas etiquetas son configuradas de una determinada manera.

3.5.5. Analogías entre el modelo TSDM y el modelo TSAM

Los modelos TSDM y TSAM están basados en el paradigma CBSE. Esto significa que ambos modelos definen el sistema en términos de componentes, protocolos de comunicación y bibliotecas de servicio. Sus diferencias radican en matices como la descripción de las reacciones, el efecto del vector plataforma y la definición de las situaciones de tiempo real asociadas a los eventos externos.

La asociación al vector plataforma es una diferencia que está presente en todos los elementos del modelo TSAM. Este valor es interesante desde la perspectiva del análisis, ya que permite categorizar los elementos por plataforma y componer alternativas en función de estrategias de composición.

En los siguientes apartados vemos una comparativa entre los modelos equivalentes del modelo TSDM y del modelo TSAM.

Protocolos

El modelo que definen los protocolos de comunicación entre componentes en el modelo TSDM se denomina MCLEV Protocol. Su equivalente en el modelo

TSAM es el modelo TSAMProtocol. Ambos modelos definen la comunicación entre componentes en términos de mensajes que pueden enviar o recibir.

Sistema

El modelo que define la composición de componentes y su comunicación es el modelo MCLEV Flat Component System. Su equivalente es el modelo FLATMCAD. Ambos son similares. Por ejemplo, ninguno de los dos permiten definir jerarquías de estados. La única diferencia es la relativa a la definición de la plataforma para la cual el componente es válido. El modelo es el equivalente.

Comportamiento de los componentes

El modelo que define el comportamiento de cada componente en el modelo TSDM es el modelo *Flat Component Model* (FLATCMP), mientras que en el modelo TSAM es el modelo TSAMComponent. En este modelo las discrepancias son mayores, ya que el modelo TSAMComponent no define un comportamiento basado en máquinas de estado, sino en cadenas de acciones que se ejecutarán cuando el mensaje es recibido; incluso estas cadenas de reacciones ya no incluyen elementos condicionales, sino que se describen como posibles cadenas válidas para la reacción a un mismo mensaje. Estas reacciones deberán organizarse formando situaciones de tiempo real. En el caso del modelo FCM las cosas son un poco distintas. Las reacciones son descritas por un lado mediante una máquina de estados, donde cada uno de los elementos que la conforman están categorizados por tipos de transiciones, y por otro lado, en función de las acciones que serán ejecutadas por la recepción de un mensaje. Esta reacción puede tener diferentes elementos condicionales que tienen en cuenta estados internos del propio componente. Las reacciones en este modelo son expresadas en función de la naturaleza de los elementos y del código empotrado en este modelo. Ahora queda más claro que la generación de los elementos `MsgHandler` del modelo TSDM tiene como objetivo facilitar la transformación hacia modelos reactivos, los cuales son típicos del modelo de análisis. En nuestro caso, en particular, veremos el modelo MAST y el modelo PCM como modelos de análisis.

Biblioteca de servicio

El modelo que define las bibliotecas de servicio en el modelo MCLEV es el modelo *EDROOM Service Library* (EDROOMSL), mientras que en el modelo TSAM es el modelo TSAMSL. Este trabajo fue realizado de una manera colateral al trabajo de la presente Tesis Doctoral (Nilas, 2014). Las diferencias son

similares a las que hemos visto entre los modelos de componentes. En el caso TSDM se definen las interfaces de las librerías de servicio, y en el modelo TSAM estas interfaces contienen los AST con los tiempos de ejecución anotados.

3.5.6. Transformación desde el modelo TSDM al modelo TSAM

Al igual que la transformación desde los modelos EDROOM al modelo TSDM, también la transformación del modelo TSDM al modelo TSAM se divide en las siguientes fases:

- Transformación de los componentes.
- Generación de los protocolos de comunicación.
- Transformación de las bibliotecas de servicio.

En el cuadro 3.2 se muestra un resumen de las transformaciones.

Cuadro 3.2: Resumen de la transformación desde el modelo TSDM al modelo TSAM.

Elementos de TSDM.	Detalle TSDM.	Elemento de TSAM.
Componentes.	Componente: C_i.	Un componente en el modelo transaccional.
	Por cada `MsgHandler`.	Se crea de 1 a n `TSAMMessageHandler`.
	Para cada secuencia de `MsgHandler` de un mismo contexto.	Se agrupan en `TSAMMessageHandler` como `TSAMBasicHandler`.
	Por cada `TSAMMessageHandlerItem`.	Se crea un `TSAMMessageHandlerItem` de naturaleza equivalente.
	Por cada elemento de decisión Dc_i de un `MsgHandler`.	Se crea `TSAMMessageHandler`, que corresponde a un camino de ejecución.
Protocolos de comunicación.	Protocolo de comunicación: Pr_i.	Un protocolo de comunicación en el modelo TSAM.
	Por cada mensaje de entrada: $MsgI_i$ del tipo asíncrono en Pr_i.	Se genera un mensaje de entrada: $TSAMMsgI_i$ asíncrono en el modelo TSAM.
	Por cada $MsgI_i$ del tipo síncrono en Pr_i.	Se genera un $TSAMMsgI_i$ síncrono en el modelo TSAM.

continúa

continúa

Elementos de TSDM.	Detalle TSDM.	Elemento de TSAM.
	Por cada mensaje de salida: $MsgO_i$ del tipo asíncrono en Pr_i.	Se genera un mensaje de salida: $TSAMMsgO_i$ asíncrono en el modelo TSAM.
	Por cada $MsgO_i$ del tipo síncrono en Pr_i.	Se genera un $TSAMMsgO_i$ síncrono en el modelo TSAM.
	Por cada mensaje: Msg_i del tipo msgReply en Pr_i.	Se genera un mensaje: $TSAMMsg_i$ msgReply en el modelo TSAM.
Biblioteca de servicio.	Biblioteca de servicio: Li_i.	Una biblioteca de servicio: $TSAMLi_i$.
	Por cada Li_i.	Se crea un $TSAMLi_i$.
	Por cada función definida en la librería: Fx_i, que es parte de la interfaz pública.	Se crea un SAP_i.
	Por cada Fc_i que NO es parte de la interfaz pública.	Se crea una $Internalfunction$.
	Por cada Fc_i se analiza su AST con RVS.	Se crea el árbol de caminos de ejecución.
Anotación de parámetros de tiempos de ejecución.	Tiempos de ejecución: $ExTi_i$.	Se crea un $TACODE_i$.
	Por cada Cmp_i.	Se crea un $TACODE_i$.
	Por cada Li_i.	Se crea un $TACODE_i$.
	Por cada camino de ejecución: $ExecPath_i$ de un `TSAMMessageHandlerItem` definido en un Cmp_i.	Se define una entrada con sus $ExTi_i$.
	Por cada camino de ejecución: $ExecPath_i$ de un Fx_i asociada a la Li_i.	Se definen una entrada con sus $ExTi_i$ en el $TACODE_i$.
Definición de eventos patrones de activación.	Patrones de activación PAc_i.	Se definen en un $TEvent_i$ en el modelo TSAMRTRequirement.
	Por cada PAc_i.	Se genera un $TSAMPAc_i$ Se especifica el $TSAMCmp_i$, $TSAMPrt_i$ y $TSAMMsg_i$.
	Por cada PAc_i del tipo periódico.	Se genera un $TSAMPAc_i$ del tipo periódico, se tiene que especificar el periodo.
	Por cada PAc_i del tipo aperiódico.	Se genera un $TSAMPAc_i$ del tipo esporádico, se tiene que especificar el menor tiempo de llegada.

continúa

continúa

Elementos de TSDM.	Detalle TSDM.	Elemento de TSAM.
	Por cada PAc_i del tipo ráfaga (*bursty*).	Se genera un $TSAMPAc_i$ del tipo ráfaga, se especifica el menor tiempo de suceso de la interrupción, el número máximo de sucesos de la ráfaga, y el menor tiempo entre cada suceso.
Definición de los *deadlines*.	*deadlines*: D_i.	Se definen en un $TSAMD_i$ en el modelo TSAMRTRequirement.
	Por cada D_i.	Se genera un $TSAMD_i$, donde se tiene que especificar un `TEvent` y su `TSAMMessageHandlerItems`.
Definición de parámetros de tiempos de ejecución dependientes de la plataforma.	tiempos de ejecución dependientes de la plataforma: Plt_i.	Generación del modelo TSAMPlatform.
	Procesador.	Se anota la latencia de gestión de interrupciones, rango de prioridades y temporizador asociado.
	Planificador.	Se anota: ▪ Latencia cambio de contexto. ▪ Rango de prioridades.
	`Bottom-half`.	Número de hilos.
	mutex.	Se anota: ▪ Latencias de captura. ▪ Latencias liberación.
	Temporizadores.	Se anotan: ▪ Latencias de gestión. ▪ Sincronización local.

continúa

continúa

Elementos de TSDM.	Detalle TSDM.	Elemento de TSAM.
	Temporizadores.	Se anotan: ▪ Latencias de gestión. ▪ Sincronización local. ▪ Precisión. ▪ `TEvent` asociados.

3.5.7. Transformación de los componentes

La generación del modelo requiere diferentes transformaciones de diferentes modelos fuente. La generación de toda la información relativa a los elementos `TSAMMessageHandler`, `TSAMBasicHandler` y los `TSAMMessageHandlerItems` se extrae de una transformación desde el modelo FLATCMP. La definición de los caminos de ejecución, por otro lado, se extrae desde otra transformación. Ambas generan el modelo definitivo del TSAMComponent.

La transformación desde el modelo FLATCMP al modelo TSAMComponent define los `TSAMMessageHandler` junto con el resto de sus elementos relativos, como son los puertos de comunicación. Por cada elemento `MsgHandlers` su equivalente es el `TSAMMessageHandler`. Los elementos tanto `MsgHandlerWithTrigger` como los `MsgHandlerWithOutTrigger` se transforman en `TSAMBasicHandler`. Los `TSAMBasicHandler` equivalen a los diferentes contextos de las máquinas de estados del modelo EDROOM. Cada `MsgHandler` tiene asociado un conjunto de `MsgHandlerItem`, los cuales tienen un equivalente en el modelo TSAM. Se muestra en la siguiente lista:

- $MsgDataHandler \Leftrightarrow$ `TSAMMHiDataHandler`.
- $Action \Leftrightarrow$ `TSAMMHiAction`.
- $Send \Leftrightarrow$ `TSAMMHiSend`.
- $Invoke \Leftrightarrow$ `TSAMMHiInvoke`.
- $MsgReply \Leftrightarrow$ `TSAMMHiReply`.

La transformación desde el análisis del AST con la herramienta RVS genera los caminos de ejecución del modelo TSAMComponent. Una de las opciones de esta herramienta es realizar análisis del AST de código en C, C++ y Ada. La

transformación que genera el AST fue un trabajo paralelo de la Tesis Doctoral (Nilas, 2014). Por cada componente desplegado se analiza el código relativo a cada `MsgHandlerItem`. Como sabemos, cada `MsgHandlerItem` tiene su relativo `TSAMMessageHandlerItem`. Esto facilita la transformación.

3.5.8. Protocolos de comunicación

La transformación se realiza desde el modelo MCLEV Protocol al modelo TSAMProtocol. Como hemos visto en la sección 3.5.5, ambos modelos son equivalentes semánticamente y sólo los diferencian los elementos sintácticos, cambiando únicamente la definición de las palabras reservadas. Los elementos que se generan en el modelo TSAMProtocol en la transformación son tanto la descripción de los mensajes como su naturaleza.

3.5.9. Biblioteca de servicio

La transformación se realiza desde el modelo MCLEV Library al modelo TSAMSL.

La transformación es similar a la realizada en los componentes, la cual fue explicada en la sección 3.5.7. Es similar ya que la estructura de la librería se extrae del modelo TSDM correspondiente, pero hay parte de la información que tiene que ser generada mediante otra transformación. Esta transformación es la que genera los caminos de ejecución. Como hemos visto anteriormente en las analogías entre estos modelos, el modelo TSAMSL especifica cuáles son las funciones que hacen de interfaz denominadas como SAP, pero el caso del modelo MCLEV Library no especifica estas funciones. Por este motivo, éstas tienen que ser descritas en el proceso. La manera de identificarlas es gracias a la transformación relativa al AST mediante la herramienta RVS. La herramienta RVS identifica todas las funciones que pertenecen a la librería. Como parámetro de esta transformación el usuario deberá especificar aquellas funciones que forman parte de la interfaz pública y cuáles no.

Para ello cada función será del tipo *internal* o *provided*. La transformación desde el modelo TSDM Library genera de una manera automática las dependencias entre otras librerías del modelo TSAM. La relación de las transformaciones es uno a uno. Esto implica que las dependencias descritas en el modelo MCLEV Library serán equivalentes en el modelo TSAMSL.

3.5.10. Transactional Analisys Timming Code

El modelo TACode se anota con los tiempos de ejecución de cada uno de los caminos de ejecución. Éstos están descritos en el contexto de cada `TSAMMessageHandlerItem` y de cada función asociada a una TSAMSL. La anotación de este modelo no se realiza desde el modelo diseño TSDM, ya que no contiene la información de una manera explícita. Se tiene que caracterizar el tiempo de ejecución del código del componente o librería de servicio. En un trabajo ortogonal a esta Tesis Doctoral se realizó una transformación automática (Nilas, 2014). Una vez usada la herramienta RVS para caracterizar el componente, la base de datos con los tiempos de ejecución se transforma a un modelo TACode.

Los tiempos de ejecución deben ser anotados por cada uno de los caminos de ejecución.

3.5.11. Transactional System Analysis Real-Time Requirement Model

Los patrones de activación, junto con las restricciones temporales, deben ser definidas para poder realizar análisis de planificabilidad y de rendimiento. Esta información debe ser anotada por un experto de dominio, quien debe valorar y conocer las diferentes situaciones en las que el sistema tiene que trabajar. En concreto, debe especificar cada uno de los eventos de naturaleza de interrupción o temporización, especificando la naturaleza del evento con uno de los patrones de activaciones que se definen en este modelo, como son periódico, esporádico y de ráfaga. Además, el experto de dominio extraerá de los requisitos del sistema las restricciones temporales.

3.5.12. Transactional Platform Configuration Model

El modelo TSAMPlatform no puede ser transformado desde el modelo TSDM. Este modelo tiene que ser definido mediante otros modelos, en particular debe ser tarea del desarrollador de la plataforma. No hemos propuesto una transformación en la versión presentada en esta Tesis Doctoral. Actualmente se anota manualmente. La anotación recomendada de sus parámetros debe realizarse mediante el uso de plataformas ya calificadas, como es el caso de sistemas operativos, como la versión de `EdiSoft` de RTEMS o placas calificadas de algunas versiones comerciales, como son el caso de las placas de `Aeroflex` Cobham Gaisler (2015). Cada uno de los parámetros deberá ser anotado con WCET.

3.5.13. Situaciones de tiempo real

Estas situaciones permiten vertebrar estrategias de análisis. Su anotación es manual, ya que la información del modelo diseño no la contiene, al menos de una manera explícita. Con ella podemos dividir las reacciones del sistema dependiendo de las situaciones a las que se enfrenta. Por ejemplo, en sistemas empotrados es común tener situaciones nominales, de configuración del sistema, o sistema degradado. Todos estos casos suelen tener comportamientos exclusivos que sólo se dan durante ese estado. Con las situaciones de tiempo real podemos generar modelos de análisis exclusivos de conjuntos de situaciones de tiempo real. Además se pueden especificar patrones de composiciones de reacciones a eventos por parte del sistema. Estas estrategias permiten definir qué reacciones son compatibles o no dependiendo de los propios patrones de activación. En el siguiente capítulo veremos cómo las situaciones de tiempo real son usadas para definir reacciones en modo ráfaga, típicas en la gestión de telecomandos, y cómo se pueden definir situaciones que optimizan las reacciones que contienen gestión de errores.

En las subsecciones anteriores nos hemos centrado en cada uno de los modelos concretos del TSAM, definiendo la transformación o una definición manual. En este caso las situaciones de tiempo real se definen en varios modelos, pero lo tratamos como un elemento independiente. Los modelos donde se definen son TSAMComponent y el TSAMRTRequirement. En el modelo TSAMComponent el experto de dominio etiquetará con una o varias situaciones de tiempo real los `TSAMMessageHandler`. En el modelo TSAMRTRequirement se etiqueta mediante cada evento con una o varias de las situaciones válidas para ese evento `TEvent`.

3.5.14. Resumen

En esta sección se mostró el modelo TSAM, junto con la transformación desde el modelo TSDM al modelo TSAM. En donde el modelo TSAM es:

- Un modelo orientado al análisis basado en el paradigma CBSE.
- Un modelo que describe las reacciones de cada componente como una secuencia de acciones de naturaleza explícita mediante los `TSAMMessageHandlerItem`.
- Un modelo que describe por cada función de una manera explícita, ya sea función asociada a librería o embebida en las reacciones de los componentes.

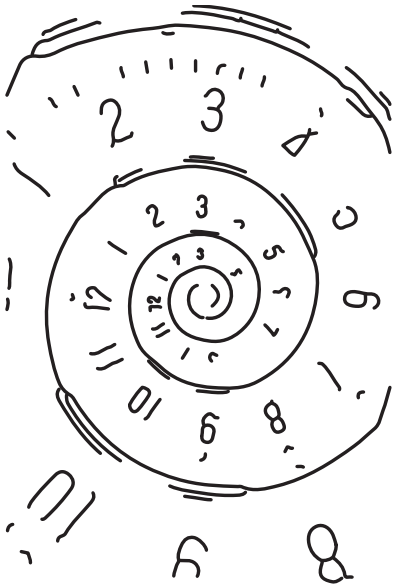
- Un modelo donde cada rama AST es anotada con las propiedades EFP.

Hemos podido ver cómo de cercana es la distancia entre los modelos TSDM y el modelo TSAM, donde todos los elementos son equivalentes menos en la relación entre el código y las anotaciones. Desde los modelos TSDM se generan los modelos TSAM, y desde el código embebido en los componentes y que conforman las librerías es analizado y transformado en sus correspondientes AST, una vez que han sido generados todos estos modelos.

En el caso del modelo TSAM, éste utiliza la infraestructura para el modelado de las EFP.

Capítulo 4

Modelo de análisis de planificabilidad



El hijo de Laertes que habita en Ítaca. Lo vi en una isla derramando abundante llanto, en el palacio de la ninfa Calipso, que lo retiene por la fuerza. No puede regresar a su tierra, pues no tiene naves provistas de remos ni compañeros que lo acompañen por el ancho lomo del mar

H. Homero

Controlar la complejidad es la esencia de la programación

B. Kernigan

4.1. Introducción

En este capítulo abordamos la integración de un modelo de análisis de planificabilidad dentro del proceso propuesto en esta Tesis Doctoral. En la imagen

4.1 mostramos un típico proceso en V donde se aprecian los elementos que discutimos en este capítulo. Vemos que a partir del modelo transaccional, en particular el modelo TSAM, se genera el modelo de análisis MAST. El modelo y la transformación que se define en este capítulo corresponden a una fase de validación de sistemas, en particular generamos un modelo que permite poder validar las restricciones temporales.

En la imagen 4.2 mostramos los modelos y las transformaciones tratados en este capítulo. Vemos que la transformación parte de un modelo orientado al análisis TSAM, generando el modelo MAST. Este modelo TSAM tiene que tener todos sus elementos anotados, como es el caso de tiempos asociados a las respuestas de los componentes, librerías de servicio y tiempos asociados a la plataforma.

El modelo de análisis de planificabilidad es uno de los objetivos marcados para la reducción de costes propuesto en esta Tesis Doctoral. Gracias a la transformación automática se genera un resultado exigido en el estándar ECSS-E-ST-40C. En concreto, el objetivo que se cubre es:

- «Generación del modelo de planificabilidad en conformidad con el estándar ECSS-E-ST-40C».

Ambos modelos, el modelo TSAM y el modelo MAST, son modelos cercanos. Esta cercanía es patente en la definición de las reacciones del sistema frente a eventos mediante transacciones del tipo *run-to-completion*. Este tipo de transacciones una vez activadas deben de ser finalizadas.

Este capítulo se divide en las siguientes secciones. En la sección 4.2 hacemos una introducción a la herramienta MAST. En la sección 4.3 se detallan las transformaciones desde el modelo transaccional al modelo MAST. Por último, en la sección 4.4 describimos estrategias de situaciones de tiempo real que permiten que algunas situaciones en los modelos transaccionales sean analizables por la herramienta MAST.

4.2. MAST

4.2.1. Introducción

MAST es una herramienta de código abierto que permite caracterizar análisis de tiempos de sistemas de tiempo real. En concreto, la herramienta realiza los siguientes análisis:

- Análisis de planificabilidad, ¿se cumplen las restricciones temporales?

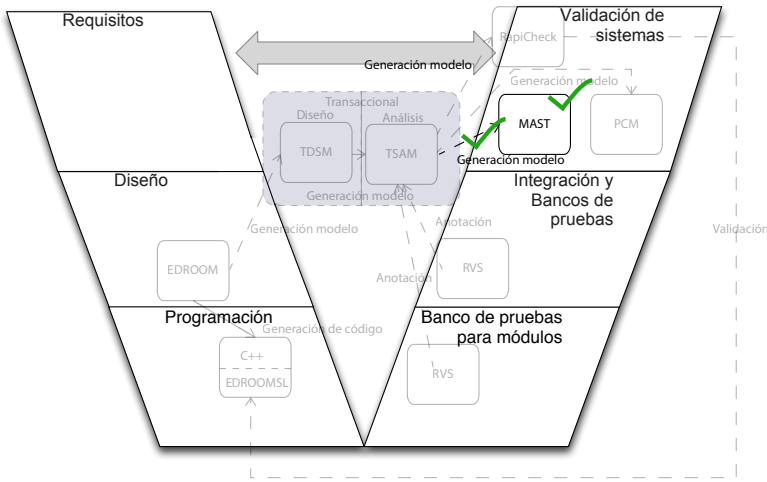


Figura 4.1: Esquema del proceso y la cadena de transformaciones vistas en este capítulo. Las transformaciones y modelos descritos en la figura permiten la generación automática desde el modelo TSAM al modelo MAST.

- Cálculo del tiempo de bloqueo, ¿cuál es el tiempo de bloqueo que produce una determinada tarea o recurso compartido?
- Análisis de sensibilidad, ¿cuánto tiempo de margen tenemos hasta sobrepasar la restricción temporal?
- Asignación óptima de prioridades, ¿cuál es la configuración de prioridades para las tareas que satisfacen las restricciones temporales?
- Simulación, cálculo de la dispersión de los tiempos de respuesta.

MAST es una herramienta compatible con los estándares más comunes usados en la industria de software de tiempo real. Permite modelar el comportamiento de la *Embedded Development Platform* (EDP) del tiempo para plataformas que están basadas en Ada, *Portable Operating System Interface* (POSIX) e *Integrated Modular Avionics* (IMA), entre otras muchas.

MAST permite describir escenarios complejos como es el caso de sistemas distribuidos. Describe sus nodos de procesamiento y la topología de comunicación. Así se pueden realizar análisis de utilización e identificación de los cuellos

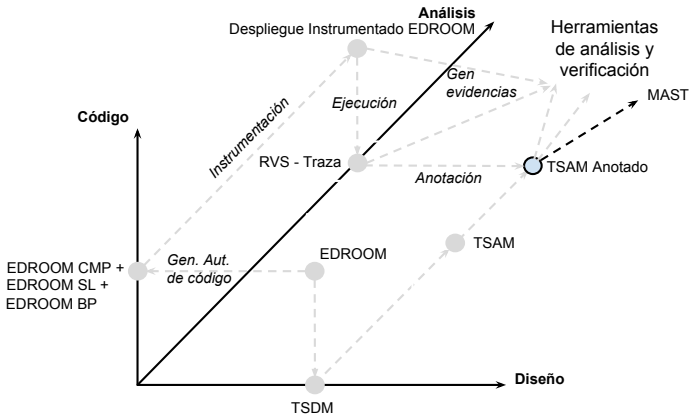


Figura 4.2: En este esquema vemos en qué punto de la transformación y sus coordenadas estamos. En este capítulo sólo tratamos con modelos enmarcados en el eje de análisis, en particular, con el modelo TSAM y el modelo MAST.

de botella. Esta utilización se describe en función del porcentaje de tiempo que permite al sistema ser planificable, lo que posibilita identificar la flexibilidad para aumentar la carga de procesamiento en nodos y puntos de la red.

Las herramientas provistas por MAST permiten realizar análisis de alternativas de diseño. Se pueden describir conjuntos de usos de sistema (*Workload*), de los cuales podrán ser comparados sus resultados y elegir el más adecuado en función de sus restricciones temporales y su rendimiento. Incluso pueden compararse diferentes modelos de diseño identificando sus cuellos de botella, y su flexibilidad a incrementos de carga de computación.

MAST es una herramienta que ha tenido un largo recorrido. Ha sido usada con éxito en un gran número de proyectos. Proyectos en diferentes ramas de los sistemas empotrados, como son el caso de telecomunicación, aviación, automoción y espacio. Algunos de los proyectos más destacados son ASSERT(ESA, 2008), Ocarina, Chess(Montecchi y Lollini, 2011), TASTE(Perrotin et al., 2012).

En la última versión de MAST 2.0 ha sido incluida una descripción de modelos de entrada y de salida mediante MDE, específicamente en la plataforma EMF. Esta estrategia permitió con éxito al CTR generar una cadena de modelos anotados desde herramientas que usan el estándar UML-MARTE al modelo

MAST. De esta manera, desde un modelo de diseño generan modelos MAST equivalentes.

Los diferentes análisis más destacados del modelo MAST 2.0 son:

- Análisis de planificabilidad usando estrategias de *Offset*, los cuales están basados en sistemas *Response Time Analysis* (RTA) con prioridades fijas.
- Análisis de planificabilidad holístico para sistemas RTA con prioridades fijas y *Earliest Deadline First* (EDF).
- Análisis de planificabilidad clásico RTA (sistemas monoprocesador y prioridades fijas).
- Análisis de planificación EDF con asignación de prioridades RTA para planificadores jerárquicos.
- Análisis de planificación de planificadores jerárquicos basado en prioridades fijas y RTA.
- Análisis de asignación de prioridades para RTA.

4.2.2. Modelo MAST 2.0

En esta sección realizamos una descripción del modelo de entrada, descripción del modelo de análisis, y el modelo salida, modelo de resultados de MAST. En concreto el relativo a la versión 2.0. El modelo se divide en cinco elementos, los cuales describimos a continuación:

- **Recursos de procesamiento**, definición de la plataforma. Los elementos que pueden ser definidos son: *Central Process Unit* (CPU) y temporizadores.
- **Recursos compartidos**, definición de accesos en exclusión mutua. Éstos son configurados con diferentes protocolos de accesos a recursos compartidos.
- **Recursos de planificación**, definición de los parámetros de planificación. Por cada planificador se deberá especificar su política de planificación (prioridades fijas, EDF, etc.) junto con el rango de prioridades. Además también se especifican los parámetros relativos a las latencias que se introducen en el sistema.

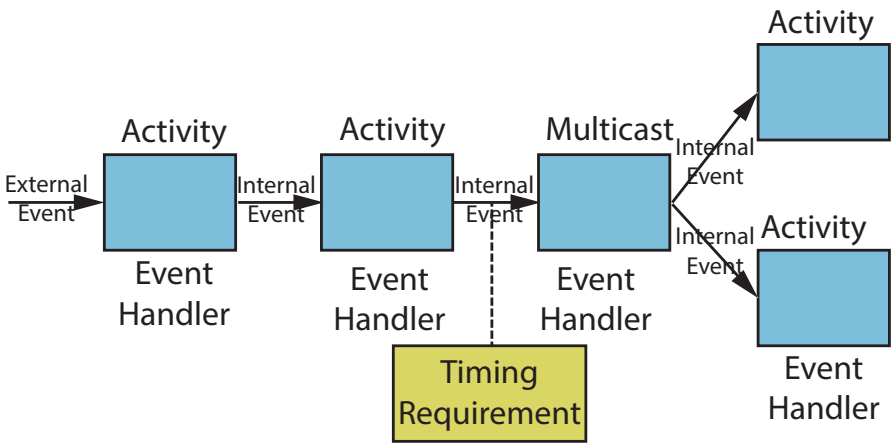


Figura 4.3: Transacción del modelo MAST. Se muestran los elementos típicos que forman parte de una transacción en MAST. Una transacción está formada por `EventHandlers` e `InternalEvents`. Los `EventHandlers` pueden ser de diferente naturaleza. En la figura hemos especificado dos, los cuales son: `Activity` y `Multicast`. El primero permite especificar operaciones que se ejecutan en la transacción, el segundo permite modificar el flujo de la transacción. Los `InternalEvent` enlazan los `EventHandler`. Éstos, además, pueden tener asociados requisitos temporales en forma de `TimingRequirement`. El evento `ExternalEvent` es el encargado de especificar el patrón de activación.

- **Operaciones**, definición de las operaciones que son ejecutadas en el sistema. Se definen accesos a zonas de exclusión mutua y sus perfiles de tiempos de ejecución.
- **Transacciones**, definición de los eventos que interactúan con el sistema. Esta interacción del evento se define en función de cadenas de operaciones, a las cuales se les asocian sus restricciones temporales. Se muestra un ejemplo de transacción en la figura 4.3.

Para describir de una manera más amena y detallada cada uno de los elementos que hemos mencionado anteriormente, hemos definido un ejemplo mínimo. El ejemplo, consta del caso típico productor/consumidor. El escenario son dos tareas y un recurso compartido. Sus nombre respectivos son:

- Productor P.
- Consumidor C.
- Recurso compartido RS.

El funcionamiento del ejemplo mínimo será el siguiente:

- La tarea P produce un telecomando de una manera esporádica. Cada vez que se recibe un telecomando externo, éste se procesa y se envía a la tarea C para que sea ejecutado. El menor tiempo de llegada del telecomando es de 1 segundo.
- La tarea P envía un telecomando al recurso compartido RC.
- La tarea P envía un mensaje a la tarea C comunicándole que hay un telecomando que consumir.
- La tarea C consume un telecomando del recurso compartido RC.

La plataforma sobre la que se ejecutará el sistema es un sistema monoprocesador, del cual tendremos en cuenta los posibles *jitter* y tiempos de ejecución.

En las siguientes secciones recorreremos cada elemento del modelo MAST del sistema que hemos descrito.

Recursos de procesamiento

En nuestro caso la plataforma sobre la cual se ejecutará el software planteado es un monoprocesador, como ya vimos. Los parámetros que tienen que ser anotados en el modelo MAST son los siguientes:

- **Tiempo de la gestión del *Interrupt Service Routine* (ISR)**, en nuestro caso configuramos el parámetro en 1 *milisegundo*.
- **Rango de prioridades válido**, en este caso configuramos el rango de prioridades entre 101 y 117 prioridades hardware —similar a sistemas como `Leon`—.
- **Temporizadores del sistema**, en este caso no configuramos ninguna clase de temporizador.

Recursos compartidos

Como hemos visto en el enunciado del ejemplo que hemos propuesto tiene que ser definido un recurso compartido. Se define un recurso compartido con un protocolo *Priority Inheritance Mutex*, como hemos visto, éste se denominará `RC`.

Recursos de planificación

El planificador que vamos a usar en este ejemplo es de prioridades fijas, donde el rango será de 1 a 100 prioridades. Además se deberá especificar el peor cambio de contexto, el cual configuramos a 2 ms.

Operaciones

Se definen cuatro operaciones:

- **Recepción telecomando**, ésta es la encargada de gestionar un telecomando. Configuramos su WCET a 100 ms.
- **Depositar telecomando**, se encarga de depositar un telecomando en el recurso compartido RC. Configuramos su WCET a 50 ms. Además, se accede al recurso compartido RC, accediéndose y liberándose en la operación.
- **Capturar telecomando**, se encarga de recolectar un telecomando del recurso compartido RC. Configuramos su WCET a 50 ms. Configuramos su WCET a 50 ms. Además, se accede al recurso compartido RC, accediéndose y liberándose en la operación.
- **Ejecución telecomando**, se encarga de ejecutar el telecomando. Configuramos su WCET a 300 ms.

Transacciones

Hemos definido una transición que gestiona los telecomandos. El orden es el siguiente, la tarea P recibe un telecomando, éste es depositado en el recurso compartido RC, el cual está protegido por un *mutex*. Una vez que ha sido depositado el telecomando, la tarea C despierta, recoge el telecomando y lo ejecuta. También hemos definido una restricción temporal. La restricción está

asociada al evento interno *Final*. Ésta se encuentra configurada a 700 ms. En la imagen 4.4 mostramos la transacción resultante.

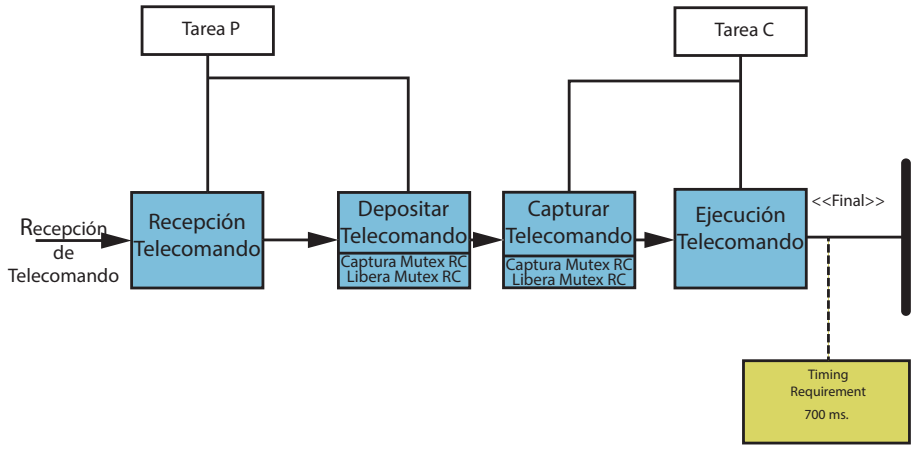


Figura 4.4: Transacción MAST del modelo de ejemplo propuesto.

Resultados de la herramienta MAST

Los resultados que presentamos son los relativos al análisis de planificabilidad, donde los resultados que obtenemos corresponden a una asignación de prioridades óptima, los tiempos de respuesta de las transacciones y a la susceptibilidad de los tiempos de respuesta.

Prioridades válidas para las tareas

Las prioridades válidas para las tareas que hemos definido en el ejemplo son:

- Tarea P [1-100].
- Tarea C [1-100].

Tiempos de respuesta

El tiempo de respuesta para la única transacción es de 506 ms. Esto está por debajo del *deadline* —-el cual es 700 ms— por lo que podemos afirmar que

las restricciones del sistema que hemos planteado cumple con las restricciones de tiempo.

Susceptibilidad de los tiempos de respuesta

Para concluir el ejemplo comprobaremos la sensibilidad de los tiempos de respuesta. La susceptibilidad de la única transacción definida en el sistema es del 39,84 %. Esto significa que podemos aumentar en ese porcentaje el tiempo de respuesta de la transacción sin que violemos la restricción temporal de la misma.

4.3. Transformación desde TSAM a MAST

La transformación desde el modelo TSAM al modelo MAST contiene varias perspectivas:

Cuadro 4.1: Resumen de la transformación desde TSAM a MAST.

Categorías de elementos generado.	Elemento TSAM.	Elemento MAST.
Recursos de procesamiento.	Recurso procesamiento: RP_i.	Se generan todos los parámetros asociados al RP_i, la transformación sólo define monoprocesador.
	Por cada `TMTicker`.	Se genera un `Ticker`, se especifican sobrecarga y periodo.
	Por cada `TMAlarmClock`.	Se genera un `One-shot`, se especifica la sobrecarga.
	Sólo hay un único RP_i.	Se genera un `ProcessingResource` del tipo `RegularProcessor`, se especifica el `Timer` asociado, `ISRSwitch`, rango de prioridades de interrupción.
Planificador.	Planificador: ScH_i.	Se definen todos los elementos asociados al ScH_i.
	La clase `TMPrimaryScheduler`.	Genera la `PrimaryScheduler`, se asocia el `ProcessingResource`, los parámetros que se transforman son: rango de prioridades y el cambio de contexto, sólo se define un único planificador.

continúa

continúa

Categorías de elemento a generado.	Elemento TSAM.	Elemento MAST.
Recursos compartidos.	Recurso Compartido: RC_i.	Se define un `SharedResource`.
	Si está configurado el protocolo `TMMutexCeiling`, por cada `TSAMComponent` del tipo proactivo o reactivo.	Se genera un `InmediateCeilingMutex`.
	Si está configurado el protocolo `TMMutexCeiling`, por cada `TSAMComponent` del tipo proactivo o reactivo.	`MutexInheritanceProtocol`.
Recurso de planificación.	Recurso Planificación: RPS_i.	Por cada `TSAMComponent` del tipo proactivo o reactivo se genera un RPS_i.
	`TSAMComponent` Proactivo.	Se genera un `ThreadScheduler`.
	`TSAMComponent` Reactivo.	Se genera un `ThreadScheduler`.
Operaciones.	Operaciones: Op_i.	Por cada `TSAMMessageHandlerItem` se genera una Op_i.
	Por cada `TSAMMHiDataHandler`.	Se genera una `SimpleOperation`.
	Por cada `TSAMMHiAction`.	Se genera una `SimpleOperation`.
	Por cada `TSAMMHiSend`.	Se genera una `SimpleOperation`.
	Por cada `TSAMMHiInvoke`.	Se genera una `SimpleOperation`.
	Por cada `TSAMMHiReply`.	Se genera una `SimpleOperation`.
	Por cada `TSAMAsynchMsgHandler`.	Se genera un conjunto de `CompositeOperation`: ▪ Formados por las `SimpleOperation` relativas a las `TSAMMessageHandlerItem`. ▪ Dos `SimpleOperations` que contienen un SR_i relativo al `TSAMComponent` al que pertenece el `TSAMAsynchMsgHandler`.

continúa

continúa

Categorías de elemento a generado.	Elemento TSAM.	Elemento MAST.
	Por cada `TSAMSynchMsgHandler`.	Se genera una `CompositeOperation` formada por: ▪ Las `SimpleOperations` relativas a los `TSAMMessageHandlerItems`. ▪ Por dos `SimpleOperations` que contienen un SR_i relativo al `TSAMComponent` al que pertenece el `TSAMAsynchMsgHandler`.
Transacciones.	Transacción: Tr_i.	Por cada `TEvent` se genera una Tr_i.
	Por cada evento `TEvent` del tipo `TBusrtyEventPatter`.	Se genera `BurstyEvent`.
	Por cada evento `TEvent` del tipo `TSporadicEventPatter`.	Se genera `SporadicEvent`.
	Por cada evento `TEvent` del tipo `TPeriodicEventPattern`.	Se genera `PeriodicEvent`.
	Se navega partiendo de cada `TEvent` por cada `TSAMMHiSend` y por cada `TSAMMHiInvoke`.	Se forma una transacción nueva.
	Por cada `TSAMMHiSend` y por cada `TSAMMHiInvoke`.	Se define un `InternalEvent` del tipo `Fork`.
	Por cada `TDeadline`.	Se genera un `HardDeadline` sobre el `InternalEvent` específico.

- Definición de la plataforma con sus propiedades EFP (*switch context time*, *interrupt service request*, *release* y *adquire mutex*, etc.), políticas de planificación y protocolos de inversión de prioridad. Estos parámetros se definen en TSAM en el modelo `TSAMPlatform`.
- Definición de las operaciones, las operaciones MAST se crean a partir de los elementos que describen las reacciones en en los `TSAMComponent` `TSAMMessageHandlerItem` y de las funciones asociadas en las librerías `TSAMSAP`.
- Definición de los eventos, transacciones y restricciones temporales. Los patrones de activación asociados al evento en el modelo TSAM se deta-

llan en el modelo `TSAMRTRequirement` y la transacción se compone gracias a los modelos `TSAMComponent` y `TSAMSL`, que define la reacción frente al evento. Las cadenas de `TSAMMessageHandlerItem` y llamadas `TSAMSAP` se transforman en una transacción con sus operaciones equivalentes del modelo MAST. Las restricciones temporales se definen en el modelo `TSAMRTRequirement`. Se describen mediante un `TSAMMessageHandlerItem` y la información relativa a la restricción.

Para generar todas las perspectivas hemos dividido la transformación en seis etapas. Éstas se muestran en la figura 4.5. Cada una de las etapas está dedicada a generar un elemento específico del modelo de MAST resultante. La transformación se ha organizado empleando un archivo QVTO para cada paso. Todas ellas toman el modelo TSAM como entrada. Estas etapas son explicadas en las siguientes secciones. Un resumen de la transformación se muestra en el cuadro 4.3.

4.3.1. Recursos de procesamiento

La primera transformación es la relativa a la generación del modelo de MAST del hardware. En nuestro caso, definimos el procesador y el hardware que dan soporte a los servicios de temporización. En concreto, este modelo incluye la información relativa a la frecuencia del microprocesador, el tiempo de latencia de interrupción, el rango de prioridad de las interrupciones y la caracterización de la frecuencia del reloj monotónico de la plataforma. Además, incluye una abstracción para modelar el hardware que da soporte a la programación de eventos en el tiempo, que puede ser de tipo alarma (*ticker*) o *one-shot*. Los datos de entrada se toman del modelo de descripción de la plataforma (`TSAMPlatform`) incluida en el TSAM.

4.3.2. Planificador

En la siguiente etapa de transformación se genera el planificador del modelo MAST. Tienen que ser especificados varios parámetros, tales como el tiempo de latencia de cambio de contexto o el rango de prioridad de la tarea. Al igual que en la primera transformación, la información de entrada se extrae del modelo TSAM, en concreto el modelo `TSAMPlatform`.

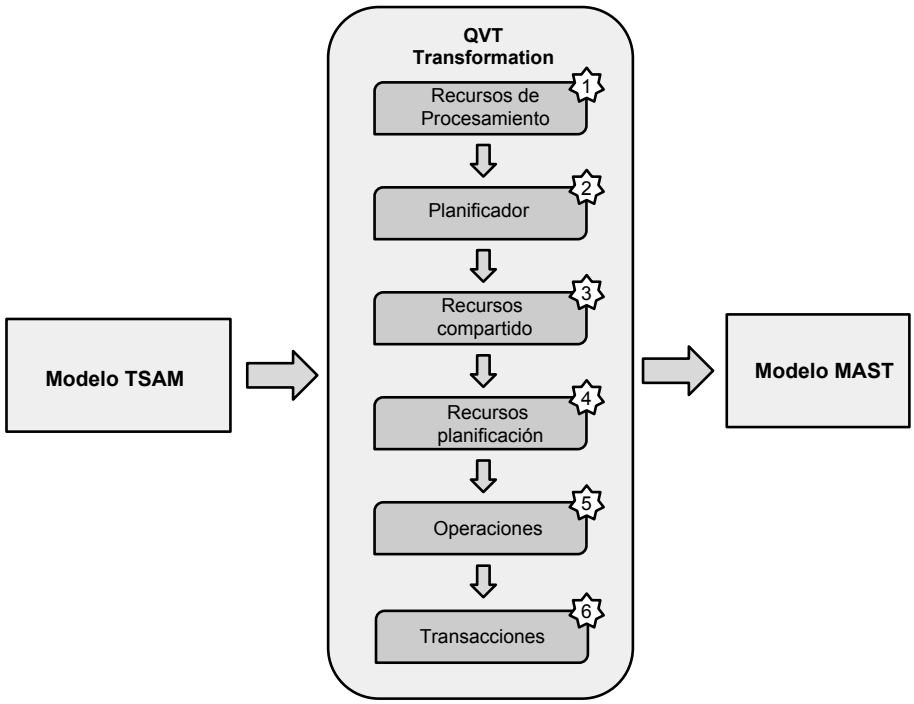


Figura 4.5: Transformación desde el modelo TSAM al MAST. Ésta consta de un total de 6 pasos que corresponden a cada uno de los elementos que componen el modelo MAST.

Recursos compartidos

En esta etapa transformamos los recursos compartidos MAST. La transformación crea un recurso compartido por cada instancia de componente. Si el componente es del tipo proactivo o reactivo, se crea un recurso compartido del modelo MAST que tiene la misión de modelar el acceso a la cola de mensajes del componente —de tal manera que los mensajes se consumen de una manera secuencial—. Por otro lado, si la instancia es de un componente del tipo recurso compartido —recurso compartido definido en el modelo de componentes del modelo transaccional—, el elemento recurso compartido que se genera tiene como objetivo modelar el acceso al propio componente. Los recursos comparti-

dos generados deben estar protegidos con uno de los protocolos de acceso que evitan fenómenos relativos a la inversión de prioridad. Ejemplos de estos protocolos son los del tipo techo prioridad inmediato (Goodenough y Sha, 1988) o los protocolos de herencia de prioridad (Sha et al., 1990). Esta información se extrae de los modelos de descripción de plataforma.

Recursos de planificación

En esta etapa de transformación, se definen los servidores de planificación *scheduling server* del modelo MAST. Por cada instancia de componente, ya sea proactiva o reactiva, se crea un servidor de planificación para modelar la tarea subyacente que implementa su funcionalidad. Además, la tarea que ejecuta los elementos `Bottom-halves` asociados a las interrupciones y eventos de temporización también se modelan con esta abstracción. Dependiendo de la configuración del *run-time component* de EDROOM, la gestión de los `Bottom-halves` puede gestionarse bien con una tarea para todos los `Bottom-halves` o con una tarea para cada uno de los `Bottom-halves`. En el primer caso, un único servidor de planificación es creado mientras que para la segunda opción habrá una tarea por cada `Bottom-half`. La información sobre la configuración del *run-time*, así como la prioridad de los diferentes eventos, se extrae de los modelos `TSAMRTRequirement` y del modelo `TSAMPlatform`.

Operaciones

Tras la generación de los recursos de planificación *schedulable resources*, generaremos las operaciones MAST. Estas abstracciones se usan para modelar las demandas de recursos computacionales en términos de la cantidad de tiempo de ejecución que consumen. El modelo MAST tiene capacidad de definir las operaciones de una manera agregada jerárquicamente. Éstas son llamadas operaciones compuestas (*composite operations*), que se definen como un conjunto ordenado de operaciones simples o de otras operaciones compuestas. Los manejadores de mensajes asíncronos de componentes proactivos y reactivos se modelan como una composición de operaciones compuestas y simples. Su transformación es más compleja que las anteriores, de tal manera que, por claridad, descomponemos su transformación en los siguientes pasos:

- En primer lugar, se crean dos operaciones simples que modelan el bloqueo y desbloqueo, respectivamente, del recurso compartido correspondiente al

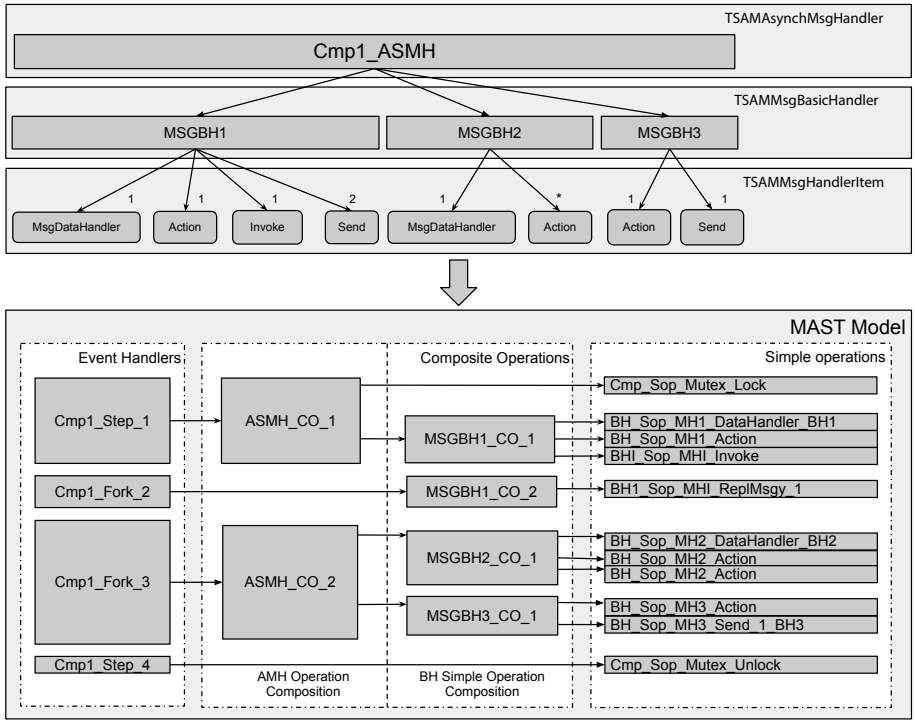


Figura 4.6: Transformación de las operaciones desde el modelo TSAM al modelo MAST.

componente que define el manejador de mensajes asíncrono. Estas operaciones están situadas al principio y al final de toda la cadena de operaciones que forman la operación compuesta. Los tiempos de ejecución asignados a estas operaciones se estiman y especifican en el modelo de configuración de la plataforma `TSAMPlatform` de MAST a nivel global.

- Un manejador de mensajes asíncrono `TSAMAsynchMsgHandler` se descompone en una secuencia de manejadores básicos (`TSAMBasicHandler`) que son, a su vez, formadas por una secuencia de elementos `TSAMMessageHandlerItem`. En esta parte del proceso de transformación, se genera una operación simple por cada uno de estos elementos `TSAMMessageHandlerItems`. La cantidad de tiempo de procesamiento asignado a las operaciones

simples es el WCET, anotada en el modelo TACode relativo al `TSAMMessageHandlerItem` ,[1] y el WCET asociado a las demandas de cada uno de los SAP realizadas por `TSAMMessageHandlerItem`, entonces el WCET resultante es la suma del WCET `TSAMMessageHandlerItem` y las llamadas a los SAP.

- Una vez que las operaciones simples que modelan los diferentes `TSAMMessageHandlerItem` de `TSAMBasicHandlers` se han generado, éstos se agrupan en las operaciones compuestas. Esta agrupación se realiza de acuerdo con el siguiente criterio: una operación compuesta debe terminar, de tal manera que otra se inicia cuando una operación simple modela una operación de envío, ya sea de naturaleza `TSAMMHiSend` o `TSAMMHiInvoke`. Este proceso de agrupación se denomina composición de operaciones simples asociadas a los manejadores básicos que denominaremos *Basic Handler-Simple Operation* (BH-SOP).

- Por último, se realiza otro proceso de composición, etiquetado, denominados *Asynchronous Message Handler-Operation Composition* (AMH-OC). En este caso, las operaciones resultantes obtenidas de los diferentes manejadores básicos, ya sean simples o compuestos, se reagrupan y son dispuestas siguiendo los mismos criterios que se utilizan para el BH-SOP. Este nuevo nivel de composición tiene como único objetivo reducir los elementos que después forman parte de las transacciones.

La figura 4.6 muestra un ejemplo de una asignación entre un `TSAMAsynchMsgHandler` y su modelo de MAST correspondiente. El manejador de mensajes está formado por una secuencia de tres manejadores básicos, los cuales son: `MSGBH1`, `MSGBH2` y `MSGBH3`, que a su vez están formados por una serie de elementos, tales como acciones, envíos asíncronos (*sends*) y envíos síncronos (*invokes*). Como se ha descrito anteriormente, la transformación se divide en una serie de pasos. En el primer paso del proceso, se crean las operaciones simples que modelan el acceso a los recursos compartidos del componente. Estas operaciones se pueden ver en la figura 4.6 con los nombres `Cmp_Sop_Mutex_Lock` y `Cmp_Sop_Mutex_Unlock`.

En la siguiente parte del proceso, los elementos de los manejadores básicos de mensajes `TSAMBasicHandler` serán transformados en las operaciones simples

[1] El `TACode` modela las distintas ramas que puede tener el código, y las dependencias de los SAPs de cada rama. Para calcular el WCET del `TSAMMessageHandlerItem`, primero se calcula el WCET de cada SAP y a partir de esos cálculos se calcula el WCET de cada rama y se toma el máximo.

`Simple_Operations` de MAST. Por ejemplo, el elemento `TSAMMHiDataHandler`, cuya naturaleza es un elemento básico del manejador denominado `MSGBH1`, se transformará en la operación `BH_Sop_MsgDataHandler_BH1`. El resto de las operaciones simples están situadas en la columna más a la derecha de la figura.

Después de que todas las operaciones simples hayan sido definidas, tiene lugar el proceso de agrupación del tipo BH-SOP. Como se puede observar en la figura, las operaciones sencillas están agrupadas siguiendo los criterios reseñados anteriormente para formar los grupos: `MSGBH1_CO_1`, `MSGBH1_CO_2`, `MSGBH2_CO_1` y `MSGBH3_CO_1`.

En la última etapa de la transformación, el proceso de agrupación AMH-OC reúne las operaciones compuestas de los diferentes manejadores básicos, siguiendo los mismos criterios utilizados en el proceso de agrupación anterior. En el ejemplo de la figura, las operaciones compuestas resultantes se muestran en la columna llamada operación de composición `AMH`. Estas operaciones compuestas son `ASMH_CO_1`, las cuales contienen las operaciones básicas `Cmp_Sop_Mutex_Lock`, la operación compuesta `MSGBH1_CO_1`, además de la operación `ASMH_CO_2`, que agrupa las operaciones de `MSGBH2_CO_1` y `MSGBH3_CO_1`.

4.3.3. Transacción

La última etapa del proceso de transformación abarca la definición de la transacción del modelo de MAST, que especifica los eventos que tienen lugar en el sistema y los manejadores de eventos (*event handlers*). Un manejador de eventos representa una acción que se activa a la llegada de un evento en particular, o un grupo de eventos —esto dependerá de la naturaleza del patrón de activación—. Hay dos tipos principales de elementos que componen la transacción: `Activity` y manejadores de flujo de control (*control flow handlers*). Una `Activity` representa la ejecución de una operación por un recurso planificable, en un recurso de procesamiento, y con algunos parámetros de planificación asignados. Por otro lado, los manejadores de flujo de control se utilizan para generar diferentes eventos internos de salida y, por lo tanto, modificar, dependiendo de su tipo, la trayectoria de la transacción. Un ejemplo de este tipo de manejadores de eventos es el manejador del tipo `branch`, que define una entrada y un conjunto de eventos de salida, generando un evento en sólo una de sus salidas cada vez que llega un evento de entrada —se pueden asignar distribuciones—. Para cada evento definido en el modelo de los requisitos de tiempo real `TSAMRTRequirements` se define una transacción del modelo MAST. Los manejadores de eventos se definen de acuerdo con las siguientes reglas:

- La cardinalidad de los `TSAMMessageHandler` conectados a la recepción de un mensaje particular es mayor que uno, un manejador de flujo de control del tipo *branch* en el que cada una de las salidas representará los posibles flujos de ejecución. Hay que tener en cuenta que las transacciones se construirán en función de una situación de tiempo real. Sólo aquellos, `TSAMMessageHandler` anotados con la situación de tiempo real válida serán usados en la composición de una transacción.
- Cada manejador de mensajes se compone de un conjunto de pasos. Los pasos están formado por las operaciones que ejecutan, recursos compartidos a los que se accede o se libera y el servidor de planificación en el cual se ejecutan las operaciones.
- Cuando un manejador de mensajes incluye el envío de un mensaje asíncrono, en la transacción se introduce un manejador de flujo de control del tipo *fork*, que tiene dos salidas. Cada vez que el *fork* es alcanzado por el flujo, el manejador de eventos interno genera un evento en cada una de las dos salidas al mismo tiempo. Este elemento del modelo se utiliza para representar el envío de mensajes asíncronos, de tal manera que el flujo del servidor de planificación actual continúa su ejecución mientras que al mismo tiempo se encarga de la programación de la recepción.
- Cada envío de un mensaje síncrono estará representado por tres manejadores de paso (*handler step*), uno para la operación de invocación, otro para la ejecución de la operación síncrona y otro que representa el manejo del mensaje de respuesta (*message reply*) por el recurso compartido. Hay que tener en cuenta que la transformación de los mensajes síncronos `TSAMSynchMsgHandler` implementada por componentes reactivos o recurso compartido difieren en un aspecto. Los componentes reactivos tienen asociado un hilo de ejecución, cosa que los recursos compartidos no tienen. Esto se traduce en que la transacción definirá un `EventHandler` del tipo `Fork` para modelar el `TSAMSynchMsgHandler` gestionándose en paralelo en otro recurso de planificación. En caso de que sea un recurso compartido, el elemento `Fork` no se añade a la transacción y estas operaciones son introducidas en el mismo flujo.
- Los elementos del tipo `TSAMMHiSend` y del tipo `TSAMMHiInvoke` se resuelven mediante las conexiones que hay detrás de cada envío. De esta manera la transacción se compondrá de todos los elementos que satisfacen la respuesta a un evento, elementos que están separados en diferentes componentes.

4.3.4. Restricciones temporales

Las restricciones temporales indican restricciones asociadas a un punto en la transacción. En el modelo TSAM son especificadas en el modelo `TSAMRTRequirement`. Los *deadlines* se establecen sobre un `TSAMMessageHandlerItem`, para un determinado `TSAMBasicHandler` —esto es debido a que un `TSAMMessageHandlerItem` puede ser definido en varias secuencias—, pero un *deadline* se asocia a una reacción determinada. Estas restricciones se especifican en el modelo MAST con la asociación de un `LocalDeadline` al `InternalEvent` correspondiente.

4.4. Descripción de las situaciones de tiempo real

No todas las expresiones del lenguaje MAST están disponibles para todas las herramientas de la *suite*. Entre estas expresiones se encuentra el operador `Branch`, el cual no está disponible para las herramientas de análisis, aunque sí para las de simulación. Éste es el mismo operador que hemos usado en las transformaciones para especificar que un mismo mensaje recibido por un componente puede tener más de un tipo de reacción.

Las situaciones de tiempo real permiten clasificar las reacciones de los componentes y los eventos. Éstas se definen y se asignan de una manera fácil mediante etiquetas que denominamos `TSAMRealTimeSituations`. Por ejemplo, para un determinado evento `TEvent` podemos asignar diferentes etiquetas; las reacciones por parte del sistema sólo podrán ser aquellas que tienen las mismas etiquetas. Cuando hablamos de reacciones nos referimos a los `TSAMMessageHandlers`.

Para que la transformación se habilite a un modelo que soporte el análisis de planificabilidad, todas las reacciones a un mismo mensaje tienen que ser clasificadas en función de su situación de tiempo real, de tal manera que no puede haber más que una rama en un punto de elección etiquetado con la misma situación de tiempo real. Ésto permite usar las herramientas de MAST relativas al análisis de planificabilidad, si por contra no se cumple esta restricción, sólo se podrá usar el simulador *JSimMAST*.

Cada evento tiene la posibilidad de definir, mediante etiquetas de situaciones de tiempo real, los manejadores de eventos, que son los correspondientes para un determinado evento en una determinada situación de tiempo real, de tal manera que cuando se componga la transacción sólo los `TSAMMessageHandler` que estén etiquetados con las mismas situaciones de tiempo real formarán parte del flujo.

Hay situaciones de tiempo real típicas, como son las relacionadas con los modos de funcionamiento del sistema. Un sistema puede tener distintos modos de trabajo, en los que se activen distintos eventos, o se dan diferentes respuestas a los mismos eventos por parte de los componentes. Las situaciones de tiempo real nos permiten definir estrategias de análisis con las cuales caracterizar, por ejemplo, sólo las peores reacciones, de tal manera que el análisis de planificabilidad se valide para la peor situación.

Hemos descrito dos estrategias de etiquetado entre estas situaciones de tiempo real ligadas al análisis. El primer patrón lo denominamos gestión por interrupción y el segundo patrón se denomina gestión por sondeo. Son dos tipos de situaciones típicas en software de unidades de control embarcadas en satélite y de amplio uso en otras áreas también relacionadas con los sistemas de tiempo real.

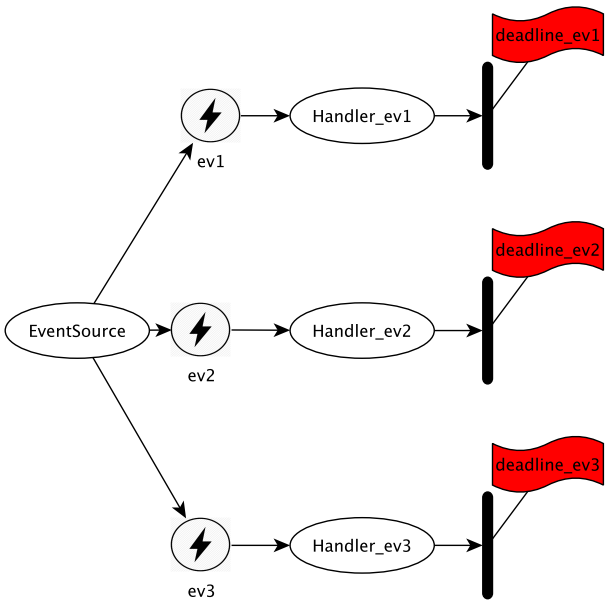


Figura 4.7: Generalización del patrón por interrupción.

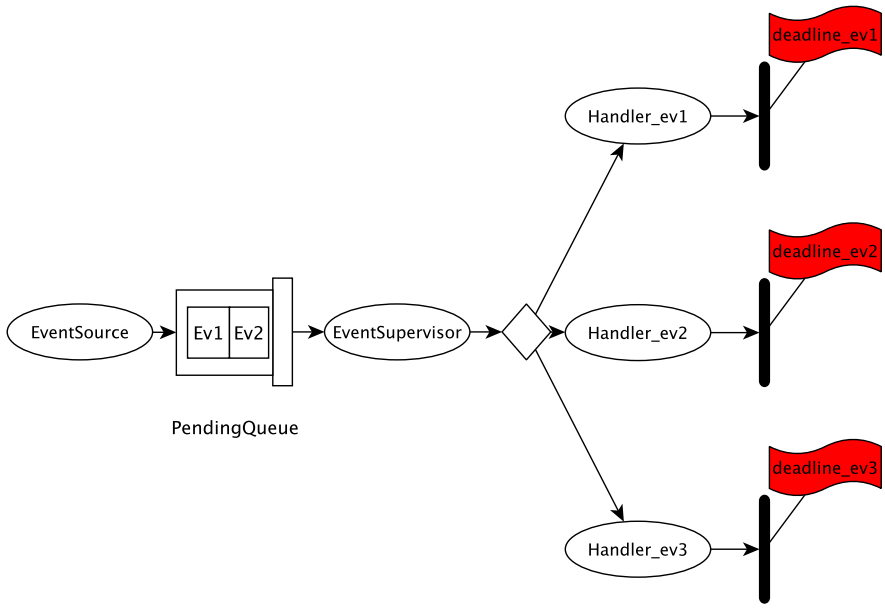


Figura 4.8: Patrón de interrupción con supervisor.

4.4.1. Patrón de gestión de evento por interrupción

El patrón de gestión por interrupción describe un comportamiento en el cual el objetivo del módulo es la gestión de un conjunto de eventos que son registrados por una tarea externa —por ejemplo, mensajes recibidos por el servicio de comunicaciones—. Para cada evento, se realiza una actividad de gestión específica, y se define un plazo específico para su ejecución. Este patrón se muestra de manera conceptual en la figura 4.7 y la implementación, en la figura 4.8, donde podemos observar que pueden ser recibidas diferentes interrupciones, las cuales son gestionadas por un gestor y, dependiendo del tipo de excepción, éstas pueden ser gestionadas por diferentes hilos y acciones. La implementación muestra una cola con información que dependiendo del tipo de evento indica cómo gestionar la información en la cola.

Para la implementación correcta de este patrón en MAST se deberá definir un evento por cada gestión. En el caso mostrado en la figura 4.9 se muestra un

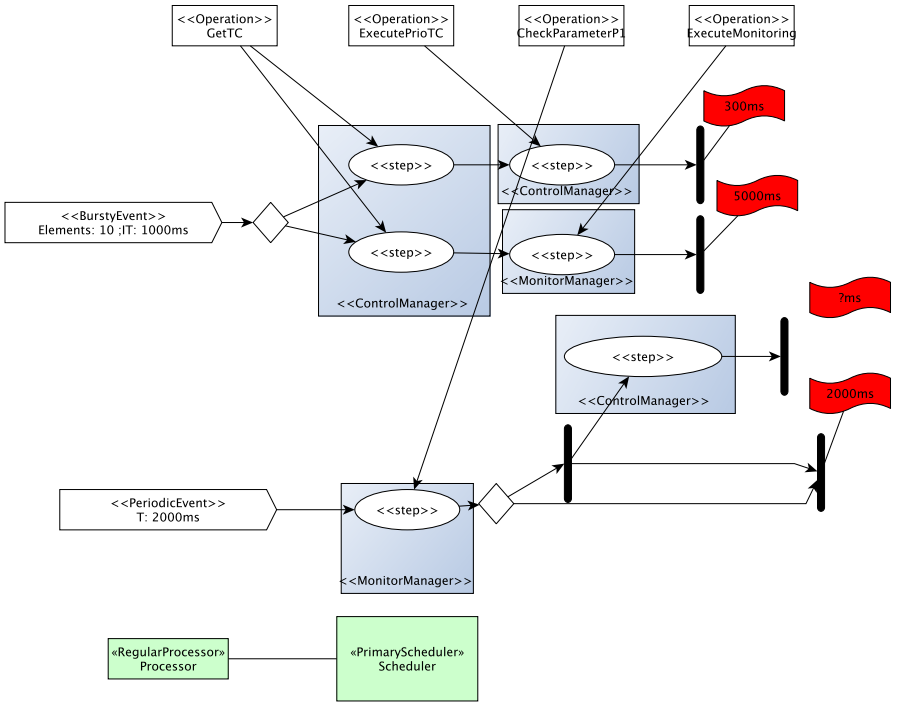


Figura 4.9: Descripción del problema con el patrón.

modelo MAST. El modelo demultiplexado se muestra en la figura 4.10. Podemos ver cómo el evento se transforma en cada una de las posibles gestiones que pueden llevarse a cabo. Los parámetros relativos al patrón de activación serán los de cada gestión por separado. Por ejemplo, en caso de que el evento sea del tipo ráfaga se demultiplexará en los parámetros de los nuevos eventos. Por ejemplo, la suma del número máximo de eventos demultiplexados será igual al evento.

Implementación en el modelo TSAM

Para generar un modelo MAST con estas propiedades hay que configurar el modelo TSAM de la siguiente manera. Cada `TSAMMessageHandler` perteneciente

a cada una de las gestiones es anotado con una situación de tiempo real, por ejemplo, con un nombre descriptivo de la gestión. Cuando se definen los eventos para cada gestión en el modelo `TSAMRTRequirement` cada uno deberá especificar los `TSAMMessageHandler` válidos que corresponden a la situación de tiempo real relativa a su gestión. Esta configuración dará un modelo MAST con las propiedades anteriormente descritas.

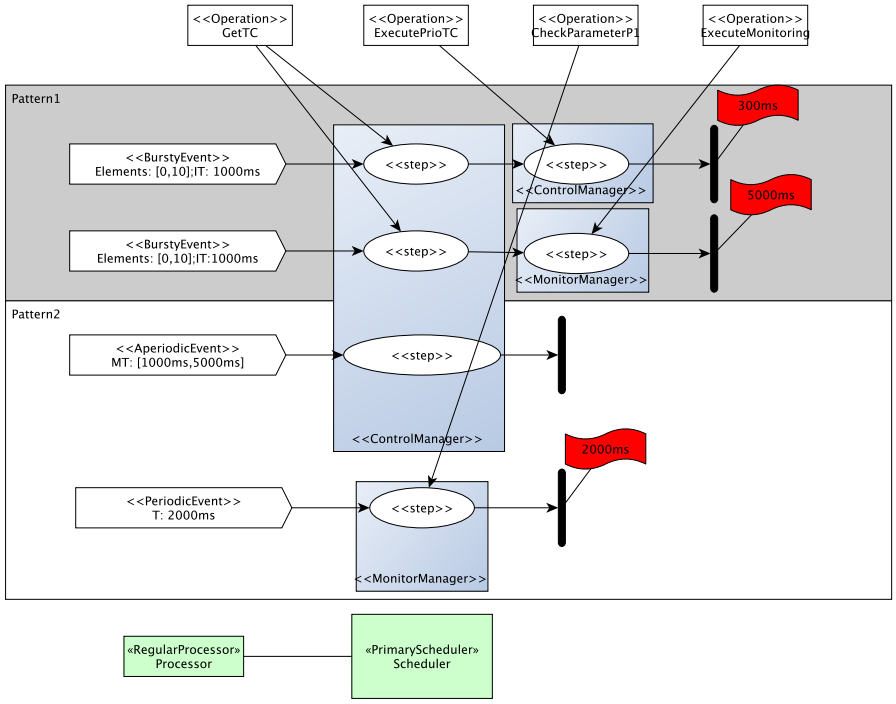


Figura 4.10: Patrón por interrupción. Aplicación del patrón.

4.4.2. Patrón de gestión de eventos por sondeo

El segundo tipo de patrón del que hemos hablado es el que corresponde al manejo de eventos por sondeo. El objetivo del módulo es la gestión de un conjunto de situaciones de excepción que son registradas por otras tareas de la aplicación.

Las excepciones son registradas en una lista cuyo acceso está protegido por un mutex `ExceptionMutex`. Las situaciones de excepción son gestionadas por una tarea de sondeo que se activa periódicamente cada cierto tiempo, resolviendo una excepción por cada activación.

Para la implementación correcta de este patrón en MAST se deberá definir un evento para la acción de gestión del sondeo y un evento por cada suceso de manejo de excepción. En el caso mostrado en la figura 4.9 se muestra un modelo MAST simplificado. El modelo demultiplexado se muestra en la figura 4.10, podemos ver el evento de gestión del sondeo y un conjunto de evento de gestión de excepciones. Los parámetros relativos al patrón de activación para el evento de sondeo son los especificados, mientras que para el resto de eventos de excepción, por ejemplo patrón de activación esporádico, es el tiempo mínimo para la excepción.

Implementación en el modelo TSAM

Para generar un modelo MAST analizable con estas propiedades hay que configurar el modelo TSAM de la siguiente manera. Al `TSAMMessageHandler` asociado a la ejecución sin errores se le etiqueta con una situación de tiempo real. El resto que contemplan situaciones de tiempo real de errores son etiquetas con el sufijo de error. Al `TSAMMessageHandler` encargado de la gestión de las excepciones se le asigna un evento externo, con un patrón del tipo esporádico configurado con el menor tiempo de excepciones que aparecen en el sistema. Esta configuración da lugar a un modelo MAST.

4.4.3. Ejemplo de los patrones de las situaciones de tiempo real

Para poder ilustrar los anteriores patrones planteamos un ejemplo ilustrativo. Un sistema que debe gestionar dos tipos de telecomandos, los cuales son notificados por interrupción y depositados en un *buffer*. Además deberá, de una manera periódica, recolectar una serie de valores; en caso de error, un conjunto de acciones tienen que ser llevadas a cabo.

El ejemplo, muestra claramente los dos patrones que hemos expuesto anteriormente. El caso de los telecomandos corresponde al patrón de evento de gestión por interrupción 4.4.1, y el caso de la comprobación corresponde al patrón de gestión de eventos por sondeo 4.4.2. Un primer modelo MAST es mostrado en la imagen 4.11.

La implementación se basa en los siguientes mecanismos:

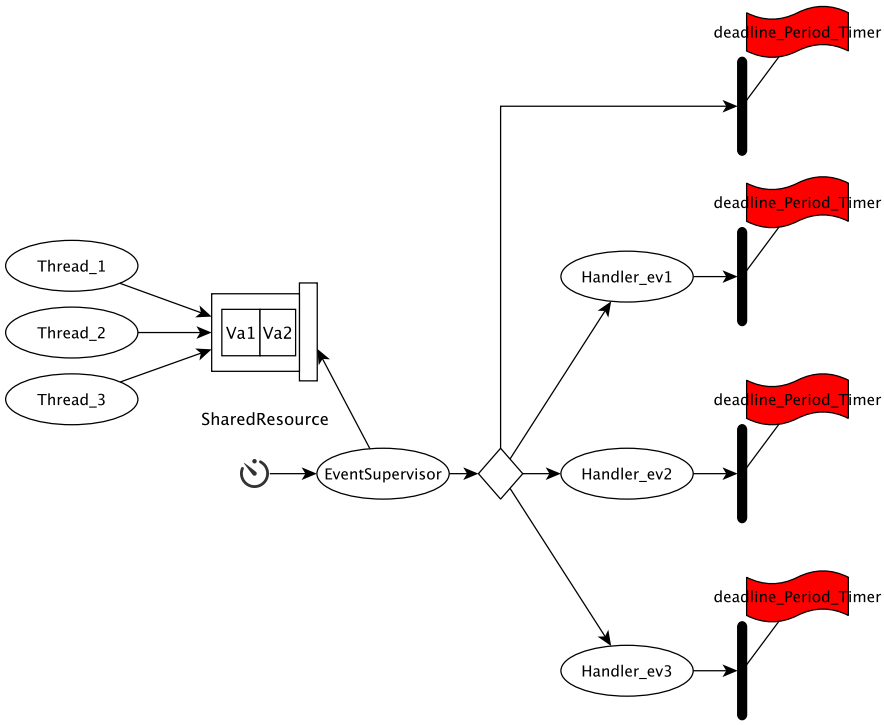


Figura 4.11: Patrón por interrupción. Aplicación del patrón.

- Un `EventSource` registra los eventos que genera en una única cola FIFO de eventos pendientes de ser tratados (*PendingQueue*).

- Tras la generación de cada evento, `EventSource` realiza una activación de la tarea de supervisión de eventos (*EventSupervisor*).

- La tarea de supervisión identifica la naturaleza del evento, y en función de ella delega en una tarea específica (`Handler_ev1,Handler_ev2, Handler_ev3`) su tratamiento con la prioridad requerida.

La implementación se basa en mecanismos justificados en:

- Una tarea *PollingTaskque* se activa periódicamente (cada 2 *segundos*). Lee si existe alguna excepción pendiente. En caso positivo la identifica y, en caso negativo, finaliza.
- En función del tipo de excepción realiza una operación diferente en un *thread* diferente, denominado *ExceptionThread*.
- Toda la actividad generada la tiene que ejecutar antes de la siguiente activación.

4.5. Limitaciones y restricciones

Unas de las limitaciones más significativas es la sobreestimación por la propia composición. La suma de todos los WCET puede contener situaciones en el hardware que no son posibles. Esta sobreestimación no se tendría con un análisis a nivel de transacción y no de las partes de la transacción, pero hemos visto las ventajas que reportan las propiedades de *composability* y *compositionality*. Además esta sobreestimación no suele ser muy significativa.

La herramienta no permite definir elementos `Multicast` del tipo `Branch`. La actual versión de MAST 2.0 no soporta este tipo de `EventHandler`. Esto significa que debemos usar las situaciones de tiempo real para forzar que no se generen elementos del tipo `Branch`.

Elementos anotados de una manera manual desde el modelo TSAM pueden ser no coherentes. Por ejemplo, la introducción de fenómenos en la plataforma que alteran el resultado del análisis de planificabilidad. La coherencia entre el modelo de análisis y el sistema final desplegado en la plataforma se subsana con el proceso de verificación que se presenta en el Capítulo 6.

4.6. Resultados MAST

Los resultados que obtenemos de la herramienta MAST a partir de la proyección realizada desde el modelo TSAM se especifican a continuación:

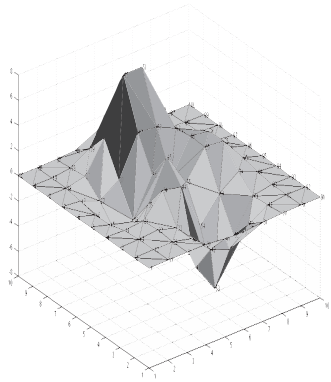
- **Restricciones temporales**, por cada restricción definida en el modelo TSAMRTRequirement se especifica si se cumple o no.
- **Tiempos de respuesta**, por cada `TEvent` y `TSAMMessageHandler` viene especificado su peor tiempo de repuesta.

- **Análisis de sensibilidad**, por cada `TEvent` y `TSAMMessageHandler` se especifica la sensbilidad de su tiempo de respuesta en términos de margen para que no se cumplan las restricciones de tiempo real.
- **Prioridad de los componentes-tareas**, por cada `TSAMComponent` se especifica una prioridad válida por la cual se respetan las restricciones de tiempo real.

En el capitulo 7 mostramos un ejemplo de la generación automática del análisis de planificabilidad —modelo MAST— a partir del modelo de diseño formal de la ICUSW.

Capítulo 5

Modelo de análisis de rendimiento



Si no puedo persuadir a los dioses del cielo, moveré a los de los infiernos.

P. Virgilio

El optimismo es un riesgo laboral de la programación; el feedback es el tratamiento.

K. Beck

5.1. Introducción

En este capítulo abordamos la integración de un modelo de rendimiento. Con este propósito fue elegida la *suite* Palladio *WorkBench*. En la imagen 5.1 mostramos un típico proceso en V donde destacamos los modelos y transformaciones que son definidas en este capítulo. Podemos ver que a partir del modelo transaccional, específicamente el modelo TSAM, se genera el modelo equivalente PCM. Este modelo se sitúa en la fase de validación de sistemas, permitiéndonos corroborar las restricciones temporales. En este capítulo la naturaleza de los

modelos que abordamos son de análisis. En la imagen 5.2 se muestran cuáles son los modelos y transformaciones del proceso que son definidos. Podemos ver que la transformación parte de un modelo orientado al análisis, generando el modelo PCM. El modelo usado para generar el modelo PCM es un modelo TSAM anotado. Este modelo TSAM tiene que tener todos sus elementos anotados, como es el caso de tiempos asociados a las respuestas de los componentes, librerías de servicio y tiempos asociados a la plataforma.

El modelo de análisis de rendimiento es uno de los objetivos marcados para la reducción de los costes económicos, propuesto en esta Tesis Doctoral. Gracias a la transformación automática se genera un modelo que permite verificar las restricciones temporales desde una perspectiva de simulación. En concreto, el objetivo que cubre es:

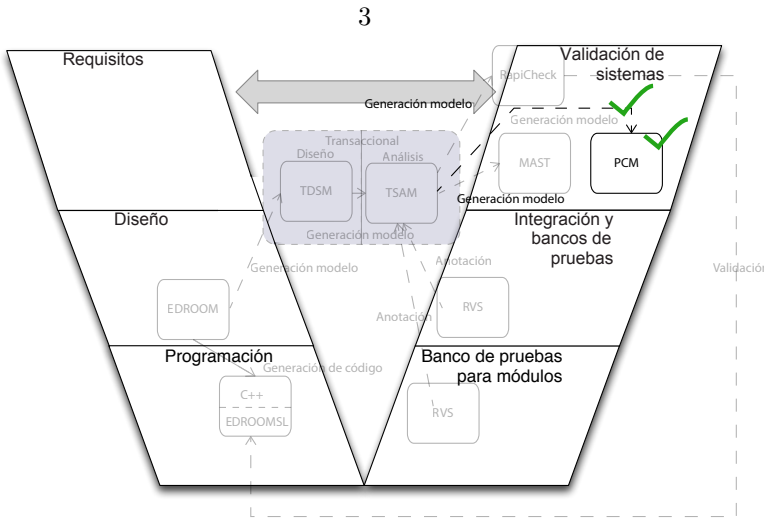


Figura 5.1: Esquema de la transformación de la generación del análisis de planificabilidad.

- «Generación del análisis de rendimiento en conformidad con el estándar ECSS-E-ST-40C. Este modelo de análisis de planificabilidad deberá ser generado a partir del modelo de diseño».

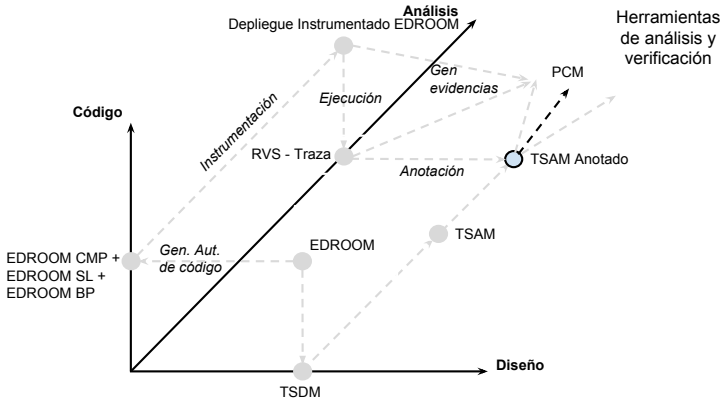


Figura 5.2: En este esquema podemos ver en qué punto de la transformación y sus coordenadas estamos. En este capítulo sólo tratamos con modelos de análisis, en particular con el modelo TSAM y el modelo PCM.

5.2. Introducción del modelo PCM-RT

Hay un gran número de herramientas que proveen análisis de rendimiento de sistemas basados en componentes. Palladio es una de ellas (Becker et al., 2009). Aunque no ha sido diseñada con el propósito de analizar software de sistemas empotrados, donde se aplica con frecuencia la planificación con prioridades fijas, puede ser extendida gracias a su arquitectura. El principal objetivo de este capítulo es habilitar la herramienta Palladio *WorkBench* para su uso en el análisis de sistemas multitarea con planificación basada en prioridades fijas, e integrarla en el proceso que describimos en esta Tesis Doctoral.

Una de las principales características de Palladio es resolver la discontinuidad semántica que existe entre los modelos de análisis y el modelo de implementación de las aplicaciones. Para resolverlo define un modelo descriptivo, llamado PCM, el cual provee un punto de vista que es más cercano a la perspectiva del desarrollador que a la del propio modelo de análisis *per se*. Este modelo es, por sí mismo, un modelo detallado de diseño de la aplicación que facilita su posterior implementación. El componente y los recursos modelados en PCM pueden ser anotados con la información relativa a su rendimiento, y a otros aspectos

tales como mantenibilidad, fiabilidad o coste. En la terminología de Palladio, estas EFP constituyen las llamadas *quality dimensions*, con las cuales el sistema es analizado. El Paladio *WorkBench* incluye un conjunto de *solvers*, basado principalmente en analizadores y simuladores. Estos permiten calcular uno o más EFP modeladas en la aplicación. Cada *solver* define su propio modelo de análisis, y el *WorkBench* incluye la transformación de modelo a modelo que es usada para obtener, desde el modelo de sistema definido por PCM, los modelos de análisis del sistema que corresponden al *solver* específico. Ejemplos de estos *solver* son: SimuCom, un simulador que estudia el rendimiento y la fiabilidad de las aplicaciones; PCM *Solver* (Becker et al., 2009), un *solver* analítico para la determinación de los rendimientos; QPNSolver (Koziolek et al., 2007), un simulador basado en redes de petri que también determina el rendimiento; y finalmente PerOpteryx (Koziolek et al., 2011), un *solver* analítico que determina el rendimiento y la fiabilidad, que tiene como objetivo asistir al arquitecto de aplicaciones en la óptima construcción de los sistemas.

Palladio presenta actualmente una limitación a la hora de modelar y analizar aplicaciones software empotradas, ya que no contempla la posibilidad de que éstas estén construidas mediante una solución multitarea basada en planificador de prioridades fijas con desalojo. Este tipo de planificación, sin embargo, es muy común en las aplicaciones empotradas, ya que facilita el cumplimiento de las restricciones temporales que se fijan como requisitos. Esta limitación se resolverá mediante una extensión de PCM y una nueva infraestructura de análisis de rendimiento que sea coherente con esta política de planificación. La solución propuesta permite la definición y la simulación de un sistema en el cual cada componente es ejecutado por una tarea con una prioridad específica. Además, cada componente puede acceder a unos recursos compartidos protegidos por protocolos que eviten la inversión de prioridades. Estos protocolos son: el protocolo de herencia de prioridad y el protocolo de techo de prioridad inmediato.

En este capítulo mostraremos cómo la extensión incorporada permite modelar aplicaciones software multitarea basadas en tiempo real. Gracias a esta extensión se define una transformación desde el modelo TSAM a este nuevo modelo, lo que dota al proceso de las diferentes herramientas de análisis de las que consta PCM.

5.2.1. Roles del Palladio Component Model

Palladio define un conjunto de roles que son los encargados de describir las diferentes vistas del modelo PCM. Estos roles tienen diferentes perspectivas del

proceso de diseño software. Estas perspectivas definen el comportamiento del software, la arquitectura del sistema, la plataforma y los casos de uso del sistema. Con este propósito se han definido los roles: desarrollador de componentes, el arquitecto software, el experto del dominio y el encargado del despliegue del sistema.[1]

Los modelos de Palladio *WorkBench* son cercanos a los programadores que desarrollan las aplicaciones. De tal manera que la curva de aprendizaje es baja, ayudando al programador a generar modelos que le permitan caracterizar las EFP del sistema tales como rendimiento, tiempo de respuesta, utilización de los recursos, fiabilidad, disponibilidad, seguridad, mantenibilidad y reusabilidad.

Palladio *WorkBench* está basado en la aplicación de técnicas MDE. El modelo de descripción creado y anotado se puede transformar a diferentes modelos de análisis. Los modelos de análisis que genera están basados en teoría de colas, procesos algebraicos estocásticos, y redes de estocásticas petri. Gracias a estas transformaciones, se eliminan las discontinuidades semánticas (Balsamo et al., 2004), eliminando los errores implícitos del proceso.

Desarrollador de componentes

El desarrollador de componentes está encargado de definir las propiedades funcionales y EFP de los componentes. La visión del desarrollador está limitada a la definición de los componentes y a cuáles son los servicios que provee y requiere en función de un conjunto de interfaces. Palladio define varios tipos de interfaces, permitiendo comunicar los componentes con elementos software (otros componentes) o hardware (recursos). Además, estas interfaces pueden tener comportamientos asíncronos o síncronos.

El desarrollador define las interfaces del tipo `Operational` para especificar la comunicación síncrona entre componentes. Estas interfaces están compuestas por *signatures*, teniendo un comportamiento similar a CORBA IDL. Además, pueden ser configuradas como protocolos, lo que permite añadir restricciones de orden de llamada de los `signature`. Las interfaces se implementan en los componentes mediante puertos. En el caso de que el componente quiera solicitar los servicios de una interfaz, el puerto deberá ser del tipo *Required Role* (RequiredRole). Si por el contrario lo que se quiere es proveer el servicio especificado por la interfaz, se utilizará un puerto del tipo *Provided Role* (ProvidedRole). Una vez definidos los componentes e interfaz, ambos son almacenados en reposito-

[1] En inglés, *system deployer*.

```
void execute(int number,
             List array){

  requiredService1();

  // internal computation
  innerMethod();

  if (number>=0)
    for (item in array)
      requiredService2();
  else
    requiredService3();
}
```

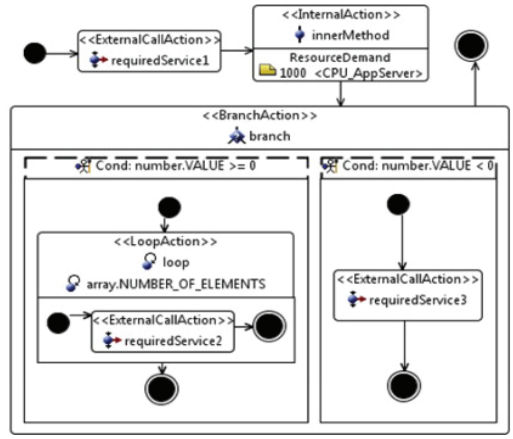


Figura 5.3: Ejemplo de un RDSEFF, a la izquierda, un ejemplo de código en Java, a la derecha su modelo equivalente en RDSEFF.

rios; a estos repositorios el arquitecto puede acceder para componer un sistema y desplegarlo sobre una plataforma.

Para modelar la comunicación síncrona, Palladio define un tipo de interfaz denominado `EventGroup` (Rathfelder, 2013). Éste tipo se utiliza para poder definir un comportamiento entre componentes similar al envío de señales entre procesos. Los puertos asociados a esta interfaz se denominan `SourcePort` y `SinkPort`. Los `SourcePort` son los encargados de generar las señales, que son recibidas por los `SinkPort`.

Los componentes, además, se suscriben a una interfaz de recursos hardware denominadas `InfraestructureInterface`. Esto permite poder desligar los componentes de la definición de la plataforma. Las interfaces de infraestructura de recursos permiten, además, definir diferentes tipos de demandas sobre el recurso de la plataforma.

Los `Resource demanding` RDSEFF modelan la implementación de una interfaz *signatures* por parte de un componente. Esto es: un componente debe definir un `RDSEFF` por cada uno de los *signature* de un puerto del tipo `ProvidedRole` o `SinkEvent`. Éstos son parametrizables en función de los parámetros del *signature*, del entorno de ejecución y de los perfiles de uso. Los RDSEFF sólo añaden elementos de control de flujo para definir cuáles son los servicios que son demandados en función de las interfaces requeridas, las dependencias paramé-

tricas y el uso que se realiza de los recursos. El objetivo de esto es mantener la propiedad de caja negra de los componentes, de tal manera que no es necesario tener conocimiento del comportamiento funcional de cada uno de ellos.

Los RDSEFF —un ejemplo se muestra en la figura 5.3— expresan su comportamiento de una manera similar al diagrama de estados del modelo UML 2.0. PCM define operadores condicionales, bucles, operadores del tipo *fork* y *joints*, y operadores de demandas sobre recursos. Cada demanda puede ser expresada como un valor fijo o mediante expresiones algebraicas, valores parametrizables o expresiones estocásticas. Parte de los RDSEFF puede ser protegida por secciones críticas. Éstos se definen mediante *Passive Resource* (PassiveResource). El alcance de un recurso pasivo es el componente en el cual ha sido definido. El acceso y liberación de estos `PassiveResource` se realiza mediante los operadores *adquire* y *release*. Un ejemplo en la figura 5.4 se muestra el uso de operadores de flujo, operadores de llamada y operadores de control de hilos de ejecución.

Arquitecto software

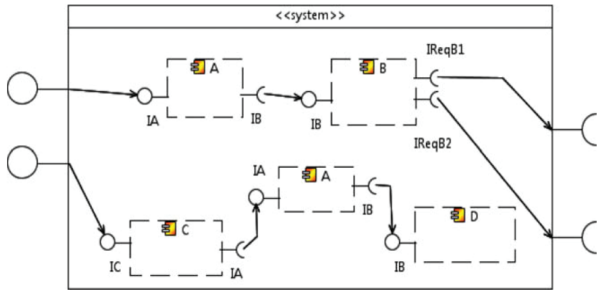


Figura 5.4: Ejemplo de arquitectura definida por el arquitecto software.

El arquitecto es el encargado de diseñar un sistema a partir de los componentes configurados en los repositorios de Palladio. En la composición no hay límite en las instancias de un mismo componente. El marco en el que los componentes son desplegados se denomina `System`. Los sistemas pueden definir también puertos, que modelan la interacción con el entorno de trabajo. Estos puertos pueden ser de dos tipos: `SystemProvidedRole`, utilizados cuando el sistema provee una interfaz, o `SystemRequiredRole`, empleados cuando el sistema requiere una interfaz. Un ejemplo se muestra en la imagen 5.4

System deployer

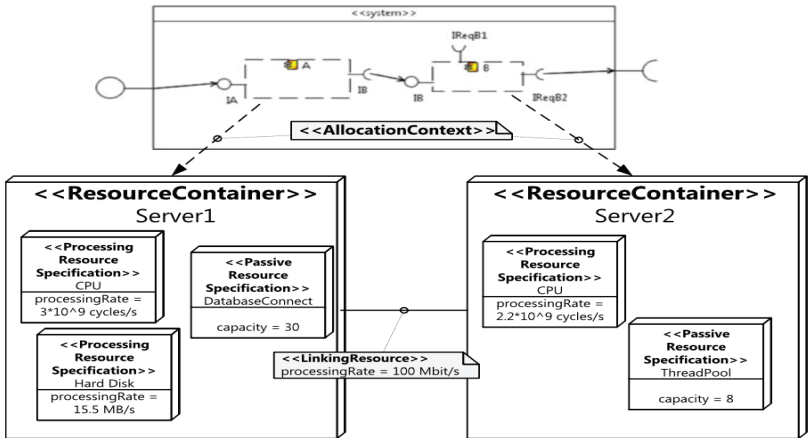


Figura 5.5: Ejemplo de despliegue de un sistema.

El *System deployer* es el encargado de definir los recursos de los que consta la plataforma. Los elementos que pueden ser definidos son: recursos de procesamiento, como CPU, y recursos de red. Además, deberá definir la relación entre los anteriores elementos. De esta manera, Palladio permite definir sistemas distribuidos. Por último, el *System deployer* debe desplegar los componentes por CPU. Un ejemplo se muestra en la imagen 5.5.

Experto del dominio

Este último rol es el encargado de crear y definir los casos de uso del sistema a partir de los cuales se analiza —un ejemplo se muestra en la imagen 5.6—. Para ello define los patrones de uso de los servicios que provee el sistema a través de sus puertos. Palladio permite definir diferentes escenarios, cada uno representa una definición de demanda en particular. Cada escenario es formado por un `Workload`. Hay dos tipos de `Workload`: `Open` y `Close`, que permiten definir dos comportamientos diferentes a la hora de generar los eventos del sistema. El tipo `Close` genera simultáneamente el total de los eventos configurados, que vuelven a ser generados cuando el último evento es consumido por el sistema, añadiendo

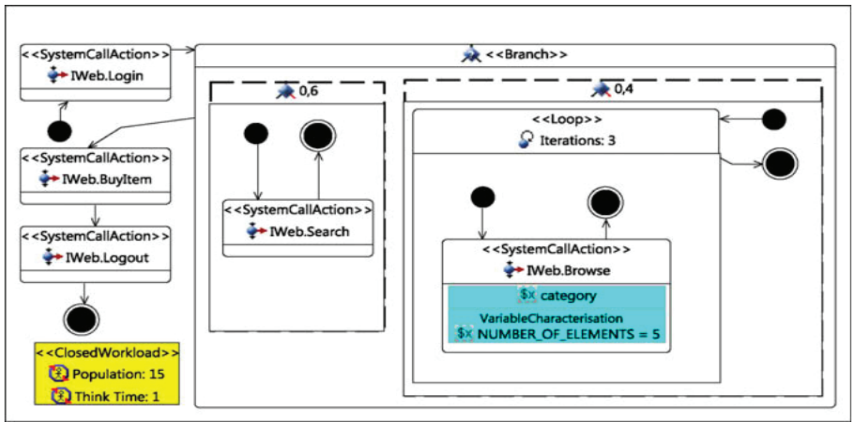


Figura 5.6: Ejemplo de una definición de un `Workload` de un caso de uso.

un tiempo de espera denominado *think time*. En el tipo `Open` se genera un evento cada cierto tiempo, y no se tiene en cuenta si ha sido consumido o no por el sistema. Cada `Workload` tiene un conjunto de operaciones similares a los RDSEFF, que ofrecen una manera versátil de definir los entornos de generación de peticiones del sistema. Estos operadores son: *loop*, *if*, *call to system* y *timing delays*.

5.3. Transformación desde TSAM a PCM

La transformación ha sido dividida en seis pasos:

- **Modelado de componentes**, definición de componentes. Se asocia un componente PCM a cada uno de los componentes empleados en el modelo TSAM.
- **Comunicación entre componentes**, definición de los protocolos de comunicación que permiten la comunicación entre componentes.
- **Modelado de los recursos compartidos**, definición de los elementos de gestión de la memoria compartidos, tales como *buffers* o *pools*, y su forma de acceso.

- **Modelado de la gestión de los eventos externos**, definición de los patrones de activación de las interrupciones y excepciones.
- **Servicio de planificación**, definición del algoritmo de planificación.

5.3.1. Modelado de componentes

Por cada `TSAMComponent` definido en el modelo `TSAMComponent` se define un `BasicComponent` de `PCM`. La simplicidad de esta transformación radica en que ambos modelos han sido definidos bajo el mismo paradigma CBSE.

5.3.2. Comunicación entre componentes

En esta sección tratamos todo lo relativo a la comunicación de los componentes. En la comunicación se describe tanto la definición de las interfaces que los componentes usan para comunicarse como la manera en que los componentes envían los mensajes.

La comunicación entre componentes en PCM se define mediante interfaces, que pueden ser de naturaleza asíncrona o síncrona, y puertos. Cada puerto tiene una interfaz asociada, que es provista si el componente es emisor del mensaje o requerida si es receptor del mensaje. Si la interfaz es de tipo asíncrona se define una interfaz del tipo `EventGroup`, consecuentemente, el emisor de mensajes tiene que definir un puerto del tipo `SourceRole`, y el receptor implementa un puerto del tipo `SinkRole`. En caso de que la comunicación sea del tipo síncrona, la interfaz es del tipo `Operational`, donde el emisor del mensaje es un puerto del tipo `RequiereRole` y el receptor es del tipo `ProvidedRole`.

Para generar las interfaces PCM nos basamos en la definición de los protocolos TSAM. Por cada protocolo y dependiendo de su naturaleza generamos una interfaz `Operational` en caso de tener en el modelo TSAM un manejador de mensajes de naturaleza `TSAMSynchMsgHandler`, y un `EventGroup` en caso de tener un `TSAMAsynchMsgHandler`. Los mensajes definidos en los protocolos se transforman a PCM en forma de métodos que deben ser implementados por los elementos que provean el servicio. La información relativa al sentido de los mensajes, mensajes de entrada y mensajes de salida, no se tiene en cuenta en la definición de las interfaces de PCM de una manera explícita, ya que sólo mediante los puertos PCM puedes requerir o proveer las funcionalidades especificadas.

Como vimos en las restricciones de diseño del capítulo 3, el consumo de los mensajes recibidos por parte del componente receptor tiene que ser secuencial.

Por este motivo se deberá acceder en exclusión mutua al componente. Para ello, por cada componente `BasicComponent` se definirá un recurso compartido `PassiveResource`. Por cada definición de acciones de reacción a un mensaje, la primera y última acción siempre serán, respectivamente, una operación de adquisición del recurso compartido y una acción de liberación de ese recurso compartido. Las operaciones que interactúan con los recursos compartidos son descritas en detalle en el apartado 5.4. La transformación relativa a este mecanismo requiere que por cada TSAMComponent se genere un `PassiveResource`. Este `PassiveResource` se asocia al componente `BasicComponent`, que era el mismo TSAMComponent.

5.3.3. Modelado de los recursos compartidos

Los recursos compartidos se definen como `BasicComponent` que sólo implementan servicios de naturaleza síncrona. Ya vimos el significado de los recursos compartidos en las restricciones aplicadas al modelo de diseño. Éstos son elementos de carácter reactivo que no tienen un hilo de ejecución. Las únicas interfaces que pueden implementar o requerir son las de naturaleza síncrona.

Por cada componente recurso compartido `TSAMComponent` se genera un `BasicComponent`. Ya vimos en el capítulo 3 que esta clase de componentes sólo interactúa mediante mensajes de respuesta.

5.3.4. Modelado de los eventos externos

En la figura 5.7 se muestran los elementos en PCM que definen tanto los eventos externos como el *System Run-Time* relacionado con su atención. En PCM la definición de eventos externos se realiza mediante la descripción de los casos de uso sobre puertos del sistema, y sus correspondientes *workloads*. Para la definición del *System Run-Time* se ha creado un componente específico.

El componente *System Run-Time* es el encargado de modelar los servicios de temporización e interrupción. Por cada elemento que se suscribe a estos servicios se creará un puerto del tipo asíncrono `SourceEvent` en el componente CRT. A su vez, por cada evento que activa el CRT se creará un elemento del tipo `SystemProvidedRole`. De esta manera, la reactividad del sistema se modela en función de eventos de interrupción y temporización.

El modelado de los `Top-half` y los `Bottom-half` se realizará mediante RDSEFF asociados al CRT. El CRT es el encargado de definir tanto el manejador de la interrupción asociada al evento, que actúa como `Top-half`, como la tarea que hace de `Bottom-half`, y que permite notificar a los componentes el evento

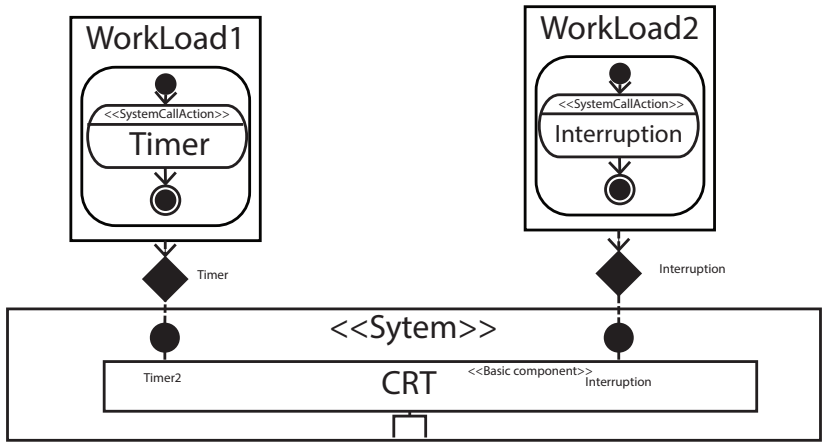


Figura 5.7: Definición de los eventos externos en PCM y su propagación hasta el *system run-time*.

mediante un mensaje. Ambos elementos están protegidos mediante un *mutex* para garantizar la secuencialidad. Este `Bottom-half` puede ser gestionado por varios hilos de ejecución, por ejemplo, usando *threads pool*. Esto en Palladio se modela de una manera equivalente configurando el número de accesos al *mutex* del `Bottom-half` con el número de hilos que gestionará las interrupciones en el `Bottom-half`.

Finalmente, los patrones de activación de los eventos se definirán en un `Workload` del tipo `Open`. El modo elegido es `Open`, ya que la generación del siguiente evento no dependerá de la finalización de la atención del mismo por parte del sistema (como pasa en el caso del tipo *Closed*), ya que los eventos pueden solaparse antes de ser atendidos. En el `ScenarioBehaviour` de los `Workload` se describe el tipo de patrón de activación para cada evento y pudiendo incluir elementos de alteración del instante de disparo, tales como los *offset*, o elementos que añadan aleatoriedad al disparo del evento. Tres tipos de patrones son descritos y mostrados en la imagen 5.8: ráfaga, esporádico y periódico. Éstos son los mismos que son descritos en el modelo TSAM.

- **Patrón bursty**, este patrón, el cual ya se definió en el modelo TSAM, nos ayuda a representar conjuntos de secuencias de interrupciones. Los pará-

metros que hay que conocer para modelar este patrón son el menor tiempo de llegada de la interrupción principal, el número de interrupciones de la secuencia y el menor tiempo entre interrupciones de la secuencia. En la figura 5.8 se presenta este patrón de activación en la primera fila, donde el relámpago grande corresponde a la interrupción principal; el pequeño, a la interrupción de los elementos de la secuencia; y cada círculo corresponde a la estimulación del sistema por cada una de las interrupciones de la secuencia. La transformación al PCM es relativamente sencilla, se configura con el parámetro `InterArrivalTime` del elemento `Workload`, a continuación en el `ScenarioBehaviour` se modela la secuencia de interrupciones. Ésta se modela mediante un bucle `LoopAction` donde el número de iteraciones es el número de elementos de la secuencia, ya sea un número máximo o una variable aleatoria. Por último, para especificar el menor tiempo de llegada se añade al cuerpo del bucle un elemento `DelayAction` que se coloca justo después de la llamada al sistema, al igual que otros parámetros que hemos visto, se puede elegir entre un tiempo máximo o una variable aleatoria. En la primera fila se puede ver la ilustración de esta transformación donde el relámpago grande se coloca justo en el elemento `InterArrivalTime`; el pequeño, en el `DelayAction`, y la operación `LoopAction` (ilustrada como un rombo) se configura con el número máximo de interrupciones de la secuencia.

- **Patrón esporádico**, este patrón nos ayuda a representar patrones de activación asociados a elementos de interrupción. El parámetro que hay que conocer para definir este patrón es el menor tiempo de llegada de la interrupción. En la figura 5.7, en la segunda fila, se ilustra el patrón, donde el relámpago especifica el *arrival time* del evento. La transformación a PCM configura el parámetro del *arrival time* en el `InterArrivalTime` del `Workload`. En el cuerpo del caso de uso `ScenarioBehaviour` se especifica la llamada al sistema.

- **Patrón periódico**, este patrón permite modelar los eventos asociados a temporizadores o eventos que se comportan de una manera periódica. El parámetro asociado a este tipo de eventos es el periodo del evento. En la imagen 5.7, en la tercera fila, se muestra el patrón del tipo periódico donde el parámetro periodo se muestra en forma de *timer*. La transformación al PCM configura el parámetro del *period time* en el `IterArrivalTime` del `Workload`. En el cuerpo del caso de uso `ScenarioBehaviour` se especifica la llamada al sistema, el cual genera una reacción.

Con el objetivo de generar, de una manera automática los modelos PCM relativos a los eventos externos, hemos definido una transformación automática. El modelo TSAM que contiene esta información es el modelo al TSAMRTRequirement. Este modelo describe los mismos patrones que han sido mencionados anteriormente. Por cada `TEvent` se generan un *workload* usando el patrón de construcción correspondiente al tipo y los parámetros de configuración como, por ejemplo, el periodo.

La definición de los `Bottom-half` y `Top-half` se realiza mediante el componente CRT de PCM. Por cada evento `TEvent` se definen como parámetros el componente `TSAMComponent` y el manejador de mensajes `TSAMMessageHandler`, que está asociado a la activación del sistema. La transformación equivalente a PCM usa el patrón relativo al CRT, donde por cada evento se crea un `SystemProvidedRole`, el cual delega en un `SystemProvidedRole` del componente *System Run-Time*. El RDSEFF es el descrito en el `TSAMMessageHandler`, que tiene como objetivo implementar las demandas sobre el recurso de procesamiento. Finalmente, el componente que hace el papel de CRT incorpora diversos RDSEFF asociados a los manejadores de interrupción y excepciones, así como el código de servicio que convierten este tipo de eventos en mensajes enviados a los componentes. Al igual que ocurre con los componentes Palladio, el componente dispone de un puerto de interfaz `IPriority` utilizado para definir la prioridad de acceso a la CPU.

5.3.5. Servicio de planificación

La planificación basada en prioridades de la CPU se modela mediante un nuevo tipo de recurso activo denominado `PPS_CPU`, al que se le asigna la política de planificación *priority preemption scheduler*. Este elemento forma parte de la extensión del modelo Palladio (Fernández-Salgado et al., 2015). `PPC_CPU` provee la interfaz `IPriority` que define una *signature* distinta para cada prioridad, tal como se muestra en la imagen 5.9, y a través de la cual se reciben las peticiones de acceso a la `PPS_CPU`, tanto de los componentes que forman parte del diseño como del *system Run-Time*.

Todos los `BasicComponent` definen un puerto del tipo `infrastructure RequiredRole` que requiere la interfaz `IPriority`. Esta interfaz que, como se ha comentado anteriormente, forma parte de la nueva extensión añadida al PCM permite especificar la prioridad con la que un componente solicita la CPU al planificador. Entre los `BasicComponent` está el correspondiente al CRT. Éste usará las prioridades más urgentes del sistema.

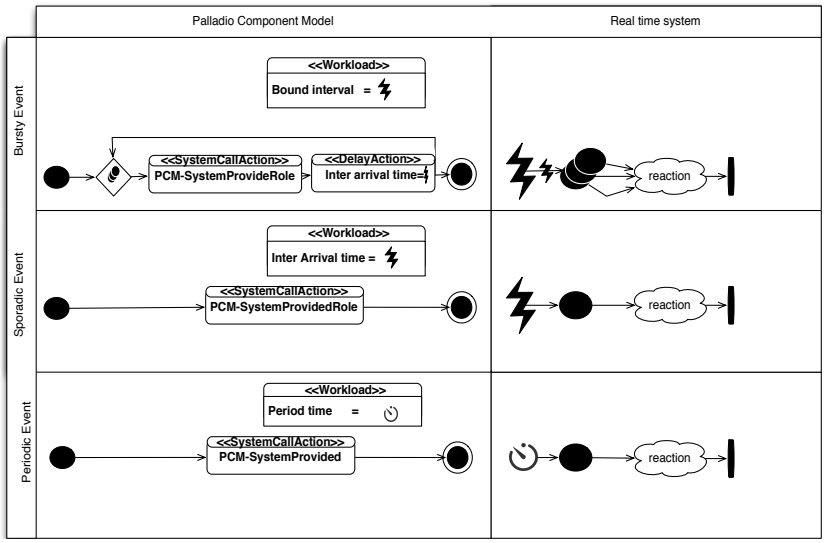


Figura 5.8: Definición de los tres tipos de patrones de activación de un evento: ráfaga, esporádico y periódico.

En la transformación a PCM desde los modelos TSAM, este recurso se genera siempre, y no depende de ningún parámetro del TSAM. La prioridad de cada uno de los componentes deberá ser especificada sobre el modelo PCM resultante. No hemos asignado ninguna prioridad a los componentes, ya que es el resultado de un análisis global del sistema, y no puede formar parte del modelo transaccional de análisis.

5.3.6. Recepción y envío de mensajes

Los RDSEFF son usados para modelar el modo en el que los mensajes son manejados por los componentes. Cada uno de los RDSEFF define las repuestas dadas por los `BasicComponent` asociados a la recepción de un evento en uno de sus `SinkRole`, teniendo en cuenta que la recepción es la equivalente a la de un mensaje que forma parte de la transacción. El envío de mensajes desde un componente a otro forma parte de los propios RDSEFF y se modela mediante el envío de un evento a través del puerto `SourceRole` asociado al correspondiente mensaje.

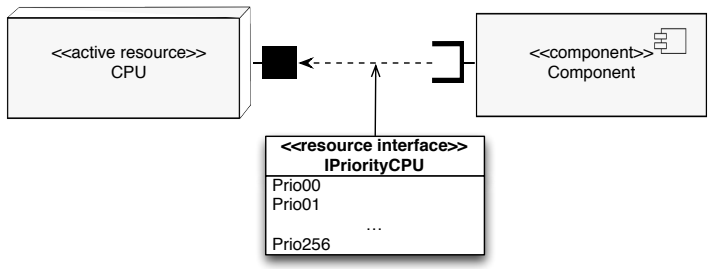


Figura 5.9: Han sido añadidos nuevos elementos para implementar una política de planificación basada en prioridades fijas. PPS_CPU es un recurso activo que implementa demandas con prioridad. IPriority es una interfaz basada en prioridades, definiendo un total de 256. Los componentes requieren un `ActiveResource` que implemente la interfaz Ipriority.

5.3.7. Prioridad del componente

La prioridad con la que se ejecuta un componente se modela mediante las demandas de los RDSEFF, y cada demanda de acceso a un `ActiveResource` se efectúa con una solicitud a la interfaz `IPriority`. Cada *signature* de la interfaz corresponde a una prioridad, como ya hemos destacado anteriormente.

5.3.8. Acceso a recurso compartido

Cuando un componente accede a un recurso compartido debe comunicar su prioridad al recurso compartido (sólo en caso de que el protocolo de acceso a recursos compartidos esté configurado a techo de prioridad inmediato). Esto se precisa en cada RDSEFF que realiza una demanda del tipo `ProvidedRole` sobre un recurso compartido. Junto con la demanda se debe especificar como parámetro la prioridad de acceso. La prioridad de las demandas dentro del recurso compartido depende del protocolo configurado.

La transformación desde los modelos TSAM al modelo PCM toca los elementos relativos a los `TSAMMessageHandler`. Cada uno de los `TSAMMessageHandler` corresponde a la reacción del componente por parte de la recepción de un mensaje a través de un puerto. En el caso de PCM, un componente puede enviar o suscribirse a interfaz. Cuando el componente es el proveedor del servicio `ProvidedRole` o el receptor de un evento `SinkEvent`, se describe la reacción del

componente mediante un RDSEFF. Dependiendo de la naturaleza de la recepción, éstos se describen de una manera distinta.

5.3.9. Respuesta a mensajes asíncronos

Por cada `TSAMMessageHandlerAsynch` se crea un RDSEFF asociado a un puerto del tipo `SinkRole`.

5.3.10. Respuesta a mensajes síncronos

Por cada `TSAMMessageHandlerSynch` se genera un RDSEFF asociado a un puerto del tipo `OperationProvided`, donde al igual que en los `TSAMMessageHandlerAsynch`, se asociará a un puerto y *signature* equivalentes en la transformación del modelo PCM.

Cada uno de los `TSAMMessageHandlerItems` corresponde a un RDSEFF. Éstos son especificados a continuación:

- **Action** y **DataHandler**, equivalen a un `ProcessingElement`, donde el WCET de la peor rama anotada es la demanda de CPU que se especifica en el modelo.
- **Send**, equivale a un `Action` que especifica como demanda de CPU el WCET asociado al elemento. Además, la siguiente operación es un `SourceEvent`. El puerto que se especifica y los mensajes son los equivalentes a los transformados en PCM.
- **Invoke**, equivale a un `Action` que especifica la demanda de CPU anotado con su WCET asociado al elemento. Además la siguiente operación es un `OperationalRequired` donde el puerto que se específica y sus mensajes son los equivalentes a los transformados en PCM.

El orden de ejecución de los diferentes elementos es el siguiente: DataHandler, Action, Send, Invoke y Msgreply.

En caso de que más de un `TSAMMessageHandler` haya sido definido en el mismo `TSAMComponent`, para gestionar un mismo mensaje por un mismo puerto, se compondrá un solo RDSEFF para cada uno. Al principio se especificará un elemento del tipo `Branch` y a cada rama correspondería un `TSAMMessageHandler`. El tipo de condicional obtenido tras la transformación es el probabilístico, donde las guardas son ejecutadas con una frecuencia que depende de la probabilidad. La distribución de probabilidad que hemos especificado es equiprobable.

En el cuadro 5.1 muestra un esquema de la transformación del modelo de sistema para el PCM extendida.

Cuadro 5.1: Esquema de la transformación a PCM.

Elemento de sistema de diseño.	TSAM.	PCM-RT.
Modelado de tareas.	TSAMComponent C_i.	A `BasicComponent` por tarea: TC_i.
Modelo de comunicación entre tareas.	Mensajes de entrada y salida.	`SinkRole` para un *msg* de entrada, `SourceRole` para *msg* de salida.
	Cada T_i maneja secuencialmente *msg* de entrada.	Cada TC_i captura `PassiveResource` cuando *msg* es recuperado de la cola, y lo libera cuando es consumido.
Modelado de acceso a los recursos compartidos.	Recursos compartidos del sistema: SR_j.	A `BasicComponent` SRC_j por SR_j.
	SR_j servicios provistos.	SRC_j define un puerto `ProvidedRole`.
	T_i acede SR_j.	TC_i define un puerto `RequiredRole` enlaza a SRC_j un puerto `ProvidedRole`.
Modelado de gestión de eventos externos.	El *Run-Time* del sistema maneja cada evento: E_k.	Un único `BasicComponent` para modelar el *Run-Time*: RTC.
	Gestión de eventos del `Top-half` del *Run-Time*.	RTC implementa un `ProvidedRole` por E_k.
	E_k propagación a T_i vía `Bottom-half`.	RTC implementa un puerto `ProvidedRole` por E_k enlazado al puerto TC_i `SinkRole`.
	E_k notificación.	Un `SystemProvidedRole`, enlaza a puerto RTC `ProvidedRole`, vía `SystemDelegationConnector`.
	E_k patrón de activación.	Un `ScenarioBehaviour` y un `Workload` son usados para notificar E_k mediante su asociado `SystemProvidedRole`.
Modelado de servicios de planificación.	T_i asignación de prioridad.	TC_i requiere un `IPriority` `RequiredRole`. TC_i RDSEFF usa la asignación de prioridad de la interfaz `IPriority`.

continúa

continúa

Elemento de Sistema de diseño.	TSAM.	PCM-RT.
	Asignación de la prioridad del *Run-Time* del sistema.	*RTC* requiere un `IPriority RequiredRole`. `Top-half` y `Bottom-half` RDSEFF requiere la interfaz `IPriority` de su correspondiente prioridad.
	Planificador basado en prioridades.	El recurso `PPS_CPU` implementa la interfaz `IPriority` requerida.
Modelado de detalles de implementación.	T_i gestión de la implementación de los *msg* de entrada.	Un RDSEFF usado para el evento `SinkRole` asociado a cada *msg* de entrada.
	E_k `Top-half` y `Bottom-half` implementación.	A RDSEFF para cada *RTC* `ProvidedRole` requiere el correspondiente *signature* `IPriority`.
	Envío de mensajes de salida.	Los RDSEFF usan mensajes de salida `SourceRole`.
	Implementación de los servicios de recursos compartidos.	Un RDSEFF para cada `ProvidedRole` con prioridad como parámetro.
	Tiempo de ejecución o tiempo de computación.	Los RDSEFF parametrizan la demanda `PPS_CPU`.

5.4. Detalles de implementación del PCM-RT

5.4.1. Introducción al comportamiento de SimuCom

SimuCom crea hilos de ejecución por cada nueva generación de eventos externos y envío de mensajes entre componentes. Teniendo en cuenta la concurrencia introducida por estos hilos, la evolución de la simulación es controlada de la siguiente manera. Cada hilo, tanto si ha sido originado por un evento externo o por el envío de un mensaje, desencadena una secuencia de ejecución de RDSEFF. El `PassiveResource` se captura en la primera acción de cada secuencia, y se libera en la última. Esto permite modelar que el componente se ajusta al paradigma RTC descrito en el capítulo 3, en el caso de que sea proactivo o reactivo, y también modela el acceso en exclusión mutua, en el caso de que el componente sea un recurso compartido. Los RDSEFF, a su vez, están construidos por

secuencia de acciones internas que generan demandas sobre los recursos `ProcessingResource`. Esto provoca invocaciones tanto del planificador asociado al `ProcessingResource` como de las funciones de acceso de los `PassiveResource`. La figura 5.10 muestra un ejemplo ilustrativo de cómo SimuCom gestiona la simulación a partir de un modelo PCM en el que una única CPU actúa como `ProcessingResource`.

SimuCom simula la evolución temporal mediante los hitos de activación temporal producidos por los eventos. Funciona de una manera similar a simuladores como *VHSIC Hardware Description Language* (VHDL) o SystemC (Grotker et al., 2002). Las activaciones temporales dependen de los tiempos de las demandas de los `ProcessingResources`, los cuales son especificados en los RDSEFF, y de los periodos de generación de los eventos `Usagescenario`. En caso de ser desalojada una demanda se genera una nueva activación temporal con lo que queda de la demanda restante.

Como se puede ver en la imagen 5.10, un `UsageScenario` genera un hilo de ejecución nuevo cada 50 unidades de tiempo (ut). El hilo generado envía un evento denominado `EventA`, el cual ejecuta una `InternalAction`. Ésta realiza una demanda al `ProcessingResource` denominado CPU, que tiene como parámetro 20 unidades de tiempo, con prioridad de ejecución 2. Esta demanda se gestiona por la *SimuCom engine Java Skeleton*, realizando las siguientes operaciones: `Queueing` pone en cola la nueva demanda; `FireStateChange` desaloja la demanda en ejecución; y `ScheduleNextEvent` planifica el siguiente componente. Todos estos métodos formarán parte de la clase que define la política de planificación. `PasivateThread` duerme el hilo hasta que la demanda es satisfecha. Una vez que el tiempo de la demanda es procesado se realiza la operación *Notify*, de tal manera que el hilo es liberado y puede reiniciar su ejecución. Una vez que todas las demandas asociadas a la reacción de un `Workload` han sido satisfechas, la reacción ha sido consumida por el sistema, siendo el tiempo transcurrido entre su inicio y final como tiempo de respuesta, lo cual es el objetivo del simulador.

5.4.2. Descripción formal del comportamiento de SimuCom

En esta sección se presenta la definición formal (Becker, 2008) de comportamiento del SimuCom con el objetivo de explicar las modificaciones necesarias para poder adaptar la nueva extensión. Se describen los elementos que entran en juego en la planificación de los procesos por parte de SimuCom. Los elementos que componen esta descripción formal son: los vectores de demandas, las

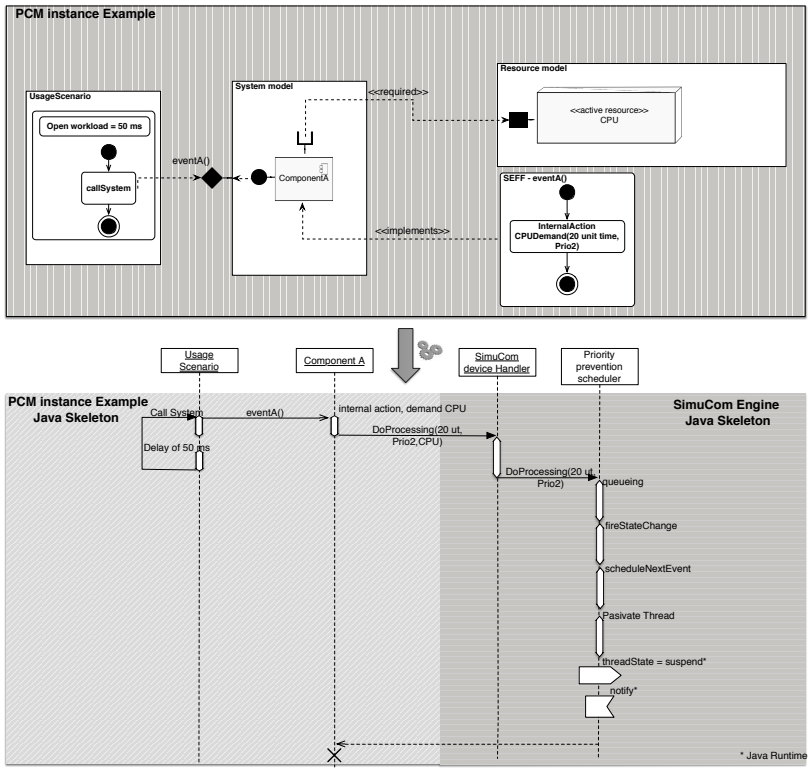


Figura 5.10: Transformación desde el modelo de componentes de Palladio al modelo de simulación SimuCom.

operaciones que modifican los vectores de demandas, los eventos que ejecutan las operaciones y, por último, la gestión de los recursos compartidos.

Definición de las colas y definición del paso de tiempo

Primero definiremos el conjunto *Demanda* como $UUID \times \mathbb{R}_0^+$. Un elemento $d \in D_n = Demand^n$ describe una demanda como una tupla con un identificador unívoco y con el tiempo restante para que la demanda sea satisfecha.

Son definidas dos notaciones abreviadas para referirnos a cada uno de los elementos de una demanda cuando forman parte de un vector. La notación $id(d)$ explícitamente se refiere al identificador de una demanda $id(d) = id$. La notación abreviada $time(d)$ expresa el tiempo restante de la demanda de un vector, por ejemplo, $d = (id, d_i), time(d) = d_i$.

Un conjunto de demandas son agrupadas en un vector de demandas. Cada uno de los vectores está asociado a un recurso, en PCM a un `ProcessorResource`. El vector de demandas es expresado como $\vec{d} = (d_1, ..., d_n) \in D_n = Demand^n$. Cada elemento d_i de un vector $\vec{d}$ es el tiempo restante para que la demanda sea satisfecha. Este tiempo restante debe ser procesado en el recurso que está asociado al vector cola. El vector está ordenado en orden de llegada $i < j$, lo que significa que la demanda d_i llega antes al recurso. El conjunto de todos los posibles vectores de la demanda es:

$$D = \bigcup_{i \in \mathbb{N}_0} D_i \tag{5.1}$$

El vector vacío y el vector arbitrario son formalizados. El elemento $D_0 = Demand^0$ denota un vector vacío, por ejemplo, vector $\emptyset$. Esto implica que existe algún vector de longitud arbitraria n, siendo ésta la longitud del vector D_m.

Se define el estado del vector como el secuente de sus elementos $< d_1, ..., d_n >$

A continuación, definimos las funciones que modifican el estado de los vectores de demandas. Se expresa la función $process : D \times \mathbb{R}_0^+ \rightarrow D'$, siendo $\mathbb{R}_0^+$ el tiempo en el cual la función *process* procesa la demanda, dando lugar a un nuevo vector. Ésta caracteriza el cambio del estado del vector con el paso del tiempo. Por ejemplo, la fórmula $process(\vec{d}, t_{Delta}) = \vec{d'}$ define el estado de un vector $\vec{d}$ después de un *timespan* de Δt como $\vec{d'}$. La definición inicial de *process* depende de la política de planificación. Por ejemplo, la definición base de SimuCom define los siguientes planificadores: *FCFS First-Come First-Serve*, *Delay* y *Processor Sharing*. Otras estrategias de planificación más modernas fueron definidas (Happe, 2009). Pero en ningún caso ha sido preparado para poder modelar políticas de planificación basadas en prioridades fijas con desalojo.

La función $NextDone$ da el tiempo restante para que una demanda sea satisfecha, siempre que el estado de la cola no cambie. Asumiendo que una función *process* es dada, definimos la función como: $NextDone : D \rightarrow \mathbb{R}$ $NextDone(\vec{d}) = max t_\Delta | t_\Delta \in \mathbb{R}_0^+ \wedge process(\vec{d}, t_\Delta) = \vec{d}$.

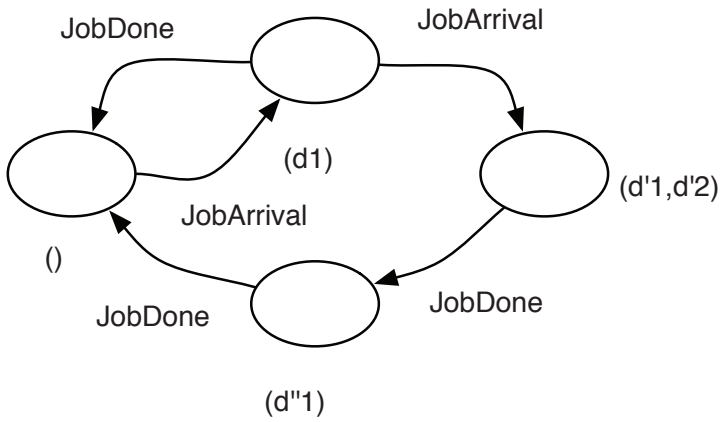


Figura 5.11: Ejemplo de un vector demanda y cómo se modifica en función de las funciones y los eventos disparados.

5.4.3. Simulación de los eventos

Dos tipos de eventos son definidos con el objetivo de definir los elementos que interactúan con el vector demanda. En la figura 5.11 están representados con flechas. Un evento es *JobArrival*, ocurre en la simulación cuando se añade una nueva demanda a la cola, y el otro evento, *JobDone*, ocurre cuando, tras una función *process*, una demanda ha sido satisfecha.

Cada evento simulado contiene el tiempo de simulación t_e. Como ha sido definido anteriormente, el estado de una cola es la tupla: $\vec{d}, t_l \in D \times \mathbb{R}_0^+$ siendo $\vec{d}$ el estado de la cola introducido en t_l, y t_l el último punto de simulación en el cual la cola ha cambiado de estado. El estado inicial para todas las colas es $(\emptyset, 0)$, es decir, la cola en el momento inicial está vacía.

La llegada de una nueva demanda se indica mediante el evento *JobArrival*, el cual notifica a un vector que una nueva demanda ha sido añadida. La cola es modificada cuando un evento *JobArrival* es recibido mediante la función *Process* acorde para ese recurso. Cuando un evento *JobArrival* es procesado en el instante t_e por un recurso, primero se actualiza su estado para reflejar el actual tiempo de simulación. Para esto se usa el último tiempo de simulación t_l

desde $(\vec{d}, t_l)$ a $(process(\vec{d}, t_e - t_l))$. Entonces se añade la nueva demanda d_{n+1} y una nueva demanda se elige para ser procesada la cola por el cambio de estado desde $(\vec{d_n}, t_e) = ((d_1, ..., d_n), t_e)$ a $(\vec{d_{n+1}}', t_e) = (d'_1, \ldots, d'_n, d'_{n+1}, t_e)$, se computa $t_{next} = NextDone(\vec{d'})$. Ahora SimuCom borra todos los eventos *JobDone*, los cuales han sido desalojados por el nuevo componente. La simulación planifica un nuevo evento *JobDone* en el instante de simulación $t_e + t_{next}$.

Si una señal correspondiente al evento *JobDone* se dispara al final del procesamiento de un componente en el instante de simulación t_e se producen los siguiente cambios en la cola de un recurso: el estado se actualiza desde $(\vec{d}, t_l)$ a $(\vec{d}, t_e) = (process(\vec{d}, t_e - t_l), t_e)$, reflejando el tiempo actual de simulación. Entonces, para toda demanda i tal que $(d_i) = 0$ es eliminada de la cola, cambiando el estado de la cola de $((d_1, \ldots, d_n), t_l)$ a $((d_1, \ldots, d_{i-1}), d_{i+1}, \ldots, d_n), t_l)$. Como consecuencia, todos los hilos que están esperando para el procesamiento de la demanda d_i retoman su ejecución. El siguiente evento *JobDone* se planifica como especificado en el evento *JobArrival*.

Finalmente expresamos $rt : ProcessingResosource \times Demand \to \mathbb{R}_0^+$, siendo $\mathbb{R}_0^+$ el conjunto de los posibles tiempos de respuesta. Esta función expresa el tiempo de respuesta de una demanda (rt) en una cola de un *ProcessingResource*. Por ejemplo, rt es el tiempo de la demanda, y el tiempo de espera en la cola. Definimos rt^d_{start} como el tiempo de simulación para la llegada del evento *JobArrival* de una demanda d, entonces $rt(pr, d) = rt^d_{end} - rt^d_{start}$. rt^d_{end} depende del progreso aleatorio, dando lugar a un resultado también aleatorio rt. El tiempo de respuesta depende del estado inicial, el estado aleatorio de la cola y todos los eventos *JobArrival* aleatorios, que ocurren durante el proceso de la demanda asociada al evento *JoDone* rt_{end}.

5.4.4. Estrategias de planificación

Actualmente SimuCom provee tres tipos de estrategias, las cuales son:

- **Round-Robin**, una aproximación de la estrategia Round Robin (Thomasian, 1998).
- **First-Come First-Served**, política de planificación que procesa las demandas por orden de llegada.

- **Delay**, todos los hilos son procesados inmediatamente,independientemente del número de recursos.

Hay que destacar que, aunque sólo se describen tres políticas de planificabilidad, la arquitectura modular de SimuCom permite integrar nuevas fácilmente. Podemos intuir en este punto cómo los elementos demanda están encapsulados. Esto implica que no sería difícil añadir nuevos parámetros sin alterar el funcionamiento de las políticas de planificación que vienen por defecto. Si queremos añadir nuevas políticas de planificación deberemos definir el orden de consumo de las demandas en el vector. Orden que puede estar basado en parámetros encapsulados en las demandas. En resumen, la estructura de SimuCom permite integrar de una manera natural nuevas políticas de planificación.

5.4.5. Descripción de las colas

El comportamiento de los recursos pasivos también es descrito de una manera formal. Una demanda sobre un recurso pasivo se describe mediante la tupla $UUIDtimeresource$. Un elemento $d_{passive} \in Demand_{passive}$ se describe como una demanda. $UUIDS$ es el identificador único del proceso que realiza la demanda y *resource* es el número de peticiones que se realiza sobre el `PassiveResource`.

Las notaciones abreviadas *id* y *resource* son definidas en orden de acceso a los elementos de un vector de demandas. Para referirse al ID, se usa la expresión $id(d) = id$. Para referirnos a un elemento *Resource*, se define el número de peticiones sobre el `PassiveResource`. Se define como $Resource \in \mathbb{N}_0^+$ el número de recursos disponibles del vector demanda.

Expresamos $\vec{d} = (d_1, ..., d_n) \in D_n = Demand^n$ como el vector que describe el estado de la cola. Cada elemento de d_i de un vector $\vec{d}$ describe una demanda de acceso al `PassiveResource`. El vector es ordenado, por ejemplo, para todo $i < j$, llegando d_j antes al recurso. El conjunto de todos los posibles vectores de la demanda es:

$$D = \bigcup_{i \in \mathbb{N}_{n_o}} D_i \tag{5.2}$$

Los elementos $\vec{d}$ son las demandas encoladas que esperan a ejecutarse en la sección crítica. El elemento *resource* indica el número de *resources* que deberán ser consumidos en orden en la sección crítica.

El tamaño de elementos que puede contener el vector demanda se expresa de una manera formal como $D_0 = Demand^0$, denotando un vector vacío, por ejemplo, vector $\emptyset$. Como consecuencia, algún vector de longitud arbitraria se define D.

Las funciones *AdquireActions* y *ReleaseAction* son las encargadas de modificar el estado de la cola. La función *AdquireAction* es la encargada de adquirir el `PasiveResource`, mientras que la función *ReleaseAction* es la encargada de liberar el recurso. La función *AdquireAction* como función principal se encarga de añadir demandas a la cola de demandas $\vec{d}$. Cuando una nueva demanda es añadida, el proceso que realiza la demanda puede ser bloqueado. Éste será bloqueado si se cumple la condición $|\vec{d}| > Resource$. En caso contrario, el proceso accedería a la sección crítica. La función *ReleaseAction* es la encargada de eliminar una demanda del vector de demanda $\vec{d}$. Cuando se realiza la operación *ReleaseAction* se despierta el proceso $d_{Resource}$.

Actualmente las demandas son encoladas en orden FIFO, de tal manera que se pueden producir fenómenos de inversión de prioridades. En la imagen 5.12 se muestra la evolución de la cola de las funciones *AdquireAction* y *ReleaseAction*. Podemos ver en este caso que el número de recursos es 1, de tal manera que la demanda $d2$ estará bloqueada hasta que no se realice una operación de *ReleaseAction*.

5.4.6. Modificaciones `SimuCom-RT`

Modificación de *Processing resource*

Para poder integrar un planificador basado en prioridades fijas con desalojo, varios elementos han tenido que ser redefinidos. Estas modificaciones añaden nuevas abstracciones, como son, prioridades, planificador de prioridades fijas con desalojo, protocolo de herencia de prioridad y protocolo de techo de prioridad inmediato.

Vector demanda

Hemos definido una notación abreviada y una función. La notación abreviada describe el valor de la prioridad de una demanda $Priority$ y la función modifica la prioridad de una demanda $ChangePriority$. La anotación abreviada indica la

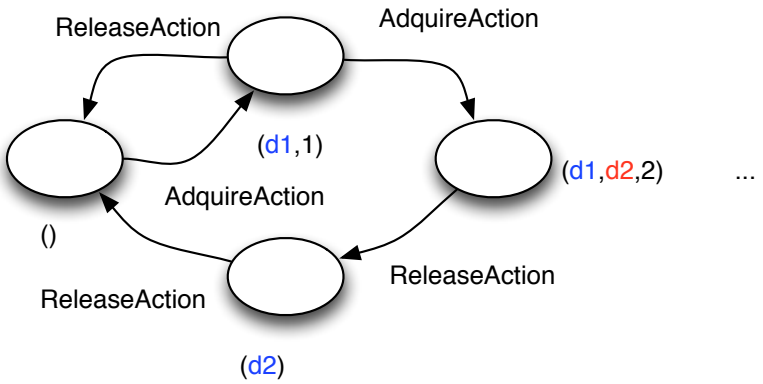


Figura 5.12: Evolución del vector de demandas de un `PassiveResource` con un *Resource* de 1.

Priority asociada a una demanda d; de esta manera tenemos $Priority(d) = P$. La función *ChangePriority* cambia el valor de la prioridad asociada a una demanda, por ejemplo, $ChangePrioriry(id, priority)$ modifica la prioridad de la demanda *id* especificada con la prioridad *priority*.

Cada vez que la función *ChangePriority* es usada, se deberá realizar una nueva planificación de las tareas. Esto se representa mediante la generación de un evento *JobArrival*.

Un nuevo elemento ha sido añadido prioridad a la definición de demanda, de tal manera que la tupla queda de la siguiente manera: (ID, D, P), siendo P la prioridad de la demanda. Esta redefinición no afecta al resto de estructuras del simulador. El contenido de las demandas sólo es usado por la función *process*, la cual, como hemos visto, implementa la política de planificación.

Nueva función *process*

Una nueva función *process* ha sido definida para expresar un planificador de prioridades fijas. Su definición formal se expresa a continuación:

Esta función tiene en cuenta el parámetro prioridad para elegir la demanda, la cual será procesada. Sólo el tiempo de la demanda más prioritaria será actualizado con el tiempo transcurrido de simulación hasta el siguiente evento *JobArrival* o *JobSone*, esto se expresa con $time(d_{min(prio)} - t_\delta)$.

$$id(d'_i) = id(d'_i) \wedge time(d'_i) = \begin{cases} time(d_{min(prio)}) - t_\Delta, & \text{if } i = (min(priority)) \\ time(d_i) = i, & \text{if } i \neq min(priority) \end{cases}$$

Figura 5.13: Definición de la nueva función *process*, que expresa una política de planificación de prioridades fijas con desalojo.

Modificación de los recursos pasivos

Como se describió anteriormente, la nueva extensión soporta dos de los protocolos de inversión de prioridad más extendidos en sistemas empotrados: como el de techo de prioridad inmediato y el de herencia de prioridad.

Al elemento demanda de recursos compartido se le añade un nuevo elemento *prioridad*. Esta prioridad será con la que se configuren las demandas sobre la CPU mientras esté capturado el *PassiveResource*. La nueva tupla quedará de la siguiente manera: $(id, resource, p)$. Una nueva función ha sido también introducida, ésta es *Priority*, que permite acceder a la prioridad de un vector demanda. Un ejemplo de su uso es $Priority(d) = p$.

Un nuevo parámetro de prioridad se añade cuando se realiza la operación *AdquireAction*. Este nuevo parámetro es la prioridad de acceso al recurso compartido. La función se redefine de la siguiente manera: $Priority \rightarrow \emptyset$.

La nueva demanda se añade al vector de demandas $\vec{d}$ en orden de prioridad, para el subvector $\vec{d} = d_{Resource}, d_m$. Esto significa que los procesos que estén esperando para acceder son planificados por orden de prioridad. Las demandas que correspondan al vector de demandas $\vec{d}$ a $\vec{d'} = d_0, d_{resource}$ pueden ver alterada su prioridad cuando es recibida una nueva demanda. La prioridad se ve alterada cuando la prioridad de la nueva demanda d_new es mayor que la que tiene las demandas $\vec{d'}$. La prioridad de los elemento $\vec{d'}$ modifica la prioridad de d_{new}. La modificación se realiza mediante la función de las colas de estado de los *ProcessingResource ChangePriority*.

La modificación permite implementar el protocolo de herencia de prioridad. El cambio de prioridad sólo afecta a las demandas que están dentro de la sección crítica, todas las demandas a un `ProcessingResource` se realizarán con la prioridad especificada en el `AdquireAction`. La prioridad de la demandas es marcada por el acceso al recurso compartido. En el caso de techo de prioridad inmediato, la prioridad especificada de acceso al `PassiveResource` será la máxima prioridad con la que se accede al recurso. En el caso de herencia de prioridad, la prioridad especificada de acceso será la misma prioridad del hilo

que accede al `PassiveResource`, pudiendo realizar un aumento de la prioridad [2] si la prioridad de la nueva demanda es mayor.

5.5. Pruebas de validación

Mediante un conjunto de pruebas se comprueba que la nueva extensión cumple con las propiedades requeridas. Con este objetivo ha sido definido un conjunto de escenarios de prueba en los que se conocen de antemano los resultados. Los escenarios y los resultados teóricos han sido tomados de Liu (2000).

Para validar esta nueva extensión han sido definidas tres pruebas. La primera está centrada en validar la política de planificación basada en prioridades fijas con desalojo, y utiliza un escenario sin recursos compartidos. Las otras dos comprueban la correcta implementación del acceso a los recursos compartidos empleando, respectivamente, los protocolos de techo de prioridad inmediato y herencia de prioridad.

Cada prueba incluye la siguiente información:

- **Descripción de componentes**, en el cuadro 5.2 se especifican elementos como tiempo de ejecución, prioridades y los periodos de activación de cada componente. Los *deadlines* de los componentes son los mismos que los periodos de activación. La asignación de prioridades se realiza mediante análisis RMS, por lo que los componentes con los periodos de activación más cortos tendrán la mayor prioridad.

- **Cronogramas**, el cronograma muestra el comportamiento teórico. Los elementos con los que se describe la ejecución de las demandas de los componentes se representan mediante rectángulos. El acceso a los recursos compartidos se describe mediante un código de colores. El color blanco es el correspondiente a la ejecución fuera de cualquier sección crítica, es decir, sin capturar ningún recurso. Otros colores y patrones representan ejecuciones en la sección crítica de los diferentes recursos. El inicio de ejecución y los cambios de contexto entre componentes son mostrados mediante diferentes líneas y flechas.

- **Gráfico con los tiempos de respuesta de cada componente**, estos diagramas muestran los tiempos de respuesta de cada componente durante una simulación de SimuCom. Para cada simulación se ha empleado un total de 20 hiperperiodos.

[2] En lengua anglosajona se emplea el termino *boosting*.

5.5.1. Prueba de planificación basada en prioridades fijas con desalojo

Estas pruebas se centran en la validación del comportamiento del planificador en términos de desalojo de componentes. En el cuadro 5.2 muestra los elementos de planificación; tres componentes con diferentes prioridades compiten por la CPU. Los resultados esperados son mostrados en el cronograma 5.14. Todos ellos tienen un comportamiento periódico y sus prioridades son inversamente proporcionales a sus periodos de activación.

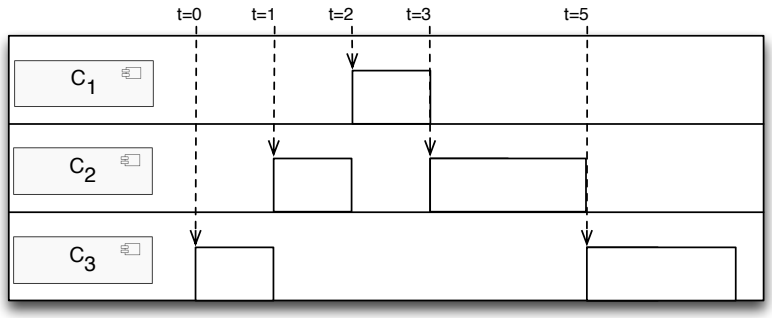


Figura 5.14: Cronograma de la prueba de desalojo.

La siguiente ecuación iterativa calcula el tiempo de respuesta t_i de cada componente C_i:

$$t_i = e_i + \sum_{k=1}^{i-1} \lceil \frac{t_i}{p_k} \rceil \tag{5.3}$$

El sumatorio sobre los componentes cuya prioridad es mayor que la del componente C_i los cuales son $1 \leq k \leq i$. Los parámetros e_k y p_k son, respectivamente, el tiempo de ejecución y el periodo de activación de dichos componentes. Finalmente, e_i es el tiempo de ejecución del componente C_i.

Los datos mostrados en la imagen 5.15 son los resultados calculados por SimuCom después de 20 hiperperiodos. Como se puede comprobar, los peores tiempos de respuesta obtenidos en la simulación concuerdan con los peores tiempos de la respuesta teórica. Los tiempos de activación han sido seleccionados de una manera intencionada para forzar los peores casos.

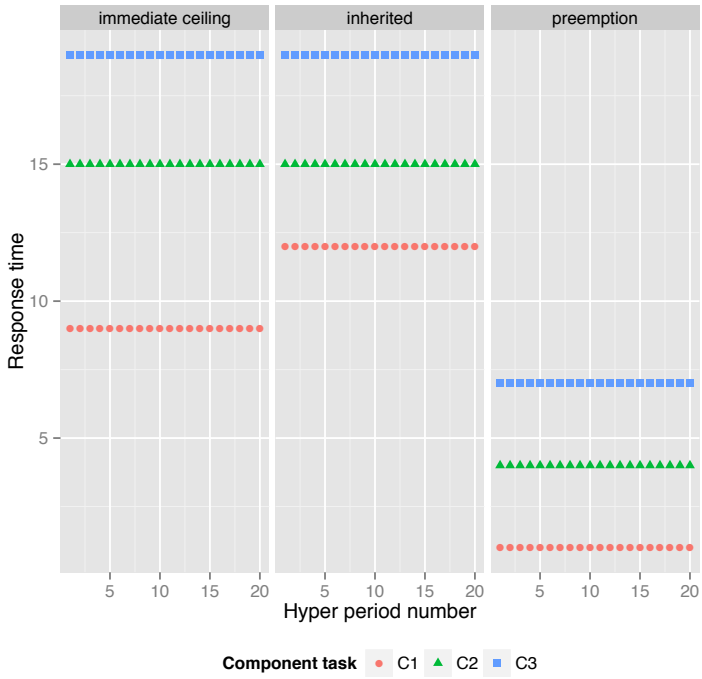


Figura 5.15: Tiempos de respuesta de las pruebas de validación.

5.5.2. Prueba del protocolo de techo de prioridad inmediato

Esta prueba se centra en comprobar el comportamiento de planificación cuando se accede a un *mutex* que tiene configurada una política de techo de prioridad inmediato. En el cuadro 5.2 muestra los parámetros para el análisis de planificabilidad. Son definidos tres componentes con diferentes prioridades que compiten por la CPU y dos recursos compartidos protegidos mediante el protocolo que quiere ser validado. La información relativa a las secciones críticas se muestra también en el cuadro 5.2 y el comportamiento teórico esperado se muestra en el cronograma 5.16.

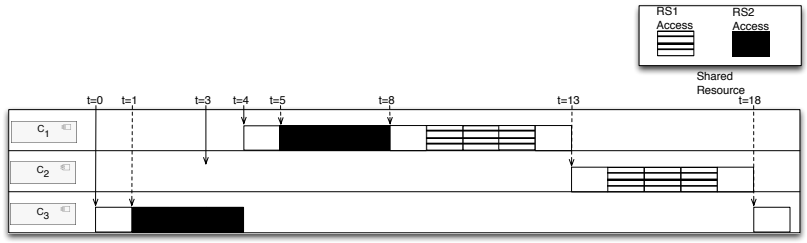


Figura 5.16: Cronograma que muestra el comportamiento de los componentes cuando el recurso ha sido protegido usando un protocolo de techo de prioridad inmediato.

La siguiente función analítica iterativa permite calcular el peor tiempo de respuesta de cada uno de los componentes (Liu, 2000; Buttazzo, 2004).

$$t_i = e_i + b_i + \sum_{k=1}^{i-1} \lceil \frac{t_i}{p_k} \rceil e_k \tag{5.4}$$

La ecuación es similar a la que fue mostrada en la sección previa, con la salvedad de la adición de algunos parámetros nuevos. Estos parámetros nuevos están relacionados con los efectos de las zonas de exclusión mutua. b_i expresa el tiempo de bloqueo sufrido por el componente C_i debido a una contención por el mismo recurso. Este parámetro se calcula para este protocolo como el mayor tiempo de acceso demandado por cualquiera de los otros componentes de algunos de los recursos accedidos por el componente C_i.

Los resultados teóricos se muestran en el cuadro 5.2. La imagen 5.15 muestra la simulación de 20 hiperperíodos. Como se puede comprobar, los resultados obtenidos por la simulación concuerdan con los tiempos obtenidos de una manera teórica.

Prueba del protocolo de herencia de prioridad

Esta prueba se centra en la comprobación del comportamiento del planificador cuando se utiliza el protocolo de herencia de prioridad. En el cuadro 5.2 muestra la información relativa a los parámetros de planificación. Han sido definidos tres componentes con diferentes prioridades que compiten por la CPU y dos recursos compartidos, los cuales también son descritos en el cuadro 5.2. Los resultados esperados son mostrados en el cronograma 5.17.

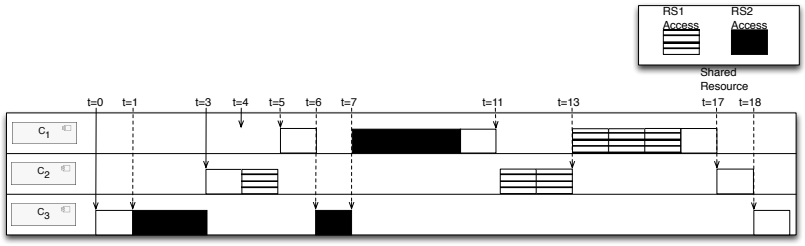


Figura 5.17: Cronograma que muestra el comportamiento de la implementación de un protocolo de herencia de prioridad.

La ecuación 5.4 muestra la función analítica iterativa para el cálculo del peor tiempo de respuesta cuando se utiliza el protocolo de herencia de prioridad. La diferencia con el protocolo de techo de prioridad inmediato es cómo se calcula el bloqueo máximo por cada uno de los componentes.

En este caso, la mayor cantidad de tiempo de bloqueo Bi corresponde al bloqueo de la sección crítica que tiene el peor WCET. Se deberá tener en cuenta que un recurso puede sólo ser bloqueado por una tarea y el anidamiento de las secciones críticas no está permitido.

A diferencia de la anterior estrategia, en ésta el mayor tiempo de bloqueo Bi corresponde a la suma de la sección crítica con el peor tiempo de ejecución. Hay que tener en cuenta que una sección crítica puede sólo ser bloqueada por una tarea en un determinado momento, y que no está permitido el anidamiento de secciones críticas.

La imagen 5.15 muestra los resultados de la extensión de SimuCom con la configuración del protocolo de herencia activo. Los resultados del análisis mostrados en el cuadro 5.3 concuerdan con los obtenidos en la simulación con SimuCom.

Cuadro 5.2: Parámetros para el análisis de tiempos de respuesta.

Componente información					Recurso compartido	
Nombre de la prueba.	Nombre.	Tiempo Ejecución(C).	Tiempo Liberación (e_i).	Prio. (p).	Recurso Crítico.	Tiempo Ejecución.
Desalojo. .	C_1	1.	2.	1.	NA.	NA.
	C_2.	3.	1.	2.	NA.	NA.
	C_3.	3.	0.	3.	NA.	NA.
Techo Prioridad Inmediato.	C_1.	8.	4.	1.	RS1;RS2.	3;3.
	C_2.	5.	3.	2.	RS2.	3.
	C_3.	5.	0.	3.	RS1.	3.
Herencia Prioridad.	C_1.	8.	4.	1.	RS1;RS2.	3;3.
	C_2.	5.	3.	2.	RS2.	3.
	C_3.	5.	0.	3.	RS1.	3.

Cuadro 5.3: Tiempos de respuesta calculados teóricamente.

Nombre de la prueba.	Nombre.	Tiempo de respuesta.
Desalojo.	C_1.	1.
	C_2.	4.
	C_3.	7.
Techo Prioridad Inmediato. protocolo.	C_1.	8.
	C_2.	14.
	C_3.	19.
Herencia Prioridad. protocolo.	C_1.	12.
	C_2.	15.
	C_3	19.

Capítulo 6

Modelo de análisis de verificación

Depurar es al menos dos veces más duro que escribir el código por primera vez. Por tanto, si tu escribes el código de la forma más inteligente posible no serás, por definición, lo suficientemente inteligente para depurarlo"

Brian Kernighan

Si McDonnalds funcionara como una compañía de software, uno de cada cien Big Macs te envenenarían, y la respuesta sería «lo sentimos, aquí tiene un cupón para dos más»

M. Minasi

6.1. Introducción

En este capítulo describimos el proceso automático de verificación, centrado en los parámetros del modelo TSAM. En la figura 6.1 mostramos cuáles son los elementos que discutimos en este capítulo. Podemos ver que a partir del modelo transaccional, en particular el modelo TSAM, se genera el modelo RapiCheck. La función de RapiCheck en el proceso es observar que un conjunto de asunciones realizadas en el TSAM se cumplen en la plataforma destino, siendo éstas corroboradas durante las pruebas de integración.

Los modelos que tratamos son de diseño y de análisis. En la figura 6.2 se muestran los modelos y las transformaciones del proceso. Podemos ver que la transformación parte de un modelo orientado al análisis TSAM, generando el modelo RapiCheck. Este modelo contiene información relativa al diseño y al análisis. Desde el modelo de diseño se genera un modelo de despliegue, del cual se extraerá una traza de ejecución. Como el modelo RapiCheck contiene ambos elementos, procesa la traza y valida que las asunciones del modelo TSAM son las correctas.

El modelo de verificación automático es un objetivo marcado en la Tesis Doctoral, en particular, el objetivo que tratamos en este capítulo corresponde a:

- «Las restricciones planteadas en el modelo de análisis de planificabilidad son corroboradas de una manera automática en el sistema final (modelo de vuelo, o modelo de ingeniería)».

El resto de este capítulo se organiza de la siguiente manera. La descripción de los diferentes perfiles de restricciones se especifica en la sección 6.3. En la misma sección se explica cómo estas restricciones pueden ser generadas automáticamente como parte de las transformaciones de código o estrategias de generación de código. Por último, en la sección 6.4 se presenta una política de instrumentación de código y sus efectos en los perfiles de tiempo de ejecución.

6.2. RapiCheck

RapiCheck es parte de las herramientas desarrolladas por Rapita System Ltd., empresa propietaria de RVS y otras soluciones para la optimización de procesos de VV de sistemas empotrados críticos en automoción, aviónica y espacio. El propósito de RapiCheck es la verificación dinámica de requisitos o contratos software en sistemas empotrados.

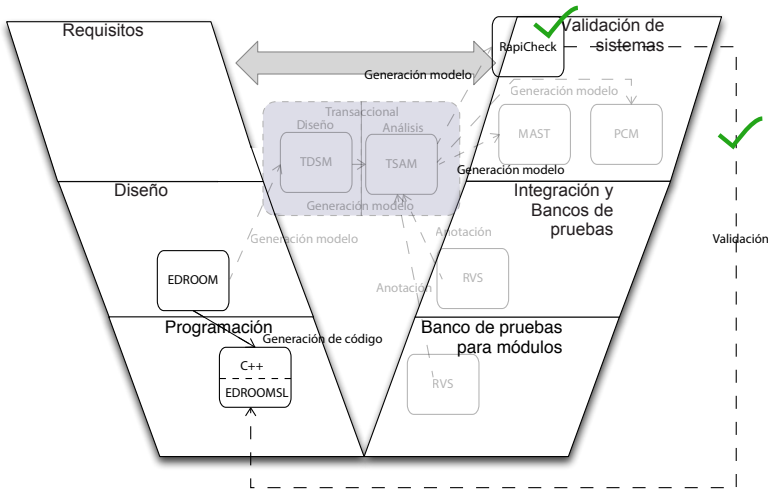


Figura 6.1: Proceso de verificación. Se muestran los modelos y el orden de los modelos que tienen que ser definidos para habilitar la transformación. Se puede ver en la figura cómo se completa la verificación del análisis de planificabilidad.

El modo de funcionamiento tiene la misma filosofía que otras propuestas de RVS RapiCheck, se basa en las observaciones realizadas en el *target* mediante instrumentación del código.

En RapiCheck se expresan los requisitos mediante AFT. Los símbolos que consume el autómata son puntos que tienen una equivalencia con elementos del código fuente. Este mapa se realiza vía instrumentación del código fuente. RapiCheck instrumenta el código automáticamente mediante la especificación de los símbolos del AFT. Puede instrumentar elementos mediante la especificación de tareas, funciones, inicio o final de función, o mediante etiquetas en el código incluidas por el usuario. Después de la instrumentación, el código es ejecutado en el *target*. Se extrae la traza y la herramienta comprueba si la traza corresponde a la gramática que acepta el AFT. Los resultados son mostrados en un informe RapiCheck, que indica las partes de la traza aceptada y los símbolos que no pudieron ser consumidos.

Dentro de las diferentes fases del proceso de VV, RapiCheck se integra en las pruebas pertenecientes a la campaña de integración, de tal manera que permite optimizar los procesos de verificación del software de dos maneras, la primera

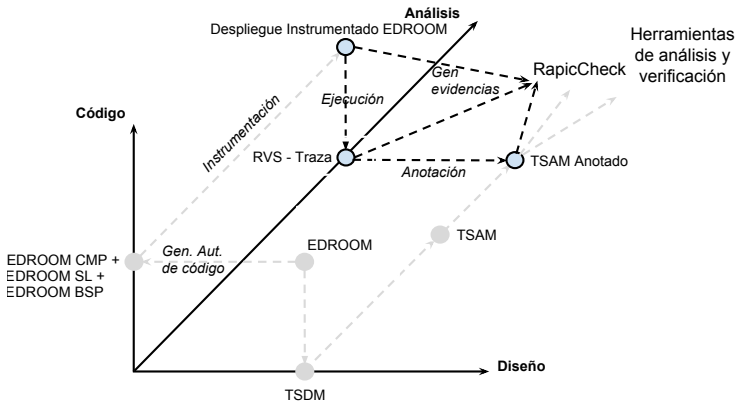


Figura 6.2: En este esquema podemos ver en qué punto de la transformación estamos y cuáles son sus coordenadas. En este capítulo sólo tratamos con modelos de validación, usando para ello la herramienta RapiCheck.

que indica que puede reducirse el número de pruebas en la fase de integración —o verificar un mayor número de requisitos con los bancos de pruebas preexistentes—, ya que, un conjunto de requisitos puede ser verificado en el mismo banco de pruebas; la segunda que permite hacer visibles errores relativos a la satisfacción de requisitos impuestos en el diseño software.

Al igual que otras propuestas de la misma empresa, se necesitan mecanismos de observabilidad en el *target*. Esto tiene una serie de limitaciones cuantitativas de los elementos que pueden ser expresados. Estas limitaciones son dadas por efectos de modificación de los *Execution Time Profile* (ETP). Son difíciles de poder cuantificar sin conocer el método de observabilidad y la sensibilidad del código que ha de ser instrumentado.

6.2.1. Definición de las restricciones en RapiCheck

RapiCheck usa un *Domain-Specific Modelling Language* (DSML) para definir los AFT que expresarán las restricciones. Las transiciones definen las restricciones temporales y los símbolos que son consumidos. Hay dos estados que tienen una función específica: el estado `Ok`, que es un estado de aceptación que ratifica

que la parte de la traza que ha sido consumida es correcta, y el estado `NoK`, el cual tiene como propósito especificar las condiciones por las cuales las restricciones no se cumplen. Las propiedades temporales se especifican en función de parámetros específicos, como el WCET.

A continuación mostramos los diferentes alcances de los que se compone la definición de una restricción en RapiCheck:

```
constraint NAME is CATEGORY

symbols
-- Los símbolos se declaran en este alcance
variables
-- Las variables son declaradas en este alcance (Éstas son opcionales)

begin
-- Los estado son definidos en este alcance
end constraint;
```

En el fragmento de código anterior mostramos un ejemplo de restricción. Cada restricción se compone de un nombre que la identifica y una categoría que permite organizarla para ayudar al usuario a visualizarla (Ltd., 2014). Los símbolos son elementos de ejecución del código fuente. Las variables permiten facilitar, como ya veremos, la definición de expresiones de cálculo de tiempos de ejecución. Por último, entre la palabra reservada *begin* y *end constraint*, se define la máquina de estados que comprobará las restricciones que se quieren verificar.

Definición de los símbolos en RapiCheck:

```
A : start "func_1";
B : task "task_1" function "func_2";
C : function "func_3";
D : task "task_2" end "func_4";
E : label "label_1";
F : task "task_3" function "func_5" label "label_2";
```

En el fragmento de código anterior mostramos un ejemplo de los diferentes elementos que se pueden definir como símbolos. Se pueden definir funciones de código fuente, tareas, funciones ejecutadas por una determinada tarea y etiquetas que son especificadas en el código.

Definición de las variables en RapiCheck:

```
variables
  delta = 0;
  period = 20;
  jitter = 2;
  deltamin = period - jitter;
  deltamax = period + jitter;
```

En el fragmento de código anterior mostramos un ejemplo de variables, que nos servirán para guardar información entre estados o simplificar las expresiones de transiciones. Por ejemplo, vemos en el código que definimos deltas, *jitter* y períodos, además se definen un delta mínimo como `deltamin` y un delta máximo como `deltamax`, donde éstos se expresan en función de otras variables. Esto nos ayudará a simplificar las transiciones asociadas a la comprobación de, por ejemplo, los delta relativos a la gestión de interrupciones.

Definición de estado y transiciones asociadas en RapiCheck:

```
  state S0 is
              when "A" => {delta = WT - S0.WT}
                  if ((delta < deltamin)  (delta > deltamax)) then
                    NOK
                  else OK;
                  end if;
              when "B" => S2;
              when "C" => {delta = 5} S3;
```

En el fragmento de código anterior vemos un ejemplo de definición de estado. El estado `S0` es el origen de cuatro transiciones, las cuales transitarán: al estado `Ok`, al estado `NoK`, al estado `S2` y al estado `S3`. Cuando se lee el símbolo A, las transiciones que se habilitan son de aceptación o de no aceptación, esto

dependerá de un condicional. A este tipo de transición se la denomina en RapiCheck como transición con condicionales. El condicional comprueba si el delta medido es mayor o menor que el especificado, en función de esto la restricción se satisface o no se satisface. Cuando el símbolo `B` es leído, se transitará al estado `S2`. Por último, tenemos una transacción que se llevará a cabo cuando se lee el símbolo `C`, ésta tiene asociada una acción, la cual redefinirá el valor de delta.

Con estos elementos se pueden definir conjuntos de restricciones. Una vez que han sido definidas, éstas son procesadas por la herramienta RapiCheck, generando las políticas de instrumentación sobre el código. Esta política es añadida al código fuente, generando una traza durante la ejecución del sistema en la plataforma de destino. La traza resultante será analizada por la herramienta en función de las restricciones especificadas, mostrando aquellas restricciones que no han sido satisfechas, especificando la porción de la traza que no la cumplió.

6.3. Perfiles de Restricciones

El proceso propuesto incluye la definición de un conjunto de perfiles de restricciones con el objetivo de verificar las evidencias en la plataforma destino. Estos perfiles verifican por un lado, el comportamiento y el WCET anotado de los modelos transaccionales (TSAM) en el modelo `TACode` y, por otro, que las transformaciones desde el modelo de diseño al modelo de análisis sean correctas. Esto implica comprobar que el comportamiento descrito en los modelos TSAM es el mismo que se observa en la plataforma de despliegue del software de aplicación. Durante las pruebas de integración se podrán activar los perfiles de verificación, eligiendo sólo aquellos que quieren ser verificados.

El proceso consta de los siguientes pasos:

1. Elección de los perfiles que son activados. [1]
2. Generación e implementación automática de las restricciones en forma de AFT RapiCheck.
3. Intervención de los eventos que quieren ser observados en el código.
4. Ejecución de las pruebas de integración en la plataforma de destino.
5. Extracción de la traza de plataforma de destino mediante un *data logger* o un analizador lógico. [2]

[1] Esta opción permite reducir la sobrecarga del proceso de verificación en el software.
[2] Tiene relación con la interacción de los modelos transaccionales y la herramienta Rapita.

6. Verificación/Contraste de las restricciones AFT de cada uno de los perfiles activados con las evidencias/traza.

Se han definido un conjunto de perfiles con objeto de verificar por un lado el comportamiento reactivo del sistema a la recepción de eventos de interrupción y temporización y, por otro lado, los tiempos asociados a la plataforma y los elementos software. Los perfiles definidos son:

- **Perfil *End-to-End-flow***, encargado de comprobar que la secuencia de disparo por un evento es la definida en los modelos `TSAM`.
- **Perfil `TSAMMessageHandler`**, comprueba que el WCET asociado a los elementos de los manejadores de mensajes `TSAMMessageHandler` es el anotado en el modelo TSAM.
- **Perfil tiempos plataforma**, verifica que el WCET y los *jitters* asociados a la plataforma y al *run-time* son los correctos.
- **Perfil de los patrones de activación**, verifica que los parámetros asociados a los patrones de activación son los especificados en el modelo TSAM. Además, también comprueba que el WCET y los *jitters* son correctos.

6.3.1. Perfil End-to-End

Este perfil tiene como objetivo verificar la secuencia correcta de `TSAMMessageHandlerItem` cuando se recibe un `TEvent`. Por cada evento se genera un elemento que es necesario verificar en forma de un AFT de RapiCheck. Los modelos transaccionales necesarios para generar este perfil son los modelos `TSAMRTRequirement`, el modelo `TSAMComponent` y el modelo FLATMCAD. Desde el modelo `TSAMRTRequirement` se extraen cada uno de los `TEvent`, y partiendo de éstos se puede extraer la secuencia de acciones con la que el sistema reacciona. El primer elemento de esta secuencia es el `TSAMMessageHandler` asociado a la tupla puerto, mensaje y componente asociado al evento. Todos los `TSAMMessageHandler` están especificados en el modelo `TSAMComponents` junto con cada `TSAMMessageHandlerItems`. Cada `TSAMMessageHandlerItem` es transformado en un estado de RapiCheck. La precedencia de los estados dependerá del orden definido en los `TSAMMessageHandler` y el modo en que son resueltos los `TSAMMessageHandlerItems` del tipo `TSAMMHiSend` o `TSAMMHiInvoke`. Para resolver

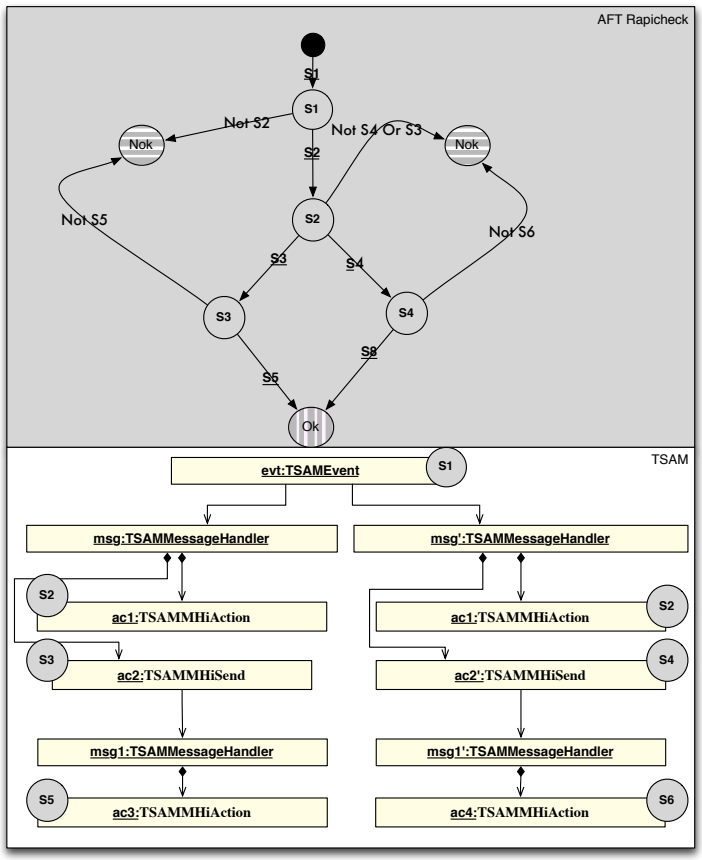


Figura 6.3: El AFT de RapiCheck generado para la verificación del WCET.

los `TSAMMessageHandlers` que son activados por los `TSAMMHiSend` o `TSAMMHiInvoke` son usadas las conexiones entre puertos, éstas son definidas en el modelo FLATMCAD.

Los tipos `TSAMMHiInvoke` y `TSAMMHiSend` generan nuevas dependencias sobre otros componentes que tienen que ser resueltas ya que estas operaciones modelan la comunicación entre tareas y generarán nuevas subsecuencias de estados en el AFT resultante de RapiCheck. Como vimos en el capítulo correspondiente a

la descripción de los modelos transaccionales, los TSAMMHiInvoke modelan las demandas sobre los recursos compartidos que no tienen una tarea propia. Por el contrario, los TSAMMHiSend modelan la comunicación por paso de mensajes entre tareas, siendo esta comunicación de naturaleza asíncrona.

El hilo que ejecuta un TSAMMHiInvoke se queda a la espera de recibir un TSAMMHiReply del componente reactivo que maneja el mensaje síncrono. Este comportamiento se resuelve primero buscando el TSAMAsynchMsgHandler que satisface la llamada síncrona y posteriormente se generan en RapiCheck dos estados: uno correspondiente al TSAMMHiInvoke y, a continuación, otro correspondiente al TSAMMHiReply. Este comportamiento se puede observar en la figura 6.3.

Los envíos asíncronos TSAMMHiSend son más complejos de resolver debido a que la secuencia dependerá ahora de las prioridades de las tareas y de la arquitectura del sistema (por ejemplo, el caso de multiprocesadores). La solución adoptada, en nuestro caso, viene de una de las restricciones de los sistemas que son soportados por este proceso. Este proceso sólo tiene en consideración sistemas monoprocesadores (ERC-32, Leon2, etc.). Esto significa que los TSAM-MessageHandler que manejan el mensaje enviado por el TSAMMHiSend y el resto de los TSAMMessageHandlerItems correspondientes al TSAMMessageHandler se ejecutarán en secuencia. La cuestión es resolver el orden de precedencia. Esto dependerá de la prioridad de cada uno de los componentes. Se han definido tres tipos de condiciones:

- **C1**, un TSAMMHiSend tiene que ser resuelto y el componente que contiene el TSAMMessageHandler es de menor prioridad, por lo que se incluye en la cola de «TSAMMessageHandler pendiente de resolver» y el resto de los TSAMMessageHandlerItem relativos al TSAMMessageHandler que contenía el TSAMMHiSend son añadidos a la secuencia del AFT.

- **C2**, un TSAMMHiSend tiene que ser resuelto y el componente que contiene el TSAMMessageHandler es de mayor prioridad. Se ejecutan todos los TSAMMessageHandlerItem asociados a este último TSAMMessageHandler, sus TSAMMHiSend se resolverán usando estas reglas de una manera recursiva, y el resto de los TSAMMessageHandlerItem que quedan del TSAM-MessageHandler que realizó la operación TSAMMHiSend son encolados en «TSAMMessageHandler pendientes de resolver».

- **C3**, todos los TSAMMessageHandlerItem del TSAMMessageHandler han sido ejecutados, por lo que debe resolverse un TSAMMessageHandler de la cola. Se extrae un elemento de la lista «TSAMMessageHandler pendientes

de resolver». Éste será el `TSAMMessageHandler` prioritario, en caso de que haya más de uno se resolverán por orden FIFO.

El funcionamiento del algoritmo es idéntico a una búsqueda en un árbol, donde cada `TSAMMessageHandler` es un nodo y el arco contiene la prioridad asociada al componente donde se deberá ejecutar el `TSAMMessageHandler`, visitando primero los nodos prioritarios. El orden de visita de los `TSAMMessageHandler` es la secuencia en la que deberán aparecer los `TSAMMessageHandlerItem`.

Un conjunto de `TSAMMessageHandler` puede estar asociado a la misma tupla puerto y mensaje. Esto significa que al menos uno de estos `TSAMMessageHandler` puede ejecutar sus `TSAMMessageHandlerItems` asociados. Éstos permiten definir que el sistema tenga diferentes estados internos y dependiendo de éstos su comportamiento sea diferente. Esta estructura se transforma en RapiCheck generando diferentes arcos, uno por cada uno de los `TSAMMessageHandler`. Los `TSAMMessageHandlerItems` de cada uno de ellos es resuelto por separado con las reglas de transformación anteriormente descritas. Se tiene en cuenta que por cada rama una lista de `TSAMMHiSend` tiene que ser resuelta y el orden de preferencia se verá alterado. En caso de que la secuencia de `TSAMMessageHandlerItem` sea la misma para dos ramas, éstas son simplificadas.

Por último, vimos como los `TSAMMessageHandlerItem` pueden definir dependencias sobre los elementos del tipo `TSAMSAP`. Por cada `TSAMSAP` requerido se definirá un estado RapiCheck. El orden de precedencia es el orden de aparición en el modelo transaccional.

6.3.2. Perfil TSAMMessageHandlerItem WCET

El segundo perfil verifica el WCET anotado en los `TSAMMessageHandlerItems` y los `TSAMSAP`. Vimos que cada `TSAMMessageHandlerItems` es anotado con su WCET en el modelo `TSAMComponent`. Por cada `TSAMMessageHandlerItem` se genera un AFT de RapiCheck. El AFT contiene cinco estados mostrados en la figura 6.4. El primero, encargado de detectar el inicio de la función asociada al `TSAMMessageHandlerItem` o `TSAMSAP`, cuyo nombre se forma con el nombre del `TSAMMHItemInit`. Otro estado es el encargado de detectar el final de la función asociada al `TSAMMessageHandlerItems` o al `TSAMSAP`, éste se denomina `TSAMMHItemFinish`. Por último, los tres restantes estados son: el estado inicial, el estado de aceptación (`Ok`) y el de no aceptación (`NoK`). La condición para que la verificación no se satisfaga y transite al estado `NoK` es que el tiempo transcurrido en el `TSAMMessageHandlerItem` desde el principio a su final sea mayor que el WCET anotado. En caso contrario se transitará al estado `Ok`.

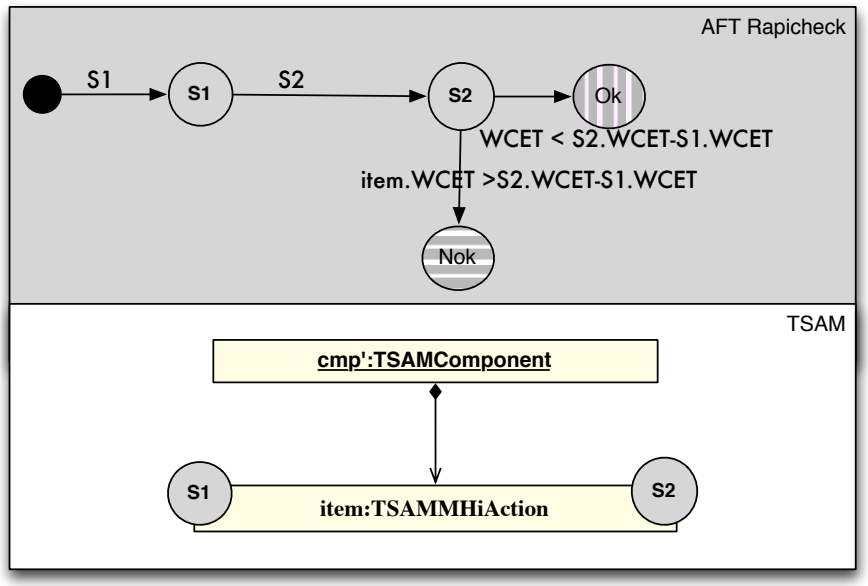


Figura 6.4: Transformación desde la definición de un `TSAMMessageHandlerItem`. En la parte superior se muestra el AFT de RapiCheck que define la restricción y en la parte inferior elementos del modelo TSAM que son usados para realizar la tranformación.

6.3.3. Perfil de *jitters* y primitivas del WCET asociado a las primitivas del *run-time* y plataforma

Los parámetros de la plataforma que afectan al tiempo de respuesta deben ser incluidos como un perfil para ser verificado, a pesar de que el modelo original no incluye el código fuente. Estos parámetros están ocultos dentro del sistema operativo. Para este perfil no hemos especificado ninguna transformación automática de AFT RapiCheck, ya que deberán ser generados manualmente. Esto es debido a que las primitivas de una plataforma a otra pueden cambiar. Aunque sería interesante investigar sólo aquellos *run-times* que satisfacen primitivas estándares, como es el caso de la API POSIX, dando la posibilidad de generar procesos automáticos.

6.3.4. Perfil de requisitos de tiempo real (*Dealine*)

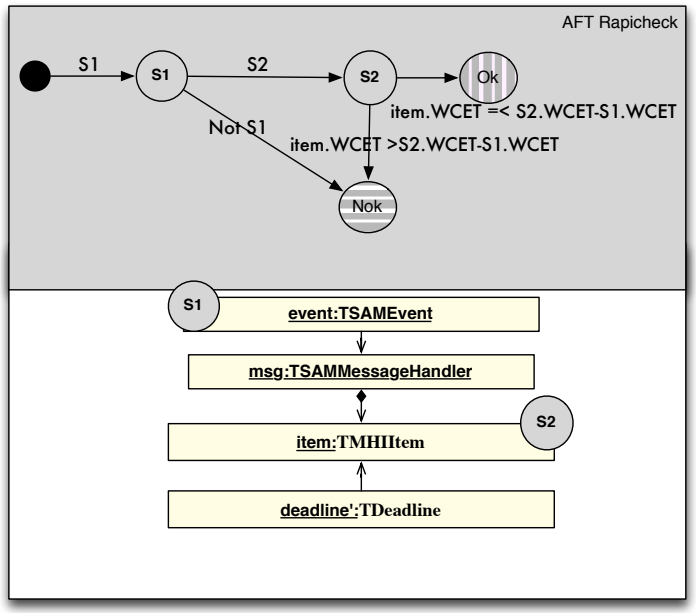


Figura 6.5: AFT RapiCheck, que verifica las restricciones de tiempo real (*deadline*).

El objetivo de este perfil es verificar que las restricciones de tiempo real (*deadline*) se cumplen. El modelo `TSAMRTRequirement` define los *deadline*. Por cada `TDeadline` se genera un AFT de RapiCheck. El tipo de AFT es muy similar al que ya se mostró en 6.3.2. Se define un total de cuatro estados, donde un estado representa la captura del evento, denominándose igual que el `TEvent` añadiendo el sufijo -*init*. Otro estado representa el final del `TSAMMessageHandler` anotándose su `deadline`, denominándose con el nombre del `TSAMMessageHandler` y el sufijo -*finish*. El resto de los estados son: el estado aceptación `Ok` y de no aceptación `NoK`. La transición al estado `NoK` se transita si el tiempo transcurrido desde el primer `TSAMMessageHandler` asociado al `TEvent` hasta el tiempo en el que se ejecuta el `TSAMMessageHandlerItems`, que contiene el *Deadline*. En caso

contrario se transitará al estado de aceptación `Ok`. El AFT se especifica en la figura 6.5.

6.3.5. Perfil de patrones de activación

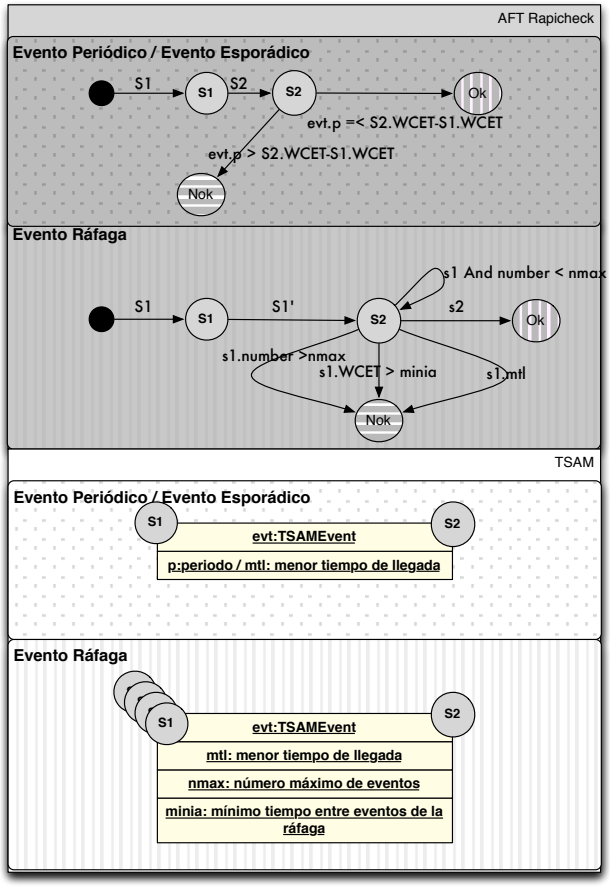


Figura 6.6: AFT RapiCheck, que verifica los patrones de activación.

Con este perfil se quiere verificar que los patrones de activación de los eventos son los especificados por el modelo `TSAMRTRequirement`. Un AFT de RapiCheck es generado por cada `TEvent`. Cada tipo de evento tiene diferentes parámetros que deben ser verificados, por lo que el AFT que se genera dependerá del tipo de evento.

Patrón de activación esporádico/periódico

Los eventos del tipo periódico o esporádico deben verificar el periodo o el menor tiempo de llegada entre eventos. El AFT resultante tiene tres estados, éste se muestra en la figura 6.6.

- Estado que describe cuando el evento es disparado. Éste se denomina igual que el `TEvent` que verifica.
- Estado de no aceptación `NoK`.
- Estado de aceptación `Ok`.

La transición desde el estado `TEvent` al estado `NoK` se realiza cuando el tiempo transcurrido entre dos `TEvent` es mayor al periodo del evento. Hay que tener en cuenta que las semillas de ambos eventos deben diferir en el campo `ramdom seed`. En caso contrario se transitará al estado `Ok`. Este comportamiento se muestra en la figura 6.6.

Patrón de activación Ráfaga (*Bursty*)

En los eventos del tipo ráfaga (*bursty*) se deben verificar los siguientes parámetros: el mínimo tiempo entre eventos, el número máximo de eventos por ráfaga y el menor tiempo entre los elementos de la ráfaga.

El AFT generado consta de cuatro estados.

- Estado de verificación de ejecución del evento. Éste se denomina igual que el `TEvent` que verifica.
- Estado de verificación de cada elemento de la ráfaga. Se denomina `InterArrivalTimeNumberActivations`.
- Estado de no Aceptación `NoK`.
- Estado de Aceptación `Ok`.

Anteriormente, aunque además de comprobar el periodo de activación de cada ráfaga, se verifica la frecuencia de llegada y el número de elementos de cada ráfaga. Para éstos se definen dos nuevas transiciones relativas al número de activaciones y el tiempo entre cada suceso de la ráfaga. Por cada nuevo evento de la ráfaga se define una transición que parte desde el estado `InterArrivalTimeNumberActivations` y tiene como destino el estado `InterArrivalTimeNumberActivations`. Siempre que se cumpla una de las siguientes condiciones se define una transición al estado `NoK`:

- Cuando el periodo del evento es mayor al especificado.
- El número de eventos es mayor.
- Cuando el tiempo entre eventos es menor al especificado.

6.3.6. Perfiles de situaciones de tiempo real

Este perfil permite verificar las situaciones de tiempo real definidas en el modelo `RTRequirement`. Ya vimos, en el capítulo 3 correspondiente a la descripción de los modelos TSAM, que cada `TSAMMessageHandler` del modelo *transactional component model* puede tener una relación de una o más situaciones de tiempo real. Esto significa que los elementos de estos `TSAMMessageHandler`, los `TSAMMessageHandlerItem`, son ejecutados sólo cuando una determinada situación de tiempo real está activa en el sistema.

Una situación de tiempo real está formada por:

- **Identificador**, nombre que describe de manera inequívoca la situación de tiempo real.
- **Punto de modulación**, formado a su vez por la situación de tiempo real de entrada y el `TSAMMessageHandler` que indica la activación de la situación de tiempo real.

De una manera ilustrativa se muestra en la figura 6.7 la definición de la restricción. Podemos ver que un AFT ha sido definido para cada situación de tiempo real. Esto permite verificar los elementos válidos para cada una de las situaciones de tiempo real. Por otro lado, otro AFT se define para comprobar que los pasos de una situación de tiempo real a otra son válidos. Con esto podemos corroborar que las acciones ejecutadas en el sistema son las especificadas en su comportamiento.

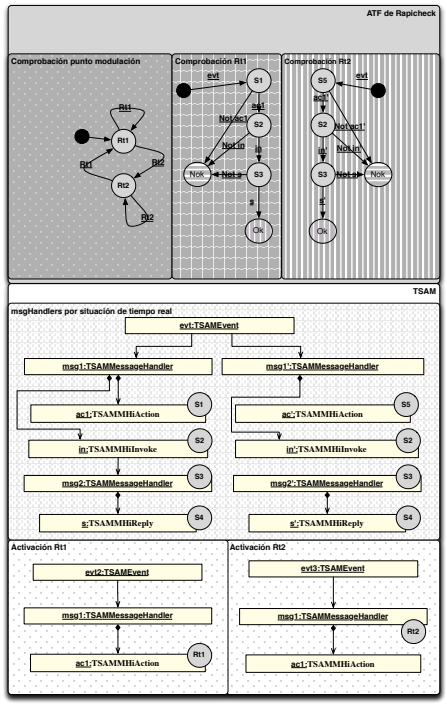


Figura 6.7: AFT RapiCheck que verifica las situaciones de tiempo real.

Se puede decir que cuando se produce un cambio de situación de tiempo real los nuevos `TEvent` sólo podrán ejecutar los `TSAMMessageHandler` correspondientes a esta situación de tiempo real. Los eventos que habían sido activados antes del cambio de la situación de tiempo real acabarán su reacción con la que empezaron su ejecución. Con este propósito se define un AFT de RapiCheck con el objetivo de comprobar que el orden de activación de las situaciones de tiempo real corresponde a los puntos de modulación definidos. De esta manera se genera un estado por cada situación de tiempo real. Los arcos de este AFT son los puntos de modulación, y para transitar de un estado a otro se deberá identificar la ejecución del primer elemento del `TSAMMessageHandler`, que permite la modulación de la situación de tiempo real.

Se deberá verificar que una vez activada una situación de tiempo real todos los `TSAMMessageHandler` son los relativos a esa situación. Para esto se genera un AFT de RapiCheck por situación de tiempo real y por `TEvent`. Pero sólo contendrá los `TSAMMessageHandler` pertenecientes a la situación de tiempo real que vamos a verificar. Los elementos de la traza generada durante la ejecución de la prueba de integración pueden contener más de una situación de tiempo real. Esto implica que deberá ser dividida en tantas trazas como situaciones de tiempo real hayan sido definidas, con el objeto de verificar cada uno de los AFT relativos a cada `TEvent` y situación de tiempo real. La división es fácil de realizar, los puntos de modulación son elementos de corte de trazas. Hay que tener en cuenta que los `TEvent` que no han finalizado su ejecución deberán ser contenidos en esta misma traza. Éstos son fáciles de identificar, ya que los identificadores `event-seed` tras el punto de corte todavía son los pertenecientes a la anterior situación de tiempo real. Una vez alimentado cada AFT se podrá comprobar si las evidencias son las correspondientes a las definidas en el modelo.

6.4. Instrumentación y sobrecarga

6.4.1. Instrumentación

Con el objetivo de demultiplexar diferentes eventos se ha definido un identificador de punto de instrumentación. [3] El motivo radica en que un mismo `TSAMMessageHandlerItem` puede ser ejecutado por diferentes eventos. Esto implica que si queremos comprobar por separado dos secuencias de ejecución asociadas a dos eventos que ejecutan un mismo `TSAMMessageHandlerItem` , la traza debe ser demultiplexada. La composición del identificador se muestra en la figura 6.8. Se compone de los siguientes elementos: [4]

- **Event-seed**, identifica la instancia de un evento. Se genera aleatoriamente y consta de 11 bits, de tal manera que la probabilidad de que dos eventos consecutivos tengan la misma semilla es de 4,882810−4.

- **Event-identifier**, identifica un evento. Está formado por un total de 4 bits, esto implica que se puede distinguir hasta un total de 16 eventos.

[3] Este punto de instrumentación difiere del que ha sido usado en la caracterización de los tiempos de ejecución.

[4] Esta distribución es sólo una sugerencia, puede modificarse de manera cuantitativa los números de bits o adaptarse a las necesidades y limitaciones del hardware.

- **Point-identifier**, identifica un punto en el código fuente. Se compone de 16 bits, por lo que se puede especificar un total de 65,536 puntos.

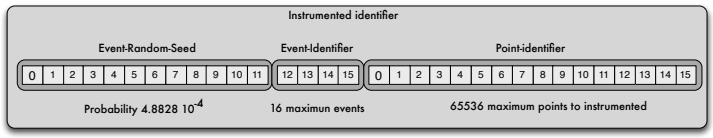


Figura 6.8: Campos que conforman un punto de instrumentación. Esta propuesta permite demultiplexar los diferentes eventos y puntos en el código fuente, de tal manera que habilita la verificación propuesta.

6.4.2. Sobrecarga

La generación de evidencias en esta propuesta está basada en la instrumentación del código. Esta instrumentación genera sobrecarga en los tiempos de ejecución, de manera que deberá tenerse en cuenta.

Es de especial interés generar la menor cantidad de elementos instrumentados para verificar todos los perfiles anteriormente citados. La política de instrumentación se realiza en conjunto con todos los perfiles que han sido habilitados, de tal manera que el número de puntos de instrumentación se optimiza globalmente. De hecho, si el perfil `End-to-End` está activo, todos los demás, a excepción del perfil `TSAMMessageHandlerItem` WCET, no generan nuevos puntos de instrumentación, por lo que no se genera una sobrecarga adicional.

Hay que tener en cuenta que la sobrecarga dependerá tanto del número de puntos de instrumentación como de la periodicidad de ejecución de un punto de instrumentación. Por ejemplo, elementos de verificación relativos a *jitters* o el cambio de contexto del sistema operativo generan una sobrecarga muy significativa en el sistema. Otros, en cambio, relacionados con la comprobación de eventos de períodos grandes generan poca sobrecarga. Esto implica que determinar la contribución en función del tipo de perfil es difícil, debido a su heterogeneidad y a la sobrecarga en los tiempos de ejecución de cada uno los de puntos de instrumentación, ya que la contribución depende de factores como la periodicidad y el tiempo de ejecución.

Uno de los aspectos más interesantes y más homogéneos para mitigar la sobrecarga es la especificación de niveles de detalle de cada uno de los perfiles. Para esto han sido definidos tres niveles de detalle.

- El nivel con más detalle que tiene en cuenta los `TSAMMessageHandlerItem`.
- El nivel de detalle intermedio que tiene en cuenta los `TSAMBasicHandler`.
- Por último, el menor nivel de detalle que tiene en cuenta los `TSAMMessageHandler`. Esto quiere decir que sólo se instrumentarán los primeros y últimos elementos del nivel de detalle y puntos de conexión con otros `TSAMMessageHandlers`, como son los `TSAMMHiSend`.

Por ejemplo, un nivel de detalle relativo a `TSAMBasicHandler` , el cual contenga un `TSAMMHiSend` se instrumenta de la siguiente manera: el inicio del primer `TSAMMessageHandlerItem` del `TSAMBasicHandler`, el primer `TSAMMessageHandlerItem` del inicio del `TSAMMessageHandlers` que es resuelto con el `TSAMMHiSend` y el final del último `TSAMMessageHandlerItem`. De esta manera, el WCET será el correspondiente a la suma de los `TSAMMessageHandlerItem` contenidos en la secuencia.

Capítulo 7

Caso de uso

La fortuna favorece a los valientes.

P. Virgilio

"Puedes tener un software de calidad o puedes tener aritmética de punteros, pero no puedes tener ambas cosas al mismo tiempo"

B. Meyer

How desperate do you have to be to start doing push-ups to solve your problems?

K. Knausgard

7.1. Introducción

El proceso de VV descrito en esta Tesis Doctoral ha sido implementado como soporte de la ICUSW.

EPD forma parte de la carga útil de la misión Solar Orbiter, cuyo objetivo científico primario es determinar *como el Sol crea y controla la heliosfera.* Dicha misión combina instrumentación de observación remota e *in-situ*, siendo EPD el instrumento encargado de la observación de partículas energéticas. EPD cubre directamente uno de los subobjetivos científicos de alto nivel: ¿Cómo las erupciones solares producen las partículas energéticas que llenan la heliosfera? El lanzamiento de Solar Orbiter está planeado para Octubre de 2018, y la misión se acercará hasta 0.28 UAs del Sol durante el perihelio. Durante parte de la órbita, la inclinación del plano orbital permitirá observar el Sol desde fuera del plano de la eclíptica. EPD estudiará las propiedades de los iones supratermales y partículas energéticas en el intervalo de energías desde unos pocos $\frac{keV}{n}$ hasta cientos de $\frac{MeV}{n}$ el rango de cada uno los sensores se muestra en la imagen 7.2. A continuación describimos someramente cada uno de los sensores[1] los cuales se muestran en la imagen 7.1.

- *Suprathermal Electrons and Protons* (STEP), observará electrones y protones supratermales entre 2 y 200 keV, utilizando imanes permanentes para la separación de electrones y protones. El sensor dispone de dos aperturas (una para electrones y otra para protones) y permitirá obtener información sobre la dirección de llegada de las partículas gracias al pixelado de los detectores.

- *SupraThermal Ion Spectrograph* (SIS), identifica las partículas mediante espectrometría de masas por tiempo de vuelo, y se basa en el diseño del instrumento *Ultra Low Energy Isotope Spectrometer* (ULEIS) a bordo de la misión *Advanced Composition Explorer* (ACE). Cubre el rango de energías entre 0,03 $\frac{MeV}{n}$ y 14 $\frac{MeV}{n}$.

- *Electron Proton Telescope* (EPT), este telescopio es una herencia del telescopio *Solar Electron and Proton telescope (SEPT)* a bordo de la misión STEREO. Fue diseñado para medir los electrones en el rango de energías de entre 20 y 400 keV *Solar Electron and Proton Telescope* (SEPT) y mide iones entre 60 y 7000 keV. Los iones son excluidos de los telescopios de electrones usando una lámina de absorción, mientras que un imán impide la entrada de electrones en los telescopios de iones.

- *High Energy Telescope* (HET), cubrirá la parte más energética del espectro observado por EPD (hasta 15 MeV para electrones y más de 400

[1] Para más información puede consultar la página oficial del grupo SRG y la página oficial de EPD hospedada por la ESA.

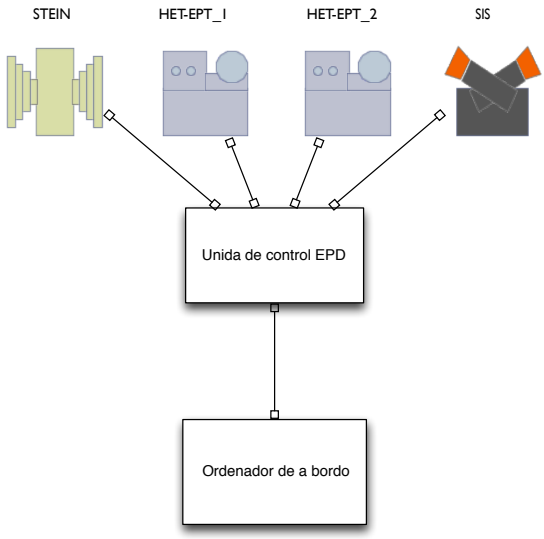


Figura 7.1: Composición de elementos del experimento EPD, embarcado en el satélite Solar Orbiter.

MeV para iones de Fe). Sus observaciones permitirán comprender el origen de sucesos de alta energía. Tanto `HET` como `EPT` cuentan con varias aperturas, de forma que permiten obtener cierto nivel de información sobre la distribución direccional de las partículas.

El instrumento consta —aparte de los sensores— de una unidad de control que permite realizar las funciones de interfaz. Esta unidad de control denominada ICU consiste en una *Control Process Data Unit* (CDPU) y una *Low Voltage Slow Control* (LVPS). Las funciones de interfaz del ICU son relativas a la comunicación entre los sensores de EPD y el ordenador de abordo de la misión (*spacecraft*). De una manera más detallada su funcionalidad es:

- Controlar la comunicación entre los sensores y el ordenador de abordo.
- Controlar la comunicación entre el ordenador de abordo y EPD.

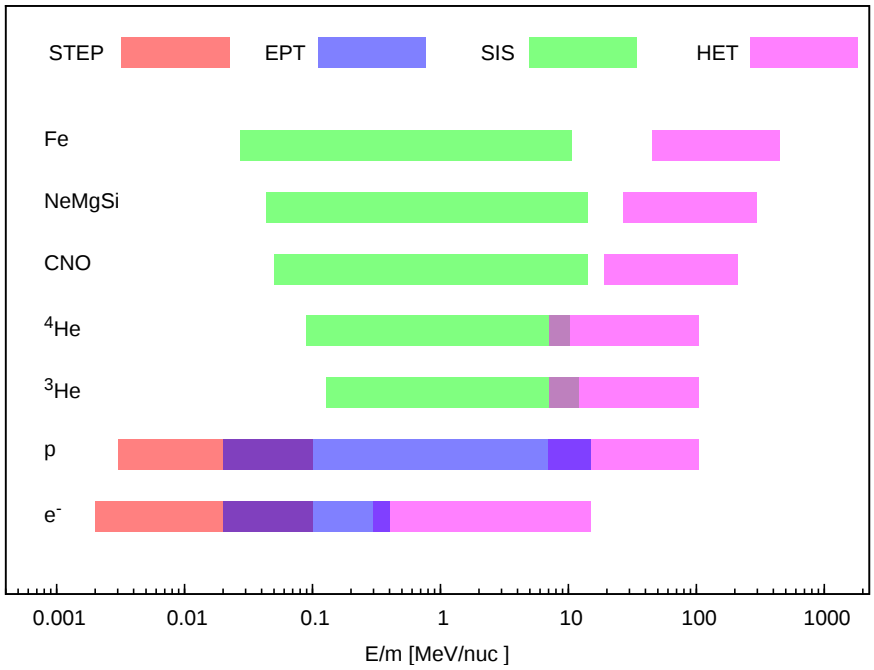


Figura 7.2: Rango de energías de los diferentes sensores que forman parte del instrumento EPD.

- Comprobar los datos recibidos de los sensores.
- Ejecutar los comandos recibidos de los sensores.
- Transmitir la telemetría de ciencia, recibida desde los sensores al ordenador de abordo.
- Comprobar los datos recibidos desde el ordenador de abordo de la misión.
- Ejecutar los telecomandos de la ICU recibidos por parte del ordenador de abordo.
- Reenvío de telecomandos a los sensores por parte del ordenador de abordo.
- Proporcionar los servicios PUS asignados a EPD.

- Gestionar las transiciones entre los modos de EPD.
- Gestionar los casos de contingencia de EPD.
- Gestionar el `Housekeeping` de EPD.

Este capítulo se estructura de la siguiente manera. Veremos la arquitectura de ICUSW donde su organización interna está formada por sus interfaces, el comportamiento de todos sus elementos, las relaciones entre ellos y el papel de estos elementos en la interacción entre la ICUSW y los sensores o el ordenador de abordo. En la sección 7.3.2 se especifica la transformación a los modelos pivotes, donde se muestran los modelos TSDM y TSAM resultantes de la transformación del modelo EDROOM de ICUSW. En la sección 7.4.1 se muestra la transformación al modelo de planificabilidad de la ICUSW con una serie de resultados previos. En la sección 7.4.2 se muestra la transformación al modelo PCM y una serie de resultados preliminares. En la sección 7.4.3 se muestra la transformación de una serie de modelos de verificación `RapiCheck` y un conjunto de valores iniciales.

7.2. Software de la unidad de control ICUSW

Las diferentes perspectivas con las que se conforma el software de la unidad de control, tienen en cuenta:

- Modos de funcionamiento.
- Organización interna.
- Interfaces.
- Comportamiento de sus elementos.
- Las relaciones entre ellos y el papel de estos elementos en la interacción entre la ICUSW, los sensores y el ordenador de abordo de la misión.

Trataremos cada uno de estos aspectos de una manera breve pero los más exhaustiva posible.

7.2.1. Modos de funcionamiento de ICUSW

Las funcionalidades de la unidad de control pueden cambiar en función del estado en la que se encuentre. Cuatro modos principales han sido definidos, un modo de configuración donde el instrumento se encuentra en fase de calibración, un modo nominal donde los servicios de ciencia y *Housekeping and FDIR* —salud y acciones de contingencia— son las principales funciones — la gestión de *House Keeping Fault and Detection Isolation and Recovery* (HKFDIR) no se explicará más adelante, esta decisión ha sido tomado por una cuestión de alcance del caso de uso—, un modo singular de ciencia, en el que un suceso científico de gran relevancia está sucediendo, y por último un modo degradado que tras una situación no recuperable del software principal —nosotros los denominamos software de aplicación— es ejecutado, donde la funcionalidad del instrumento se reduce al envío de diagnósticos de salud, y servicios de memoria que permite cargar un nuevo software de aplicación.

Modo nominal

Éste es el principal modo en el que se encuentra ICUSW. Todos los servicios están activados, y se recoge de una manera regular la telemetría generada por los sensores. Los datos de los instrumento se mandan con una cadencia de un segundo. Un hardware específico ha sido diseñado con el propósito de liberar de la gestión de la recepción de los paquetes de telemetría procedentes de los sensores. Este hardware captura los paquetes y los encola en un *buffer* en la memoria principal.[2] El software, mediante un algoritmo de compresión, reduce el nivel de detalle de los datos, enviando al ordenador de abordo una información de ciencia con una resolución de 10 segundos

Modo evento singular de ciencia

El modo relativo al evento singular de ciencia (*burst mode*), como su propio nombre indica, es relativo a sucesos de gran relevancia científica. En este modo es de especial interés obtener más información de lo normal, ya que datos con mayor nivel de detalle pueden ser de gran valor científico, pero la activación de este modo no puede perdurar debido a las limitaciones en el ancho de banda de la misión. Por este motivo ha sido ideado este modo. Este modo es activado por interrupción mediante un mecanismos de detección de eventos singulares. Cada instrumento comparte información con el resto y cada instrumento decide, qué

[2]Fue propuesta una estrategia de codiseño (Sánchez et al., 2013).

es bajo su prisma, un evento singular de ciencia y cómo se recolectarán los datos generados durante el evento. Este modo tiene una limitación, de tal manera que no podrá usarse más de 10 minutos diarios.

El disparador (*trigger*) de este modo es un telecomando propio de la misión definido en el estándar CCSDS que denominaremos de una manera figurada telecomando 42 — 42 es un número perfecto para responder al modo singular de ciencia, el sentido de la vida, del universo y todo lo demás—. La función principal de este telecomando es compartir información científica entre los instrumentos de Solar Orbiter. Cada instrumento puede definir una serie de umbrales para activar un modo de evento de ciencia singular basándose en la información generada por otros instrumentos (ver Figura 7.3). Este proceso consta de cuatro pasos:

1. Periódicamente, cada instrumento envía cuatro bits de información al ordenador de a bordo.

2. El ordenador de a bordo genera el telecomando 42. La información de este telecomando contiene los datos proporcionados por cada uno los instrumentos de la misión.

3. Telecomando 42 se difunde a cada uno de los instrumentos de la misión.

4. Los instrumentos de acuerdo a la información contenida en el telecomando 42 pueden activar este modo.

La desactivación de los modos se lleva a cabo o bien, porque las condiciones del evento han terminado, o se ha agotado el tiempo en este modo. Esto implica:

- Recibir un telecomando 42 que no reúne las condiciones.

- Recibir una excepción del reloj asociado al tiempo máximo por día para la activación del modo de evento singular de ciencia, el cual se programa con un máximo de 10 minutos diarios.

Modo degradado o modo de arranque

Este modo tiene una doble funcionalidad:

- Gestionar el arranque de la unidad de control.

- Gestionar un modo degradado de funcionamiento.

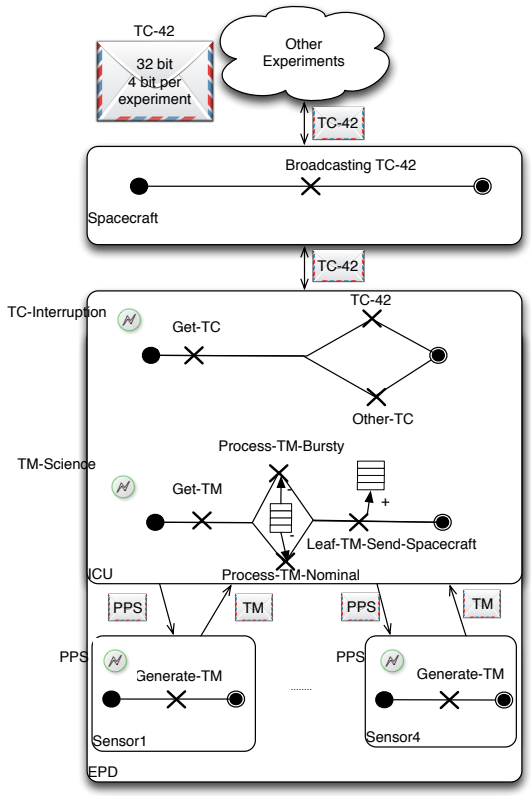


Figura 7.3: Descripción del funcionamiento del telecomando 42. Para su descripción ha sido usado el formalismo Use Case Map (UCM). Podemos ver cómo cada unos de los instrumentos envía los 4 bits. El ordenador de abordo compone el telecomando 42 y dependiendo de la información que contenga ICUSW activa el modo de evento de ciencia singular.

Cuando la unidad de control arranca este modo, realiza una serie de comprobaciones en el hardware, y si todo está correcto, despliega el software de aplicación en la memoria. Una vez desplegado el software le cede el testigo de la ejecución, este software no se volverá a ejecutar hasta que el sistema se reinicie.

Este modo gestiona los errores no recuperables, tanto software como hardware. Ejemplos típicos de los errores que producen que este modo tome el control son:

- Las restricciones temporales no se cumplen. Son detectadas mediante temporizadores (*watchdog*).
- El software de aplicación se corrompa. Uno de los motivos de las corrupción de memoria es debido a la radiación del entorno.
- La monitorización de los datos de la ICU no corresponde con parámetros de funcionamiento normal.

Cuando un error es detectado por uno de los motivos que hemos citado se realiza la siguiente secuencia de acciones:

1. El sistema se autoreinicia.
2. Se envía el motivo del reinicio a tierra.
3. Se comprueba condiciones del sistema.

Tras comprobar las condiciones del sistema dependiendo del resultado este modo tomará dos opciones:

- `Modo de recuperación`, el sistema no está habilitado para activar el software de aplicación. Esto implica que de manera periódica se envían a tierra datos de monitorización del estado de la ICU, y que se quede a la espera de una actualización del sistema.
- `Modo arranque`, el sistema proyecta en memoria principal la ICUSW —software de aplicación—.

7.2.2. Interfaces

Las relaciones de ICUSW con el hardware, los sensores y el ordenador de abordo de la misión se describen mediante interfaces. Estas interfaces son introducidas en los próximos párrafos.

Interfaces hardware y de comunicación

Esta interfaz permite la gestión de los recursos hardware. Los elementos que la componen son de naturaleza diferente según la tecnología. La ICUSW tiene una serie de memorias. La *Programmable Read-Only Memory* (PROM) que almacena el modo degradado/arranque. La memoria *Electrically Erasable Programmable Read-Only Memory* (EEPROM) encargada del almacenamiento del software de aplicación. La memoria *Synchronous DRAM* (SDRAM) que se utiliza para la proyección del software de aplicación con sus distintos segmentos.[3] Otras interfaces son las relativas a las comunicaciones. La interfaz `UART/LVDS`, que es la encargada de la comunicación con los sensores. También está la interfaz `SpaceWire` es la interfaz encargada de la comunicación con el ordenador de abordo, se encarga de la recepción de los telecomandos donde la interfaz `TMTC` controla un DMA que se encarga de gestionar la recolección de telemetrías y telecomandos de las anteriores interfaces y depositarlas en sendos *buffers* para que sean consumidos.

Se usan distintos protocolos para la comunicación con el ordenador de abordo y los sensores. Mientras que para los sensores ha sido desarrollado un protocolo específico para la misión, denominado *Sensor Transport Frame* (STF), para el ordenador de abordo se usa un protocolo estándar de la ESA, como CCSDS (CCSDS, 2000). El propósito del protocolo STF es reducir la complejidad de los servicios *Packet Utilization Standard* (PUS), como hemos visto anteriormente, son implementados por la ICU y configura los sensores mediante STF para llevar a cabo las funcionalidades específicas a nivel de instrumento.

Estructura del software

El software de aplicación ICUSW está estructurado en diferentes capas. Cada capa es independiente del resto, y el intercambio de información entre ellas se realizan a través de interfaces definidas de manera formal y clara. La figura 7.4 muestra la estructura de capas del software de aplicación ICUSW, donde cada capa sólo implementan funcionalidades, que son especificadas en EDROOM mediante librerías de servicio.

- `RTEMS 4.8`. Esta primera capa corresponde al sistema operativo de tiempo real. El sistema operativo en cuestión es el RTEMS 4.8 versión calificada por la empresa Edisoft. Esta capa tiene la funcionalidad de proveer los

[3] Segmentos de pila, segmentos de datos, y segmento de código, el código en binario tiene una estructura sparc.

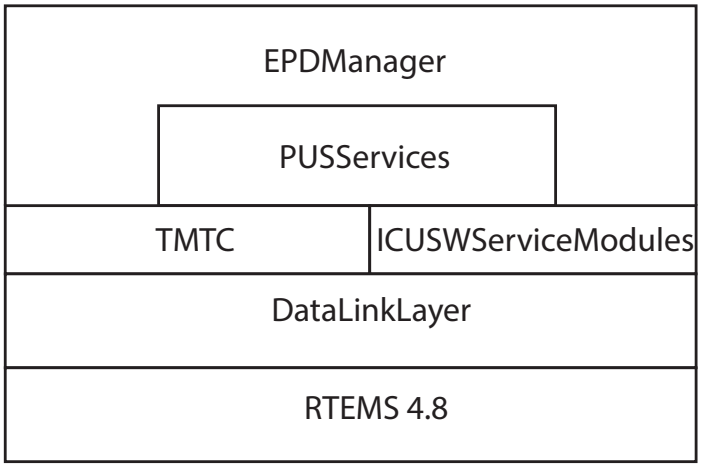


Figura 7.4: Estructura en capas del software de la ICUSW.

servicios de gestión de tareas, y servicios de programación de temporizadores, también, proporciona soporte para la instalación, gestión y uso de los controladores de dispositivos. Como son las *A Universal Asynchronous Receiver/Transmitter* (UART) o el `SpaceWire` RTOS.

- `Capa de enlace de datos (`*`DataLinkLayer`*`)`. La capa de enlace de datos es la encargada de proporcionar la abstracción para el módulo de software `TMTC` de las diferentes tecnologías de enlace de datos, que se utilizan para comunicar la ICU tanto con los sensores, como con el ordenador de abordo de la misión.

- `Telecomando y Telemetría (TMTC)`. Realiza las transformaciones pertinente y genera los datos de las cabeceras según cada protocolo.[4] También tiene la función de enrutamiento de paquetes, esto es, la capacidad que desde el segmento de tierra se mande un telecomando específico a un determinado sensor.

- `Módulos de servicio ICUSW (`*`ICUSWServiceModules`*`)`. Este módulo representa un conjunto de librerías de miscelánea que dan soporte a diferen-

[4]Especialmente CCSDS, donde se deberán tener contadores a nivel de instrumento como son contadores de paquetes, etc.

tes funcionalidades de la ICUSW. Soporte de protocolos de comunicación, soporte de comprobación de errores en memoria y su reparación. Soporte de detección de errores en alguna tecnología de comunicación. Soporte de gestión de actualizaciones del software de aplicación. Soporte a funcionalidades de alto nivel, como servicios de salud (*Housekeeping*) servicios de gestión de telemetría o ayuda a la gestión de la ejecución de los diferentes telecomandos.

- `Capa superior del software de aplicación (EPDManager)`. Es la encargada de la definición de componentes, y gestiona un conjunto de funcionalidades como un sistema multihilo. La estructura de esta capa se define por un conjunto de componentes (tareas) soportados por un CRT. La comunicación entre los componentes se realiza por medio de mensajes. La capa CRT ofrece este servicio de comunicación por medio de un puerto y una función de envío. CRT convierte interrupciones y excepciones en los mensajes enviados y en cola a los componentes. El diagrama de componentes de la capa del Administrador de EPD se muestra en la figura 7.5. En esta capa nos centraremos de una manera más extensa.

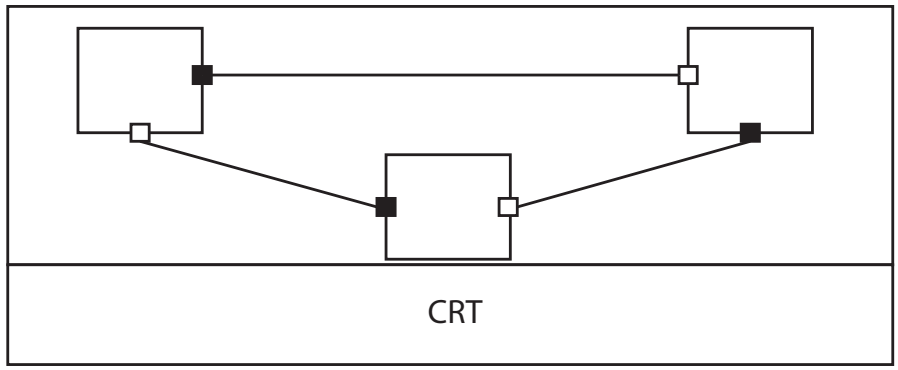


Figura 7.5: Estructura de componentes del ICUSW. Modelo de defección de componentes de EDROOM.

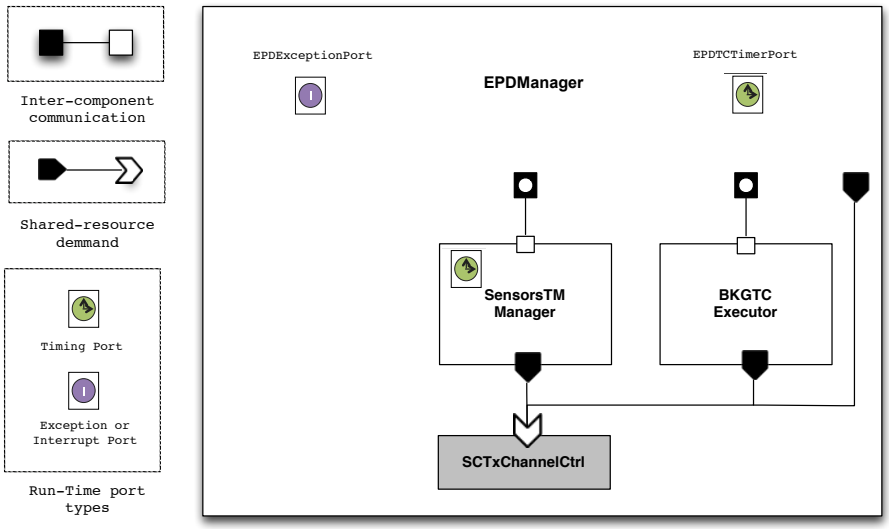


Figura 7.6: Estructura de componentes del ICUSW. Modelo de definición de componentes de EDROOM.

Capa superior del software de aplicación

La composición de la estructura de la ICUSW es mostrada en la figura 7.6. `EPDManager` es el componente principal y se compone de cuatro componentes: Tres componentes del tipo tarea y un componente del tipo recurso compartido. Los componentes tareas se denominan: `EPDManager`, `SensorTMManager` y `BKTC-Manager`. El componente recurso compartido se denomina *SCTxChannelCtrl*. A continuación detallamos el rol de estos componentes:

- `EPDManager`, componente que de manera periódica se encarga de gestionar los telecomandos recibidos por el instrumento, ejecutando los más prioritarios, y reenviando los telecomandos no prioritarios a otros componentes, dependiendo del servicio solicitado. También se encarga de gestionar los eventos críticos, incluyendo los relacionados con las excepciones software de la ICU.
- `SensorTMManager`, componente que, de manera periódica, recupera la telemetría de los sensores de EPD y los reenvía al ordenador central a través

del recurso compartido `SCTxChannelCtrl`. Otra de sus funciones es la detección y notificación al componente `EPDManager` de los eventos críticos asociados a la telemetría de los sensores. Por último, se encarga también de ejecutar los telecomandos asociados a los servicios de ciencia.

- `BKGTCExecutor`, componente que recibe los telecomandos que tienen que ejecutarse en segundo plano (*background*).
- `SCTxChannelCtrl`, este recurso compartido tiene como objetivo controlar la transmisión del *buffer* de paquetes CCSDS desde la ICU al ordenador central de abordo de la misión, asegurando que la tasa de generación de datos científicos se mantenga por debajo de los límites establecidos por los requisitos. El resto de componentes acceden a este recurso para encolar los paquetes de telemetría. La transmisión posterior de los paquetes se realiza mediante hardware dedicado.

La arquitectura dinámica de la ICUSW reside en los componentes de la capa de gestión de `EPDManager`. Cada componente se implementa mediante una tarea que coopera para resolver la funcionalidad del software de aplicación.

7.2.3. Protocolos de comunicación

Seis protocolos de comunicación definen la comunicación entre componentes —de los cuales tres definen comunicación entre componentes y otros tres definen eventos externos—. Como ejemplo tomamos el protocolo de gestión de telecomandos y el protocolo gestión de telemetría. Los dos son del tipo interrupción. El primero se encarga de la gestión de las interrupciones asociadas a la recepción de los telecomandos. El segundo es el asociado a las interrupciones relativas a la telemetría.

Los componentes suscritos a estos protocolos son: `EPDManager` y `SensorTM-Manager`. `EPDManager` atiende a la gestión de telecomandos mediante este puerto. `SensorTMManager` se ocupa de la recolección de la telemetría, su preparación, y el posterior envío de ésta a tierra.

Los componentes `EPDManager` y `SensorTM_Manager` utilizan puertos de temporización, de esta manera, `EPDManager` y `SensorTMManager` recolectan los telecomandos y telemetrías de una manera periódica.

Los protocolos de comunicación son definidos en función de los mensajes que pueden ser enviados y recibidos. En la imagen 7.7 podemos ver la definición de un protocolo asíncrono, que describe el envío de telecomandos (`SSensorTMTC`) y la recepción de eventos (`SEPDEvent`). En la imagen 7.8 podemos ver la definición de

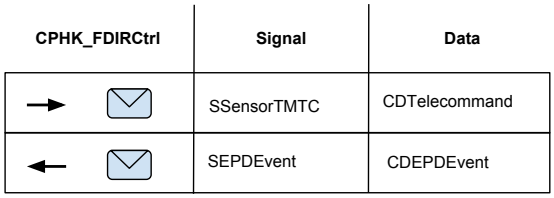


Figura 7.7: Protocolo de comunicación asíncrono. Expresa el reenvío de telecomandos como mensaje de salida, y como mensaje de entrada a la recepción de un evento.

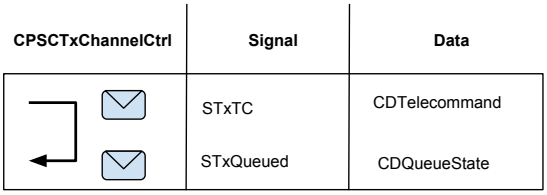


Figura 7.8: Protocolo síncrono de acceso al recurso. Define el acceso al recurso compartido `SCTxChannelCtrl`. El mensaje de entrada es la telemetría que quiere ser depositada, el componente devolverá un *reply message*, confirmando que la telemetría ha sido almacenada correctamente.

otro protocolo, del tipo síncrono, que define el acceso al recurso compartido. La definición consta del envío de un mensaje de entrada que contienen la telemetría (`STxTC`) y el mensaje de respuesta que indica que la telemetría ha sido encolada correctamente (`STxQueued`).

La comunicación entre componentes se realiza mediante la suscripción a protocolos de comunicación asíncronos, esta suscripción se materializa por parte de los componentes en forma de puertos. El acceso a los recursos compartidos se realiza mediante la suscripción a protocolos síncronos. Un ejemplo lo tenemos en los tres componentes del `ICUSW` (`EPDManager`, `SensorTMManager` y `BKGTCExecutor`) que acceden al recurso compartido `SCTxChannelCtrl`. Esta comunicación cubre la funcionalidad del envió de telemetría que genera EPD al ordenador de abordo de la misión.

Ambos tipos de puertos son mostrados en la imagen 7.6, donde se puede ver la topología de componentes y cómo se comunican.

7.2.4. Comportamiento de los componentes

Gestión de los telecomandos

EDROOM usa máquinas de estados basadas en el formalismo de Harel. Ahora veremos la descripción de tres de los componentes que conforman esta capa alta de la ICUSW, y que como hemos visto es la encargada de la definición del comportamiento.

Las transiciones encargadas de la ejecución del telecomando de sincronización con el reloj de la misión, son las denominadas telecomando prioritario. A continuación se explican las transiciones que son afectadas.

En la figura 7.9 se muestra el comportamiento de `EPDManager`. La máquina de estados define todas las acciones de comportamiento que tienen que ser realizadas por el componente. Estas tareas han sido agrupadas por los estados: `Ready`, `Telecommand`, `ModeControl` y `RebootPrograMed`. El modo de comportamiento queda encapsulado en el estado `ready`, la ejecución de los telecomandos queda encapsulada en el estado `Telecommand`, la gestión de cambios de modo queda encapsulada en el estado `ModeControl`, y el último estado indica que el sistema será degradado por un error no recuperable.

La llegada del mensaje `NextTelecommand` dispara diferentes alternativas de transiciones que parten de la transición `GetNextTc`. Estas alternativas dependen de los puntos de decisión: `Validatelecommand`, `TypeTelecomand` y `ValidExecution`. Se definen diferentes alternativas que dependen del telecomando que se recibe y el estado interno del ICUSW. Los tipos de telecomandos son agrupados en dos tipos: los telecomandos prioritarios (tienen *hard deadlines*) y los no prioritarios. A continuación describimos las diferentes alternativas:

- «El telecomando no es válido», se ejecuta la transición `TCNack`, éste tiene un `msgHandlerItem` del tipo `invoke`, denominado `FSendServTM1`, que tiene como objetivo depositar telemetría en el componente `SCTxChannelCtrl`. Esta telemetría contiene el motivo por el cual el telecomando no es válido.

- «El telecomando es válido y prioritario», se ejecutan las transiciones `AcceptTC`, `CheckType` y `ExecPrio`. Se ejecutan todos los `msgHandlerItem` correspondientes a estas transiciones, los cuales son: un `msgHandlerItem` del tipo `Action` encargado de la ejecución del telecomando prioritario, denominado `FExecPrio`; es un envío de la telemetría que contiene la información de la ejecución del telecomando, uno del tipo `invoke`, denominado `FSendTC_TM` que deposita el error en el recurso compartido `ScTxCahnnel-`

Ctrl. A partir de este punto más transiciones son ejecutadas, correspondiendo al cambio de modo.

- «El telecomando es valido no prioritario y corresponde al servicio de ciencia», las transiciones que se ejecutan son: Acceptc y TCToSensorMng. Se ejecutan todos los msgHandlerItem correspondientes a estas transiciones, como es el caso del tipo Send FForwardToSensorMng, que es el reenvío del telecomando al componente SensorTMManager, lo cual producirá la activación del componente y su consumo.

- «El telecomando es valido, no prioritario y corresponde a un telecomando de Background», Acceptc, TCToBKExec. Se ejecutan todos los msgHandlerItems correspondientes a las transiciones: getNextTC, AcceptTC, CheckTCType y TCToBkExec. La única diferente al caso anterior es la correspondiente a la transición del tipo Send FForwardToBKexecutor. Ésta describe el envío del mensaje al componente BKTCManager, el cual producirá la activación del componente BKTCManager y el consumo del mensaje.

- «El telecomando prioritario en su ejecución ha producido un error», Las transiciones que son ejecutadas son: GetNextTxmAcceptTC, CheckTCType, ExecPrio y TCExecNotCompleted. Los msgHandlerItems son parecidos al punto 1 pero con la salvedad que los msgHandlerItems asociados a la transición TCExecNotCompleted difieren de la transición TCExecOk, no en el tipo sino en la telemetría que se deposita en el componente SctxChannelCtrl. Esta telemetría contendrá la información del motivo del error a la hora de la ejecución del telecomando.

Gestión de las excepciones

La gestión de las excepciones está centralizada en el componente EPDManager. El resto de los componentes cuando detectan un error definen un msgHandlerItem de naturaleza Send que envía un mensaje al componente EPDManager, el cual atenderá la excepción, dependiendo del error el sistema se recupera o se reinicia.

Esta situación se define en la máquina de estado del componente EPDManager. Donde las transiciones que toman parte en la descripción del comportamiento son Event, NoCritical y HandleException. Los estados son Ready y RebootProgramed. Estos elementos se muestran en la imagen 7.9.

El comportamiento es el siguiente:

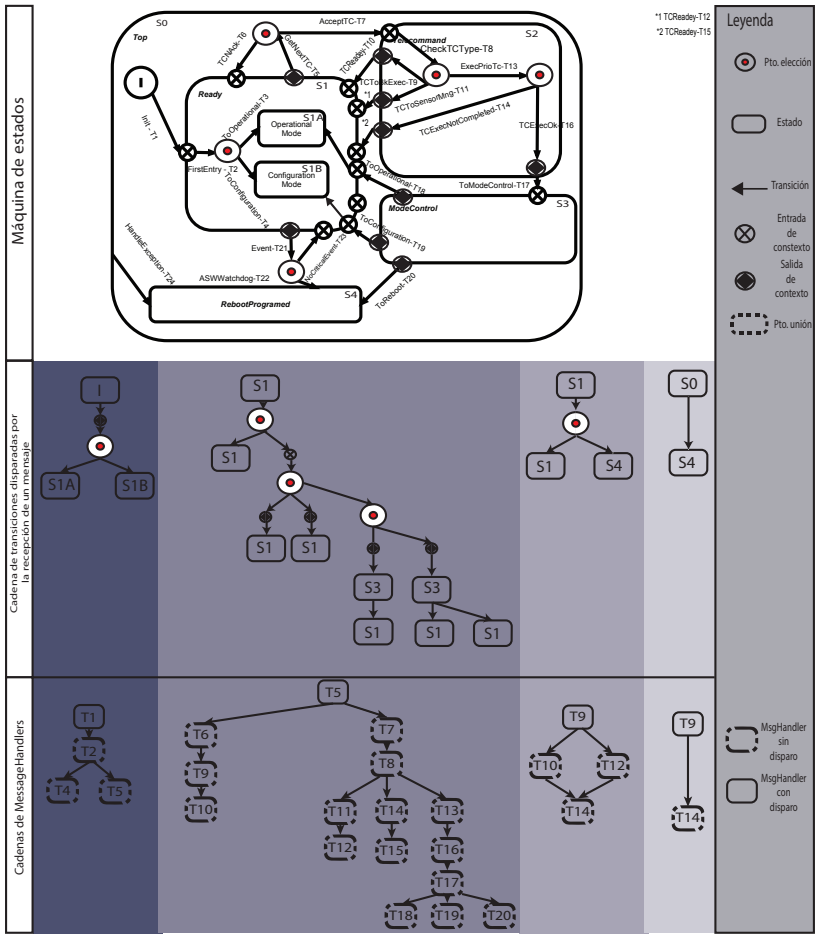


Figura 7.9: Comportamiento del componente `EPDManager`. Se representa mediante una máquina de estados basadas en la formalización de Harel, el desglose de las diferentes transiciones que son recorridas cuando un mensaje es recibido y el modelo FCM —en concreto mostramos las cadenas de `msgHandlerWithTrigger` y de `msgHandlersWithOutTrigger`.

- «Excepción hardware», una anomalía hardware ha sucedido y el disparador asociado a la transición `HandleException` ha sido disparada. En este

caso el error no es recuperable y el sistema se reinicia para que el software degradado tome el control de la ICU.

- «Event», un mensaje procedente de uno de los componentes ha detectado una situación de anomalía, una transición del tipo guarda comprobará la naturaleza del error, y si éste es del tipo recuperable se tomará la transición `NoCriticalEvent`, en caso contrario un evento crítico ha sucedido y se tomará la transición `ASWWatchdog` que indica que se fuerza el reinicio mediante una excepción relacionada con el `watchdog`.

Gestión de la telemetría

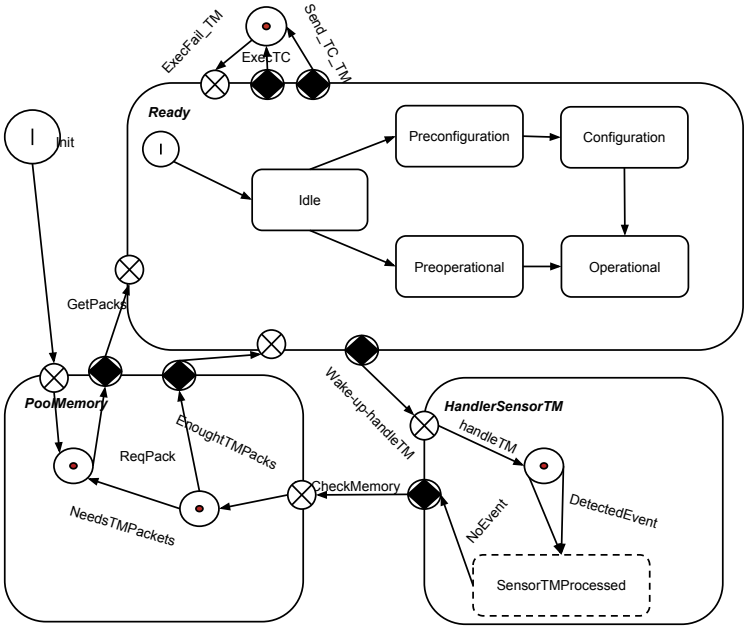


Figura 7.10: Comportamiento del componente `SensorTMManager`. Se representa mediante una máquina de estados basadas en la formalización de Harel.

SensorTMManager, es el encargado de la gestión de la telemetría, telemetría generada por los sensores. La máquina de estados del componente SensorTM-Manager se muestra en la imagen 7.10. Más en concreto esta funcionalidad es:

- Ejecución de los telecomandos del servicio de ciencia.
- Monitorización de los sensores.
- Gestión de la memoria interna de la ICU asociada a los paquetes de ciencia buffering.

Estas funcionalidades han sido encapsuladas en diferentes estados como son: Ready, HandlerSensorTM y PoolMemory.

Las transiciones encargadas de la ejecución de los telecomandos de ciencia son las transiciones: ExecTc, Send_TC_TM y ExecFail_TM. Dependiendo del punto de decisión CheckExecutionTC pueden ocurrir dos posibilidades:

- «La ejecución del telecomando es la correcta», se ejecuta el msgHandler-Item FExecTC, y una operación invoke a SendTC_TM sobre el componente ScTxChannelCtrl en el cual se deposita la información de la ejecución del telecomando.
- «La ejecución del telecomando es inválida», se ejecuta el msgHandler-Item FExecTC, y una operación invoke SendTC_TM sobre el componente ScTxChannelCtrl en el cual se deposita la información de la ejecución del telecomando, y el motivo del fallo.

Las transiciones y estados encargados de la gestión de la telemetría son los estados: HandlerSensorTM y Ready. Las transiciones como podemos ver implicadas son Wake-up-handlerTM, HandlerTM, noEvent, detectedEvent y finalmente CheckMemory. Estos elementos tienen en cuentas las siguientes posibilidades:

- «Se recolecta la telemetría sin ninguna incidencia», la secuencia de las transiciones es Wake-up-handlerTM, HandlerTM, noEvent. Como su propio nombre indica se recolecta de cada *buffer* correspondiente de cada sensor la telemetría, se procesa mediante un algoritmo de compresión de información científica, posteriormente, se deposita en el recursos compartido SCTxChannelCtrl. Para llevar esto a cabo la transición HandlerTM contiene dos msgHandlerItem, uno de naturaleza DataMsgHandler denominado FHandlerPPSIRQ y otro de naturaleza Invoke FHandlerSensorTMUnit. El primero es el encargado de recolectar la telemetría y encapsularla y el segundo de depositarla en el recurso compartido SCTxCahnnelCtrl.

Componente	Evento	Patrón de disparo	Periodo o tiempo de inter-arrival mínimo
`EPDManager` (Synch tc)	`TCRetrieving`	Ráfaga	1000 ms. (min.arrival) + 10 (max. event.) + 1 ms. (inter)
`EPDManager` (Service tc)	TCRetrieving	Ráfaga	1000 ms. (min.arrival) + 10 (max. event.) + 1 ms. (inter)
`EPDManager`	`ICUException`	Esporádico	100.000 s.
`SensorTM_Manager`	`SensorTMRetriving`	Periódico	10000 ms.

Cuadro 7.1: Patrones de disparo de los eventos ICUSW.

- «Se recolecta la telemetría con errores», la secuencia de las transiciones es la misma que hemos visto anteriormente, con la excepción de `NoEvent` por `DetectedEvent`. Ésta se produce cuando hay un error en la gestión de los *buffers* de los sensores, esta rama tiene un `msgHandlerItem` de naturaleza `Send`, que envía un mensaje al componente `EPDManager` para la gestión de las excepciones software.

Por último esta máquina de estados también define la gestión de los *pooles* de memoria reservados para cada uno de los sensores. Esta reserva, como se puede ver en la imagen 7.10, se realiza en la inicialización y cada vez que han sido gestionados los paquetes de telemetría.

Caracterización del análisis temporal

Una vez que el diseño de alto nivel ha sido definido, y previo paso a la construcción del modelo de análisis, es necesario caracterizar los tiempos de ejecución de cada acción. Además se deberán definir los patrones de activación que inician las secuencias de acciones. La información relativa a los tiempos de ejecución pueden ser obtenidos a través de los bancos de pruebas unitarios de cada una de las acciones, los patrones de activación pueden ser derivados de los requisitos. En el caso de los eventos periódicos asociados a los puertos de temporización, su patrón de activación es el periodo por el cual se programa el temporizador. De la misma manera, podrán ser configurados todas las restricciones temporales (*deadlines*) y la cadencia de los datos de entrada y salida manejados. El cuadro 7.1 resume los patrones de activación de los eventos asociados a las excepciones y los puertos de temporización de los componentes. Esta información ha sido extraída de las especificaciones de la ICUSW.

El conjunto completo de propiedades extrafuncionales relacionado al tiempo de respuesta del sistema, requiere una asignación apropiada de prioridades por componente. Específicamente, para el ICUSW las siguientes prioridades (P) han sido establecidas: $P(\texttt{EPDManager}) < P(\texttt{SensorTMManager}) < P(\texttt{BKGTCExecutor})$.

7.3. Transformación al modelo transaccional

7.3.1. TSDM

Los estados de `EPDManager` son categorizados en función de su contexto. Los tipos de estados anteriormente descritos para `EPDManager` quedarán de la siguiente manera: como estados `noLeaf`, `Ready`, `ModeControl` y `telecommand`, y como estados `Leaf OperationalMode`, `configurationMode` y `rebootMode`. Esto nos facilitará la identificación de los tipos de transiciones del componente `EPDManager`.

Se disparan cuatro tipos de transiciones del tipo `MsgHandlerWithTrigger`: la encargada de la gestión de telecomandos y cambio de modo de comportamiento de la ICUSW, la de manejo de excepciones software, la de manejo de excepciones hardware y la perteneciente a transiciones iniciales. Cada una de ellas son disparadas por una tupla $(puerto, mensaje)$ distinta.

La primera cadena de transiciones es la denominadas como gestión del telecomando. El `msgHandlerWithTrigger`, que es el primer elemento de la cadena se denomina `GetNextTC`, éste corresponde al tipo `FromNoLeafFreeExitPoint`. El resto de la cadena está formado por elementos que pertenecen a `msgHandlerWithOutTrigger`, éstos son: `acceptTC`, `checkTCType` y `ExecPrioTCCheck`, del tipo `decisionPoint`; `CheckTCTypeEntryPoint` del tipo `fromEntryPoint`; `ExecPrioTC`, `TCToBkExec`, `TCToSensorMng` y `TCExecNotCompleted` y `TCExecOk` del tipo `branch decisionPoint`. Esta cadena es más extensa, pero descartamos la parte que corresponde a los message handler de cambio de modo, ya que creemos que está fuera de esta Tesis Doctoral por una cuestión de extensión.

La segunda cadena es disparada por el `msgHandler` asociado a `Event`, del tipo `FromNoLeafFreeExitPoint`, la cadena tiene dos alternativas, la llegada a la ejecución de los `msgHandlerItems` de memoria, asociados a tipos `decisionPoint`.

La tercera es la asociada al manejo de excepciones hardware. Está formada por una sola transacción que es `HandleException`, la cual es del tipo `EdgeToState`.

Por último, la cuarta es la asociada al mensaje de inicio en la máquina de estado. Ésta es dispara por: el `msgHandler Init`, formado por una transición `FromEntryPoint`, que se denomina `FirstEntry`; y un par de decisión points denominados `ToOperational` y `ToConfiguration`.

Los estados de `SensorManager` son categorizados en función de su contexto. Los tipos de estados anteriormente descritos para `SensorManager` quedarán de la siguiente manera:

- `NLeafStateFlatDescriptor` `Ready`, `pooleMemory` y `HandleSensorTM`.
- `LeafStateFlatDescriptor` `Init`, `Preconfiguration`, `Preoperational`, `Configuration` y `Operational`.

Esto nos facilitará la identificación de los tipos de transiciones del componente `SensorManagerTM`.

Hay nueve cadenas de `msgHandler`, éstas empiezan por los `message handler`: `ExecTC`, `HandleSensorTM`, `Init`, `InitReady`, `ToConfiguration`, `ToOperational`, `ReadyToConfiguration`, `ReadyToOperational` y `ConfToOperationa`.

El `messagehandler ExecTC` es del tipo `FromNoLeafFreeExitPoint`, genera dos posibles `decisionPoints`: el `msgHandler HandleSensorTM` es el primer elemento de una extensa cadena de `msgHandler`, que tienen en consideración elementos del componente `HandlerSensorTM` y el componente `PoolMemory`.

Por último, dentro del contexto del estado `Ready` hay diferentes `msgHandlers`. Estas cadenas están formadas por un elemento del tipo `MsgHandlerWithTrigger`, los cuales son: `ToConfiguration`, `toOperational`, `ReadyToConfiguration`, `ReadyToOperational` y `ConfToOperational`.

El componente `ScTxChannelCtrl` sólo tiene un estado del tipo `leaf`. Siendo este estado `Ready`.

Han sido definidas tres cadenas de `msgHandlers`: `FreeAllocateMemory`, `QueueuTM` y `QueueuTMList`. Cada una de estas cadenas están compuesta por un solo message handler, que corresponde al `messageHandlerWithTrigger`.

7.3.2. TSAM

En esta sección definimos los modelos correspondientes al modelo TSAM de la ICUSW. Estos están formados por:

- Modelo de componentes. Se denomina como el modelo TSAMComponent.
- Modelo de protocolos de comunicación. Se denomina como el modelo TSAMProtocol.
- Bibliotecas de servicio. Se denomina como el modelo TSAMSL.
- El modelo con los tiempos de ejecución. Se denomina como el modelo TACode.
- Definición de los eventos externos y restricciones temporales. Se denomina como el modelo TSAMRTRequirement.

- Definición de los tiempos de ejecución de la plataforma. Se denomina como TSAMPlatform.

TSAMComponent

Los componentes de la ICUSW se transforman automáticamente desde los descritos en el modelo EDROOM al modelo TSAM. Por cada uno de los componentes EDROOM se genera su equivalente TSAMComponent. Esto implica que por cada uno de los siguientes componente EDROOM se genera su correspondiente componente TSAM:

- `EPDManager`, este componente es de tipo proactivo.
- `SensorTMManager`, este componente es de tipo proactivo.
- `SCTxChannelCtrl`, este componente es de tipo recurso compartido.
- `BKGTCExecutor`, este componente es de tipo componente reactivo.

La reacción de cada TSAMComponent de la ICUSW se describe en función de sus `TSAMMessageHandler`. Para ilustrar la transformación describimos la transformación de los elementos que describen la gestión del telecomando de ciencia. Los `TSAMComponent` involucrados son `EPDManager`, `SensorTMManager` y `SCTxChannelCtrl`. En la imagen 7.11 mostramos los `TSAMMessageHandler` de cada componente. Cada `TSAMMessageHandler` está compuesto por un conjunto de `TSAMMessageHandlerItems`, que a su vez tienen una relación a un modelo TACode el cual contendrá las propiedades extrafuncionales —en el caso de esta Tesis Doctoral este modelo está anotado con los tiempos de ejecución—. Por ejemplo, para el caso de la gestión de los telecomandos de ciencia, el componente `EPDManager` tiene un `TSAMMessageHandler` para su gestión, este está formado por un total de cuatro `TSAMMessageHandlerItem`, de los cuales el `TSAMMessageHandlerItem` denominado `FGetNextTC` está asociado a un TACode que tiene un WCET de 120 ms. Esta información es la que nos permitirá reconstruir una transacción completa con la información relativa al WCET.

Las situaciones de tiempo real tienen que ser anotadas manualmente con la información extraída de los requisitos. En este caso ilustramos —de idéntica manera como hemos hecho anteriormente— con el ejemplo de gestión de telecomandos de ciencia, la definición de los *tags* de situaciones de tiempo real de la ICUSW para las situaciones de eventos de error. En la imagen 7.11 podemos ver estas dos situaciones, donde la gestión de errores se puede modelar con estrategias similares a las mostradas en la sección 4.4.1.

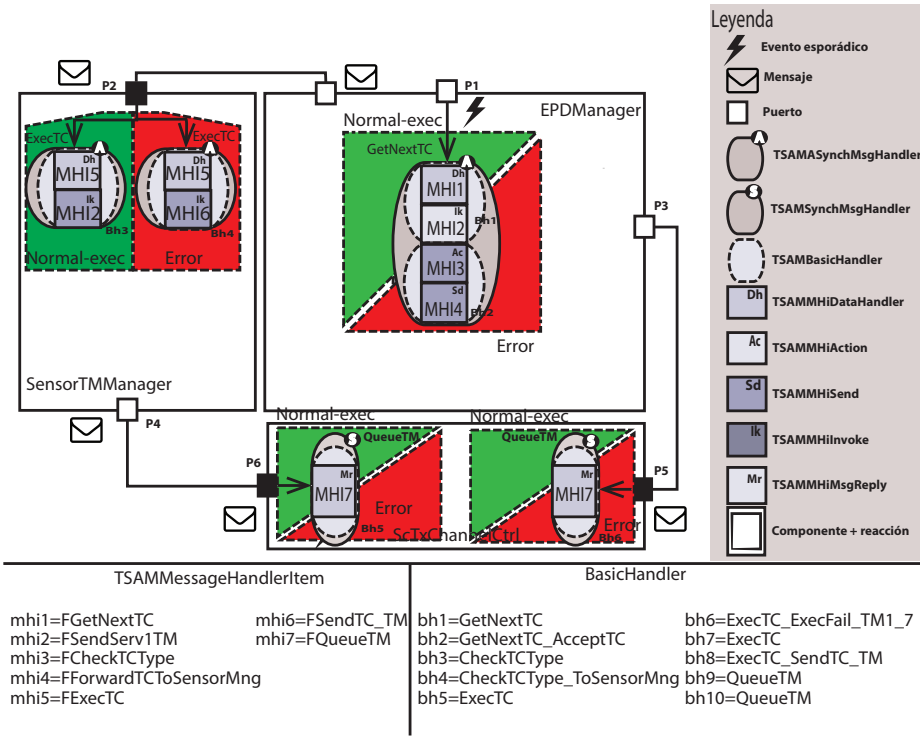


Figura 7.11: TSAMMessageHandler relativos a la gestión de los telecomandos del servicio de ciencia.

TSAMProtocol

Por cada uno de los protocolos definidos en el modelo EDROOM se define su equivalente al modelo TSAM. A continuación describimos la transformación del protocolo encargado del envío de telecomandos de ciencia y el protocolo encargado de depositar la telemetría en el componente `SCTxChannelCtrl`. Éste estará formado por protocolos equivalentes a los definidos en 7.2.3. Donde tendremos dos protocolos:

- El protocolo encargado de la gestión de los telecomando de ciencia. Este protocolo es del tipo asíncrono y contiene dos mensajes uno de entrada

y otro de salida, el mensaje de entrada es `SSensorTMTC` y el mensaje de salida es `SEPDEvent`.

- El protocolo encargado de depositar la telemetría en el componente `SCTx-ChannelCtrl`. Este protocolo es de naturaleza síncrona y está formado por dos mensajes uno de entrada denominado `STxTC` y otro de salida `STx-Queued`.

TSAMSL

Las capas que hemos visto en la sección 7.2.2 que no pertenecen al software de aplicación y al sistema operativo —RTEMS 4.8— son modeladas como TSAMSL. Su definición y transformación no forman parte de este trabajo, por lo que pueden ser consultadas en (Nilas, 2014). Habrá un TSAMSL para cada una de las siguientes librerías:

- `PUSServirce`.
- `TMTC`.
- `ICUSWServiceModules`.
- `DataLinkLayer`.

TACode

Por el momento sólo unos pocos tiempos han sido obtenidos mediante análisis de WCET usando la herramienta RVS. Por este motivo algunos de los modelos TACode han sido generados manualmente y otros de una manera automática. La generación manual de estos modelos corresponde a estimaciones basadas en presupuestos manejados por el responsable del software de la ICU el Dr. Óscar Rodríguez Polo. En el cuadro 7.2 mostramos el WCET asociado a la gestión de los telecomandos de ciencia.

Cuadro 7.2: WCET de la gestión de los comandos de ciencia.

TSAMComponent	`TSAMMessageHandlerItem`.	WCET (ms).
`EPDManager`	FGetNextTC	120 ms.
	FSendServ1TM	1 ms.
	FCheckTCType	20 ms.

continúa

continúa

TSAMComponent	`TSAMMessageHandlerItem`.	WCET (ms).
	FForwardTCToSensorMng	0.34 ms
	FExecTC	23 ms.
	FSendTC_TM	0.11 ms.
	FQueueTM	1.0 ms.

TSAMRTRequirement

Este modelo no tiene transformación automática como ya vimos en su capítulo correspondiente 3. En este modelo definimos los eventos externos y las restricciones temporales. En el caso de la ICUSW generamos al menos un evento externo para cada evento definido en los requisitos —estos se mostraron en el cuadro 7.1—. Los TEvent se muestran en el cuadro 7.3, donde podemos observar que para la gestión de los telecomandos han sido definidos hasta seis eventos con diferentes situaciones de tiempo real. Cada situación corresponde a la gestión de un tipo de telecomando y el éxito de éste en ser ejecutado —que no se generen excepciones durante su ejecución—.

Cuadro 7.3: Eventos definidos en el modelos TSAMRTRequirement pertenecientes a la ICUSW. En la columna TEvent se especifica las situaciones de tiempo real a las que está asociado el evento —estas son separadas por el símbolo dos puntos—.

TEvent	`Tipo de TEvent`.	Parámetros.
`SensorTMManager`:Nominal:Modo-evento-singular-de-ciencia:normal-exec	Periódico	10000 ms.
`EPDManager`:Nominal:Synch-tc:Modo-evento-singular-de-ciencia:normal-exec	Ráfaga	1000 ms. (min.arrival) + 10 (max. event.) + 1 ms. (inter)
`EPDManager`:Nominal:Sensor-tc:Modo-evento-singular-de-ciencia:normal-exec	Ráfaga	1000 ms. (min.arrival) + 8 (max. event.) + 1 ms. (inter)
`EPDManager`:Nominal:bk-tc:error	Ráfaga	1000 ms. (min.arrival) + 8 (max. event.) + 1 ms. (inter)

continúa

continúa

TEvent	`Tipo de TEvent`.	Parámetros.
`EPDManager`:Nominal:Synch-tc:Modo-evento-singular-de-ciencia:error	Ráfaga	1000 ms. (min.arrival) + 10 (max. event.) + 1 ms. (inter)
`EPDManager`:Nominal:Sensor-tc:Modo-evento-singular-de-ciencia:error	Ráfaga	1000 ms. (min.arrival) + 8 (max. event.) + 1 ms. (inter)
`EPDManager`:Nominal:bk-tc:normal-exec	Ráfaga	1000 ms. (min.arrival) + 8 (max. event.) + 1 ms. (inter)
`SensorTM_Manager`:Nominal:Modo-evento-singular-de-ciencia	Periódico	10000 ms.

Además una restricción temporal ha sido definida para la gestión de los telecomando prioritarios. Ésta es de 300 ms. La restricción es sobre el `TSAMMessageHandler` denominado `ExecPrioTC`, el cual se encarga de ejecutar el telecomando prioritario.

En este modelo también se definen las situaciones de tiempo real. Por cada evento se especifica que `TSAMMessageHandler` son los correspondientes a esa situación de tiempo real. Siguiendo el mismo ejemplo de la gestión del telecomandos de ciencia, hemos definido dos eventos para la gestión de los telecomandos de ciencia, uno que denominados error y otro normal-exec.

TSAMPlatform

Este modelo se genera de un manera manual con la información provista por el hardware o mediante caracterización. La plataforma definida en el caso de uso ha sido especificada en los requisitos de la ICUSW. Esta plataforma está definida principalmente por el uso del sistema operativo RTEMS, y un *System on Chip* (SoC) basado en: `Leon2`, `spacewire`, *Direct Marketing Association* (DMA), y varias UART, habiendo eliminado de su características la memoria caché, de tal manera que las propiedades de *composability* y *compositionaly* sigan presentes en el despliegue del sistema completo.

7.4. Modelos de análisis generados

En esta sección mostramos los tres resultados de la transformación desde el modelo TSAM a modelos de análisis específicos. Estos modelos son:

- Modelo de análisis de planificabilidad, modelo MAST.
- Modelo de análisis de rendimiento, modelo PCM.
- Modelo de análisis de verificación, modelo RapiCheck.

7.4.1. Modelo de análisis de planificabilidad

Mostramos la transformación entre el modelo TSAM —de la ICUSW— y su modelo equivalente en MAST, a este nuevo modelo lo denominaremos `ICUSW-MAST`. Con este propósito se aplican las reglas mostradas en el capítulo 4 para generar este nuevo modelo de análisis. La transformación de la ICUSW se aplican en el mismo orden que las reglas que definimos. Éstas las especificamos a continuación:

1. Recursos de procesamiento, se definen los parámetros de los recursos hardware de la ICUSW que influyen en el análisis de planificación.
2. Planificador, se especifican las políticas de planificaciones de la ICUSW.
3. Recursos compartidos, se definen los *mutex* definidos en la ICUSW.
4. Recursos de planificación, se configuran las tareas que son usadas en la ICUSW.
5. Operaciones, cada una de las acciones en forma de `msgHandlerItems`.
6. Transacciones, se definen las operaciones que son ejecutadas cuando un evento es recibido en la ICUSW.
7. Restricciones temporales, configuración de restricciones temporales sobre la definición de las transacciones de MAST.
8. Situaciones de tiempo real, la ICUSW tiene diferentes modos de funcionamiento, para cada uno de ellos se generará un modelo MAST.

Recursos de procesamiento

Como ya vimos en 4.3.1 la información necesaria para generar este modelo de MAST es el modelo `TSAMPlatform`. La ICUSW se ejecuta sobre una plataforma `Leon2`. Algunos de los principales valores configurados se muestran en el cuadro 7.4.

Cuadro 7.4: Valores del modelo `ICUSWMAST` relativo a los recursos de procesamiento.

Elemento del modelo MAST.	Parámetro.	Valor.
`RegularProcessor`.	Avg ISR Switch.	0.0273 ms.
	Best ISR Switch	0.027 ms.
	Max Interrupt Priority	16 prioridades.
	Min Interrupt	1 prioridad
	Worst ISR Switch	0.028 ms.
	Nombre	Leon2Proc
	Planificador	Proc1_Sched
	Factor de velocidad	1.0

Planificador

Como ya vimos en 4.3.2 en este modelo MAST especificamos los parámetros relativos al planificador. En nuestro caso la transformación sólo tiene en cuenta planificadores del tipo FPPS, el cual se decora con información de las latencias que introduce. Esta transformación es válida para la ICUSW ya que usa un planificador de este tipo. Como la ICUSW usa un sistema operativo ya calificado —éste es el sistema operativo calificado por la empresa `EdiSoft`— este modelo es anotado con los `WCET` proporcionados por la empresa. En el cuadro 7.5 mostramos los valores asociados al modelo de planificación de MAST.

Cuadro 7.5: Valores del modelo `ICUSWMAST` relativo al planificador.

Elemento del modelo MAST.	Parámetro.	Valor.
`PrimaryScheduler`.	Nombre.	MainScheduler.
	SchedulableResource	`EPDManager`, `SensorTMManager` y `BKGTCExecutor`.
	Host	`Leon2Proc`
`Scheduler Policy`	Tipo	FPPS
	Avg. Context Switch	0.0325 ms.
	Best Context Switch	0.037 ms.
	Worst Context Santucha	0.038 ms.

Recursos compartidos

Como ya vimos en 4.3.2 en este modelo MAST especificamos los parámetros relativos a los *mutex*. En el caso de la ICUSW se definieron un total de cinco componentes, por cada uno de ellos se definirá un *mutex*. A continuación se muestra una lista con la relación componente y *mutex*.

- EPDManager, EPDManagerMutex.
- SensorTMManager, SensorTMManagerMutex.
- BKGTCExecutor, BKGTCExecutor.
- SCTxChannelCtrl, SCTxChannelCtrl.

Recursos de planificación

Como ya vimos en 4.3.2 en este modelo MAST especificamos las tareas. Por cada componente proactivo y reactivo se crea una tarea de ejecución. La ICUSW tiene tres componentes de este tipo, por lo que en el modelo MAST se crean un total de tres tareas que tienen el mismo nombre que el componentes, éstas son:

- EPDManager.

- SensorTMManager.
- BKGTCExecutor.

Operaciones

Esta subsección y la siguiente son las más complejas de la transformación desde el modelo TSAM al modelo MAST. Vimos en la sección 4.3.2 cómo se realiza la transformación. Ésta consta de la transformación directa desde de los `TSAMMessageHandlerItem` a `Simple_Operations_Operations` y su agrupación en `Composite_Operations`.

Para ilustrar la generación de las operaciones de la ICUSWMAST nos centraremos en la generación de las `Operations` relativas a la gestión de telecomandos, en concreto las relativas a las `Operations` de la gestión del telecomando de ciencia. Éstas se muestran en la imagen 7.12, donde vemos como se componen los diferentes agrupaciones de `Operations`. Los `Simple_Operations` están compuesto por accesos a recursos compartidos *Mutex* —su adquisición y liberación— y cada `TSAMMessageHandlerItem` que estuviese definido en el `TSAMComponent`.

Transacciones

Como ya fue explicado en la sección 4.3.3 mostramos la definición de las transacciones que son ejecutadas cuando un evento externo es recibido por el sistema, donde vimos, que la transformación de las transacciones se realizar a partir de los modelos TSAMComponent y TSAMRTRequirement.

Al igual que en la sección anterior —las generación de las operaciones— usaremos la gestión de los telecomandos de ciencia para ilustrar esta transformación, ésta se puede ver en la imagen 7.12. Esta transacción está compuesto por un `External Event` denominado `SensorTMManagerTC`. Este evento externo ejecutará cada una de los `TSAMMessageHandler` encargado de gestionar los telecomandos de ciencia. Vemos como la transacción completa contiene un elemento `Fork` y `Branch` —hay dos `TSAMMessageHandler` encargados de la ejecución de los telecomandos de ciencia—, esta transacción sólo será válida para la herramienta de simulación *JSimMAST*. Con el objetivo de usar la herramienta de análisis de planificabilidad, dos situaciones de tiempo real fueron definidas, la situación «normal-exec» y «error». Se definieron dos eventos `SensorTMManagerTCerror` y `SensorTMManagerTC-exec-normal`, uno que gestiona los telecomandos que contienen errores y otro que no contiene errores.

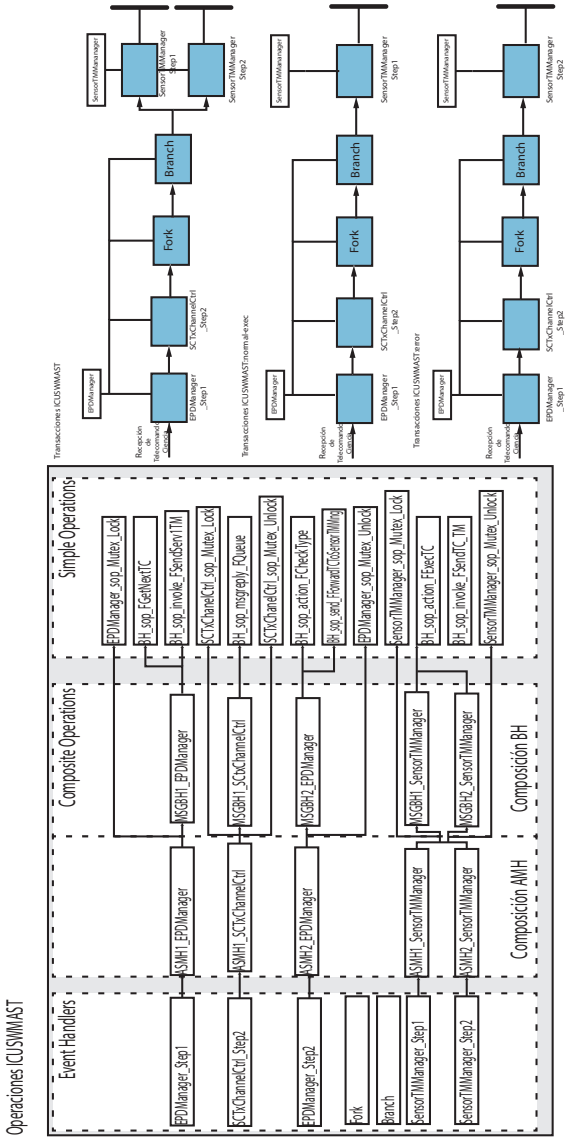


Figura 7.12: Definición de las operaciones y transacciones relativas a la gestión de los telecomandos de ciencia de la ICUSWMAST. En la parte superior mostramos las operaciones y su composición, con los tipos: BH y ASMH. En la parte inferior mostramos la transacción.

Restricciones temporales

Como ya se vio en la sección 4.3.4 se creará los `HardDeadline` en función de las especificaciones de restricciones temporales en el modelo TSAMRTRequirement. Como vimos en la descripción del modelo TSAMRTRequirement, la gestión del telecomando prioritario tiene una restricción temporal, ésta se configuró a 330 ms. —tiempo en el que deberá ser procesado un telecomando prioritario—.

7.4.2. Análisis de rendimiento (*Performance*)

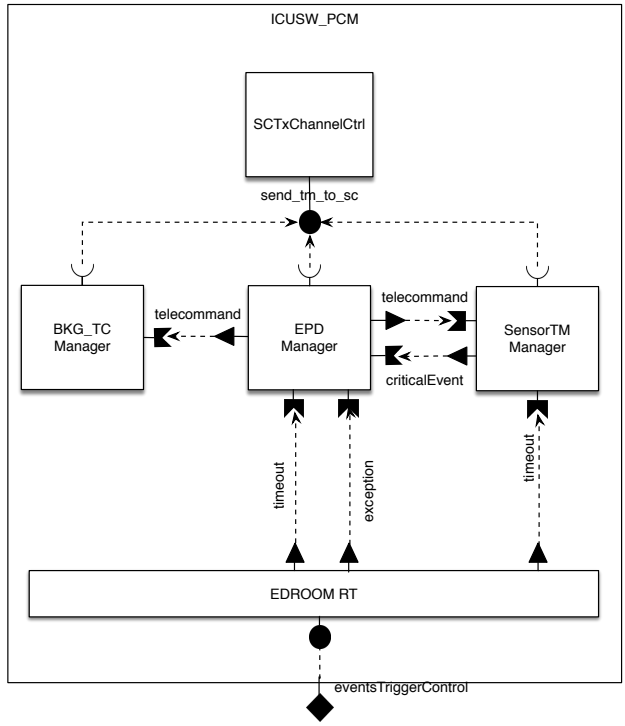


Figura 7.13: Descripción de la arquitectura del ICUSW en Palladio Component Model.

A continuación, mostramos la transformación entre el modelo de diseño del ICUSW y el modelo `ICUSWPCM`, usando las reglas que presentamos en el capítulo 5. La transformación del ICUSW se aplican en el mismo orden que las usadas en la definición de las reglas. La imagen 7.13 muestra un esquema de la arquitectura PCM obtenida. Ésta será usada para ilustrar cada una de las transformaciones.

(Regla 1) Definición de componentes

De acuerdo a la primera regla, cada uno de los componentes que fueron definidos en el ICUSW es transformado a un `BasiComponent` en el modelo `ICUSWPCM`. En la imagen 7.13 se muestra que ha sido generado un `basicComponent` equivalente para: `EPDManager`, `BKGTCExecutor` y `SensorTMManager`.

(Regla 2) Descripción de composición a nivel de sistema

La segunda regla de transformación es la relacionada con la comunicación de los componentes. Hay que tener en consideración los componentes involucrados y la dirección de los mensajes que son transmitidos entre ellos. La comunicación del ICUSW puede expresar comunicación bidireccional, por ejemplo, entre los componentes `EPDManager` <=>`SensorTM_Manager`. Los mensajes de salida de `EDPManager` son telecomandos que deberán ser ejecutados por `SensorTM_Manager`. Los mensajes de entrada al componente `EPDManager` son los eventos críticos que deberán ser manejados por éste.

La transformación de los protocolos bidireccionales producen la definición de dos interfaces del tipo `EventGroup`, una de entrada y otra de salida. Cada interfaz expresa mediante sus `signatures` los mensajes. Los componentes pueden usar estas interfaces, ya sea bien como entrada `SinkRoles`, o como salida `SourceRole`. Por ejemplo, `EPDManager` implementa `SourceRole` para el reenvió de telecomandos, mientras que para los eventos críticos implementa puertos del tipo `SinkRole`. `HK_FDIRManager` y `SensorTM_Manager` son puertos conjugados, lo que significa que es necesario intercambiar el tipo de roles, por ejemplo, en el caso de los telecomandos usa un puerto del tipo `SinkRole`, mientras que para la gestión de los eventos implementa un puerto del `SourceRole`.

(Regla 3) Descripción de recursos compartidos del sistema

La tercera es la transformación relativa a los recursos compartidos. En el ICUSW todos los componentes producen telemetría. Esta telemetría se manda al ordenador central de abordo de la misión siendo finalmente enviada a tierra.

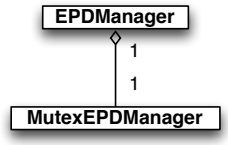


Figura 7.14: Ejemplo de la relación de agregación entre el componente y el `PassiveResource`, permite definir que los componente sólo podrán atender a una petición al mismo tiempo.

La ICU provee un módulo hardware que automáticamente controla el canal de transmisión hacia el ordenador central de abordo. Una cola *FIFO* de descriptores de transmisión es usada para gestionar el módulo hardware y en consecuencia el tráfico en el canal. La cola está protegida por una sección crítica, la cual se modelada mediante el componente `SCTxChannelCtrl`.

El único recurso compartido en el ICUSW es, por tanto, el `SCTxChannelCtrl`, mostrado en la imagen 7.13. Éste se transforma en un `BasicComponent`. Una interfaz operacional `send_tm_tc_sc` ha sido definida con el propósito de acceder a este componente. Los componentes pueden acceder a este recurso compartido definiendo una interfaz operacional del tipo `RequiredRole`, la cual se enlazará con la correspondiente interfaz operacional `ProvidedRole` del propio recurso compartido. La exclusión mutua dentro del `SCTxChannelCTrl` se materializa mediante la instanciación interna de un recurso pasivo que usa un protocolo de techo de prioridad inmediato. En este caso la prioridad del componente `EPDManager` se define siempre que se acceda al recurso compartido `SCTxChannelCTrl`. En este caso, la operación de exclusión mutua `MsgTMToSC` definida en el ICUSW (ver figura 7.14) se define adjuntando al RDSEFF correspondiente las operaciones de adquisición y liberación sobre el recursos compartido.

(Regla 4) Descripción de los RDSEFF

Todos los componentes, ya sean software de aplicación o de *system run-time*, requieren la interfaz `IPriority` del tipo infraestructura con el fin de especificar la prioridad de las demandas sobre la CPU. Un proveedor de recurso `PPS_CPU` provee esta interfaz `IPriority` para modelar la CPU del sistema y gestionar su planificación. Como se muestra en la figura 7.15 cada acción define la prioridad de la demanda de CPU y una estimación del uso de la CPU. Toda esta infor-

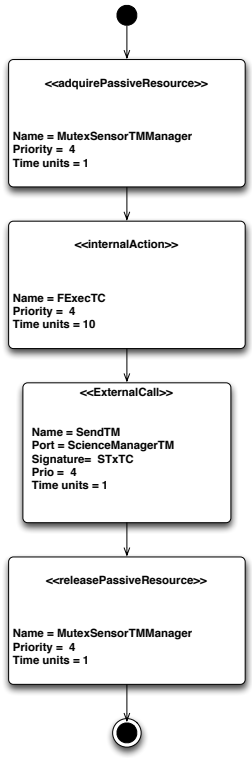


Figura 7.15: Descripcion del RDSEFF de la gestión del servicio de *SensorTM-Manager*.

mación es usada por el planificador asociado al recurso `PPS_CPU` con el objetivo de determinar el orden de acceso a la CPU.

Las prioridades de `IPriority` son categorizadas en dos grupos, por un lado, las que son asignables a interrupciones y elementos del *system run-time*, y por otro lado, a las acciones que corresponde a los componentes de aplicación. En el modelo del ICUSW esta categorización se muestra en la imagen 7.16. Las 16 primeras prioridades son asignadas al primer grupo (interrupciones, *EDROOM Run-Time* (EDROOM-RT)) y las restantes, hasta 256, son asignadas a las acciones de los componentes de `EPDManager`, `HK_FDIRManager` y `SensorTM_Manager`.

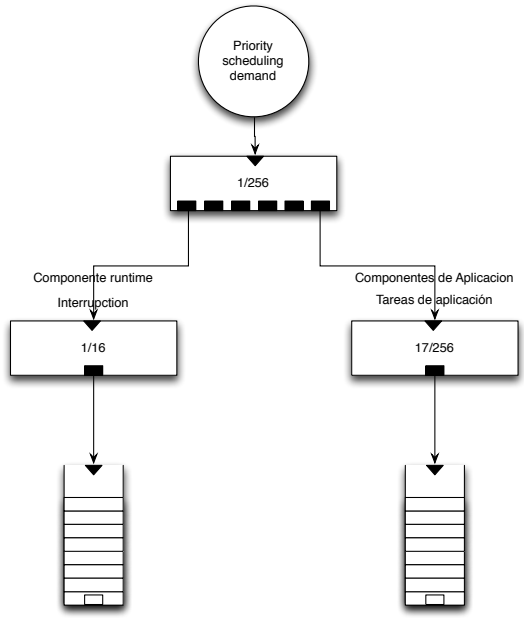


Figura 7.16: Asignación de las prioridades propuesto para el ICUSW.

Regla 5) *system run-time* y casos de uso

El `ICUSWPCM` debe modelar el comportamiento de los manejadores que el *system run-time* del sistema provee para cada una de las interrupciones y excepciones. Un único componente Palladio, llamado EDROOM-RT, agrupa todos estos manejadores y define una interfaz para el envío de notificaciones asociadas a excepciones. Entre el conjunto de suscripciones a interrupciones están las peticiones del servicio de temporización. Esta interfaz genera mensajes del tipo *timeout*. En la figura 7.13 se pueden ver el elemento en contexto con el ICUSW.

El componente EDROOM-RT provee, también, una interfaz centralizada a nivel de sistema, denominada `eventsTriggerControl`. Los puertos a nivel de sistema sólo pueden ser de este tipo por lo que la imagen exterior del sistema es un conjunto de puertos de excepción y temporización que son accesibles desde el modelo de uso (`UsageModel`).

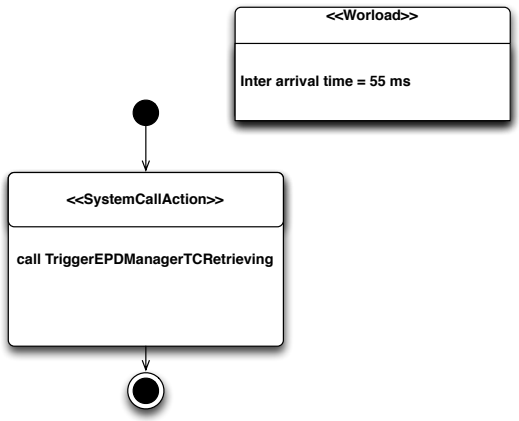


Figura 7.17: Descripción de un `ScenarioBehaviour` de un evento periódico, en este caso activa la interrupción `TriggerEPDManagerTCRetriving` de llegada de telecomandos.

Los patrones de activación del ICUSW, mostrados en el cuadro 7.1, son descritos en el modelo de uso `UsageModel`. Por cada uno de los eventos descritos en el cuadro se definen un `ScenarioBehaviour`. La figura 7.17 muestra un ejemplo del `ScenarioBehaviour` que corresponde al evento `TCRetrieving` asociado a la captura de los telecomandos enviados desde el ordenador central de abordo. El periodo del evento definido en el `Workload` es el especificado en el cuadro 7.1. La única acción definida es la llamada al `SystemProvidedRole` denominado `TriggerEPDManagerTCRetrieving` que forma parte de la interfaz `eventsTriggerControl`. Esta llamada es recogida por el EDROOM-RT que modela el manejador de interrupción que soporta el servicio de temporización, el cual enviará un mensaje de *timeout* notificando al componente que estaba suscrito al evento de temporización.

Elementos a caracterizar en el ICUSW

Los resultados presentados corresponden al proceso presentada en el capítulo 5 de la herramienta PCM. Recordemos que la ICUSW usa un planificador basados en FPPS por lo que la nueva extensión de SimuCom habilita la simulación de la ICUSW.

Los resultados que mostramos lo hacemos en función de transacciones del modelo ICUSW_PCM. La definición de transacción es idéntica que la que hemos usado en la herramienta MAST, esto es, que una transacción se define como la secuencia de acciones que se ejecutan en respuesta a un evento externo o programado desde el punto en que se activa hasta su final. Ésta requiere la cooperación de varias tareas que lo componen. Con el nuevo planificador basado en prioridades, la prioridad asignada a cada tarea componente determinará el tiempo de respuesta del evento.

El tiempo de simulación con SimuCom se ha establecido en 60 minutos. Este tiempo es suficientemente significativo si se considera un período de $1sg$ tanto en los eventos externos como los programados. Las operaciones que son analizadas son:

- La ejecución del telecomando prioritario, corresponde a la ejecución de un telecomando prioridad por el componente `EPDManager`. Recordemos que el número máximo de telecomandos prioritarios es de 10 y que su ejecución tiene que tardar menos de 300 ms.
- La ejecución del telecomando de ciencia, corresponde a la ejecución de un telecomando ciencia por el componente `EPDManager` y el componente `SensorTMManager`. Recordemos que el número máximo de telecomandos prioritarios es de 10 y que su ejecución tiene que tardar menos de 300 ms.
- La ejecución de la gestión de la telemetría por parte del servicio de ciencia, ésta se gestiona por el componente `SensorTMManager` 7.11. Recordamos que la telemetría se recolecta cada $2sg$, se procesa y se envía a tierra.

Después de la ejecución en SimuCom del modelo ICUSW_PCM fueron obtenidos los tiempos de respuesta de la ICUSW. Los resultados mostraron que se cumplen las restricciones de temporales para todos los casos durante todo el tiempo de simulación, es decir, los peores tiempos de respuesta son menores que sus restricciones temporales. Esto era lo esperado, ya que como vimos en el análisis de planificabilidad, que hemos visto en este mismo capítulo, el sistema era planificable. Hay que tener en cuenta, que cuando se trata de sistemas en tiempo real, este tipo de información tiene que ser analizado a fondo y los peores casos prevalecerá sobre los medios. A pesar de que el uso de herramientas de análisis basadas en cálculos teóricos se considera una alternativa mejor, como es el caso de la herramienta TSAM, SimuCom es capaz de producir los datos estadísticos que ésta última no puede, como son la dispersión de los tiempos de respuesta. Esta dispersión, junto con el tiempo de simulación, muestra lo fácil

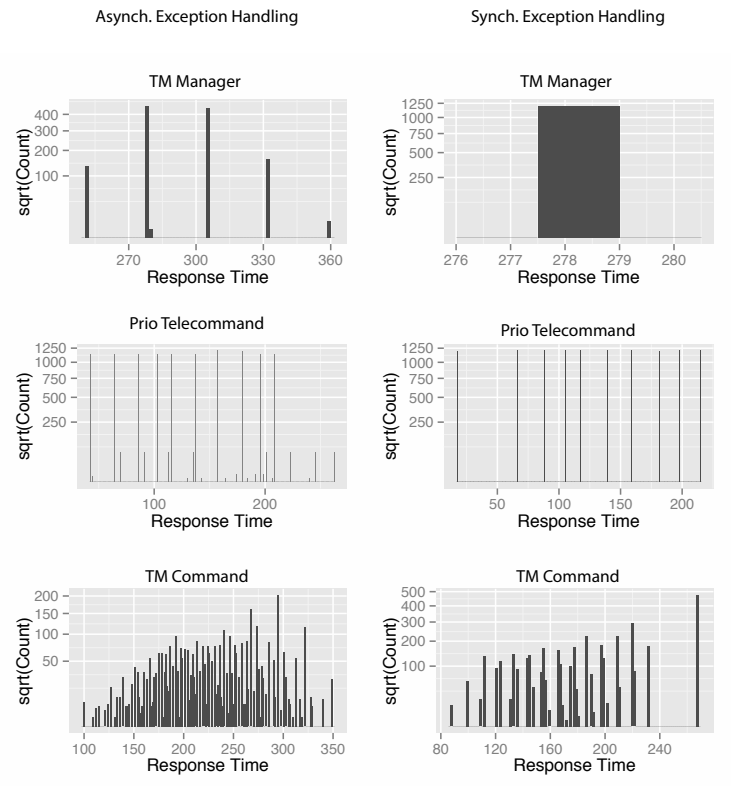


Figura 7.18: Datos obtenidos de una simulación de 60 minutos del modelo ICUSW_PCM.

que es caracterizar un sistema desde el punto de vista de sus bancos de pruebas, estrategia que como hemos visto ya en el capítulo 2, son ampliamente utilizados por la industria.

Los resultados de la simulación muestran que el el manejo de excepciones es una fuente importante de dispersión en el tiempo de respuesta de los eventos del sistema. La razón de esto es cómo esta característica fue implementada por la arquitectura de componentes de la ICUSW. El manejo de excepciones funciona de la siguiente manera: cada vez que un componente o el hardware detecta una condición de error, se lo notifica al componente de alta prioridad, es decir,

a `EPDManager`. Dependiendo del tipo de error y el estado actual del sistema, `EPDManager` puede o bien pasar al modo degradado o dejar que la aplicación continúe su ejecución después de aplicar una acción de recuperación —se puede ver en la descripción del comportamiento del componente `EPDManager`—. El hecho de que este componente sea el responsable de manejar las excepciones significa que puede adelantarse a la tarea actual y adquirir la CPU en cualquier momento, ya que hay varias fuentes de error que pueden producirse en diferentes frecuencias, por ejemplo, errores de software, errores de memoria, etc.

El resultado de este tipo de análisis es muy relevante durante la etapa de integración, cuando se lleva a cabo la campaña de pruebas, donde hemos propuesto, en el capitulo 6, una estrategia basada en la extracción de evidencias en el hardware real. Una dispersión alta en la simulación requerirá una gran cantidad de tiempo, la cual puede ser mayor al presupuesto asignado para esta etapa.

Para reducir la dispersión actual en los tiempos de respuesta del sistema hemos propuesto un cambio arquitectónico. Consiste en la introducción de un *buffer* para almacenar los eventos de error: cuando se produce un error, el componente correspondiente almacenará la información relacionada en este *buffer*. Éste será monitorizado periódicamente —una vez cada 400ms.— por el componente `EPDManager`, el cual recuperará el sistema los eventos de error generados por todos los demás componentes.

Los datos obtenidos de la simulación de la ejecución de los telecomandos prioritarios y la gestión de los paquetes de telemetría utilizando con ambas alternativas, se comparan en la figura 7.18. El cuadro 7.6 muestra los datos relevantes que resume esta comparación. La primera columna de cada alternativa muestra el tiempo máximo de respuesta en cada caso, el cual proporciona la información sobre el cumplimiento de las restricciones temporales. La segunda columnas y tercera son las relativas al DS de la frecuencia de ocurrencia de cada tiempo de respuesta y la cardinalidad del conjunto de diferentes frecuencias, que proporcionan una medida de la dispersión. A partir de estos datos hemos sacado las siguientes conclusiones:

- El tiempo de ejecución del peor caso se ha mejorado con la nueva propuesta, pero la versión anterior cumplía con los plazos. Ésto significa que esta optimización no es relevante desde un punto de vista de VV los requisitos de tiempo.
- La dispersión de la DS de la frecuencia de ocurrencia de los diferentes tiempos de respuesta y la cardinalidad del conjunto de diferentes frecuencias fue reducida.

Cuadro 7.6: La comparación entre los tiempos máximos de respuesta , la desviación estándar (DS) de la frecuencia de ocurrencia de cada tiempo de respuesta diferente y la cardinalidad de la frecuencia ajusta con ambas alternativas de manejo de excepciones.

	Asynch. Exception Handling			Synch. Exception Handling		
Evento	Tiempo max de resp.	DS of frec. del tiempo de Resp.	Cardinalidad de las frec	Tiempo max. resp.	DS de la frec de los tiempos de resp.	Cardinalidad del conjunto de fre.
Prio Telecommand	237 ms.	548.64	28	215 ms.	3.37	10
TM Manager	359 ms.	202.06	6	278 ms.	0	1

- Con la nueva propuesta, los resultados obtenidos permiten usar el proceso de verificación propuesto en el capítulo 6 de una manera más eficaz. Esta requiere que el software de aplicación ICUSW se ejecute en una versión de ingeniería o de vuelo de la ICU. La reducción dispersión obtenida permite alcanzar coberturas de decisión más alta con el mismo número de casos de prueba.

7.4.3. Modelo de verificación

En esta sección tratamos con un proceso de verificación de los elementos del modelo TSAM de la ICUSW. El objetivo es corroborar que los modelos de análisis tanto el de planificabilidad —el modelo MAST— como el de performance —modelo PCM— parten de un modelo TSAM coherente con el software desplegado en el modelo de ingeniería ICU. El proceso de verificacion usado para la ICUSW se describe en el capítulo 6.

La `ICU` tiene configurado un puerto de entrada/salida para la monitorización de los puntos de instrumentación. Su tamaño es de 16 bits, los cuales han sido distribuidos de la siguiente manera:

- **Event-seed**, identifica una instancia de un evento. hemos invertido en el 6 bits.
- **Event-identifier**, identifica un evento en particular. Hemos invertido en el un total de 2 bits, lo que nos permite identificar un total de 4 eventos.
- **Point-identifier**, identifica un punto en el código fuente. Con esta configuración identificamos hasta un total de 256 puntos de instrumentación.

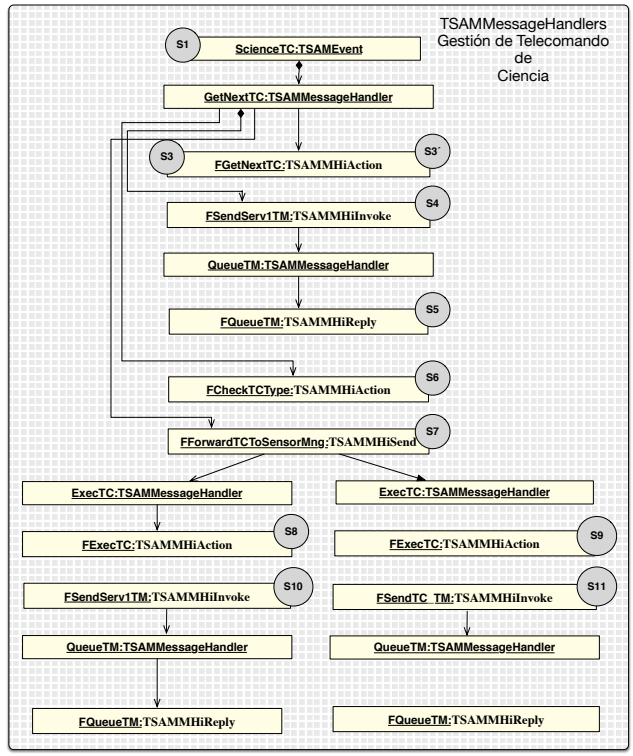


Figura 7.19: `TSAMMessagaHandler` junto con los `TSAMMessageHandlerItem` pertenecientes a la gestión de telecomandos de ciencia. Se muestran sobreimpresos los estados —puntos de instrumentación— necesarios para los perfiles de *End-to-End-flow*, `TSAMMessageHandlerItem` y patrones de activación.

Para la ICUSW usamos cuatro perfiles de los cinco descritos en el proceso. Los perfiles que dejamos fuera son los relacionados tanto con los *jitters* como los perfiles relativos a la plataforma. Entonces, los generados en la ICUSW son los siguientes:

- Perfil *End-to-End-flow*.
- Perfil `TSAMMessageHandlerItem`.
- Perfil de los patrones de activación.
- Perfil de situaciones de tiempo real.

Para ilustrar el proceso de verificación de la ICUSW usamos la gestión de los telecomandos de ciencia. Podemos observar los `TSAMComponent` junto con sus `TSAMMessageHandler` en la figura 7.19. Recordamos que el patrón de activación de los telecomandos de ciencia es del tipo ráfaga con un máximo de 10 telecomandos en $1sg$ y un tiempo mínimo entre telecomandos de un 1 ms. Además entre los distintos telecomandos de ciencia, está el telecomando 42 el cual condiciona las transacciones que pueden ejecutarse durante la nueva situación de tiempo real.

En el modelo TSAM esta circunstancia se definió con dos `TSAMMessageHandlers` asociados al mismo puerto y mensaje.

Perfil *End-to-End-flow*

Este perfil es el encargado de verificar que una transacción es válida —que los `TSAMMessageHandlerItem` ejecutados por la recepción de un evento externo son los correctos—. En la Figura 7.20 se muestra el AFT en relación con este perfil. Vemos como se comprueba que el orden de ejecución de los `TSAMMessageHandlerItem` es el mismo que el especificado en las transacciones de MAST o en los diferentes `RDSEFF` de PCM. Sólo hay una única bifurcación —ésta se puede ver entre los estados S7, S8 y S9— que es la encargada de modelar acciones cuando el telecomando es erróneo. Vimos como para los modelos de análisis de planificabilidad se definieron dos situaciones de tiempo real, una que define la correcta ejecución del telecomando —ésta está asociada al `TSAMMessageHandler` que ejecuta este tipo de telecomandos— y otra que no. En la imagen 7.21 mostramos los dos AFT que permiten verificar por separado cada una de las transacciones. Por último en la imagen 7.22 mostramos un punto de modulación relativo al ejemplo del telecomando 42, en este caso cuando se detecte su

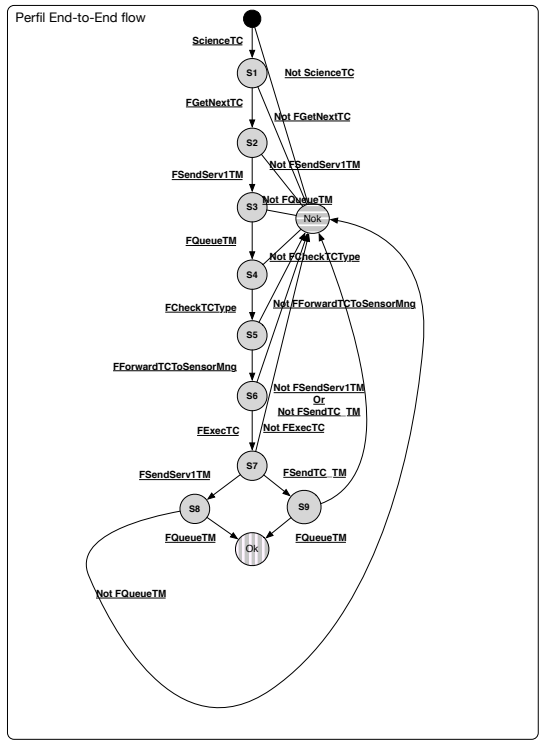


Figura 7.20: AFT de RapiCheck que verifica el perfil `end-to-end-flow` para la gestión de los telecomandos de ciencia.

ejecución sólo serán válidas un subconjunto de transacciones —éstas no se especifican pero serían las transacciones relativas a la ejecución de telecomandos de `backgroung` y de mantenimiento—.

Perfil `TSAMMessageHandlerItem`

Este perfil es el encargado de verificar que cada que vez que se ejecuta un `TSAMMessageHandlerItem` su tiempo máximo no exceda de su WCET asociado. Siguiendo el mismo ejemplo, tomamos el `TSAMMessageHandlerItem` `FGetNextTC` el cual es el encargado de capturar del *buffer* de telecomandos el siguiente telecomando que será ejecutado —en nuestro caso el siguiente telecomando de

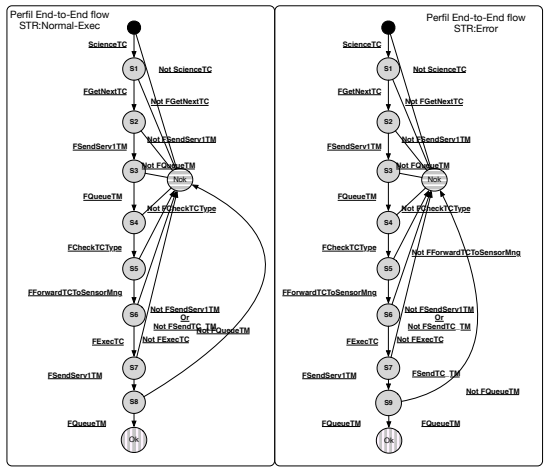


Figura 7.21: AFT de RapiCheck que verifica el perfil `end-to-end-flow` para la gestión de los telecomandos de ciencia discerniendo entre telecomandos que son ejecutados sin errores, y telecomandos que los contienen.

ciencia—. En la figura 7.23 mostramos el AFT resultante de la transformación. En ésta vemos como se instrumenta el final y el principio de la función, una vez que se detecta el punto de instrumentación de salida se calcula el tiempo transcurrido. Este tiempo corresponde con la ejecución de la función, el cual compararemos con el su WCET. Dependiendo de si supera el WCET el resultado de la verificación será positiva o negativa.

Perfil de los patrones de activación

El último perfil generado para la verificación de la ICUSW es el relativo a la comprobación de los eventos externos. Siguiendo el mismo ejemplo, tomamos el patrón de activación de los telecomandos de ciencia. Vimos al principio de esta sección, que el patrón de activación es del tipo ráfaga, donde el mínimo tiempo de llegada es de 1 sg. el número máximo de telecomandos de ciencia es de 8 telecomandos y que el tiempo mínimo de llegada entre telecomandos de la mismo ráfaga es de 1 ms. Con toda esta información se genera el AFT que vemos en la imagen 7.24. Cuando se detecta el evento externo, se comprueba, por un lado que el número de eventos de la ráfaga es menor que el número

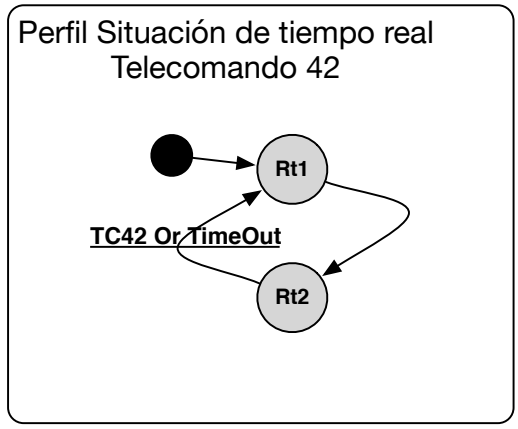


Figura 7.22: AFT RapiCheck que verifica que se active el punto de modulación cuando un telecomando de ciencia 42 es recibido, lo que condicionará a las futuras reacciones permitidas del propio componente.

máximo permitido — menos o igual a 8 telecomandos— y que el tiempo entre telecomandos es menor de 1 ms. Sólo se transitara al estado de aceptación `Ok` cuando se reciba la siguiente ráfaga.

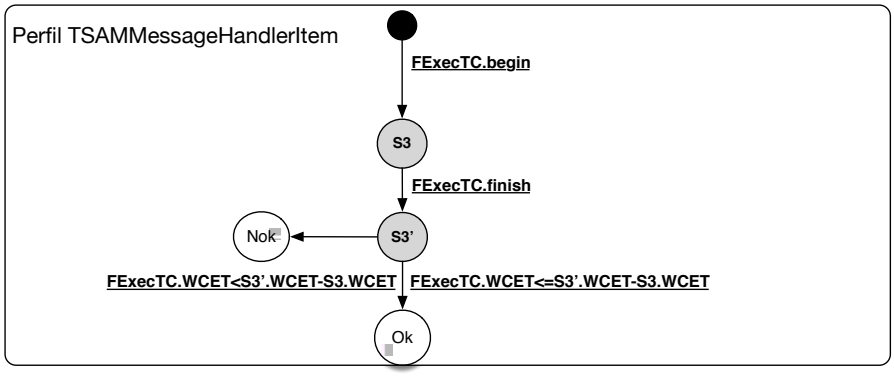


Figura 7.23: AFT `RapiCheck` que verifica que el WCET de la función `FGet-NextTC`.

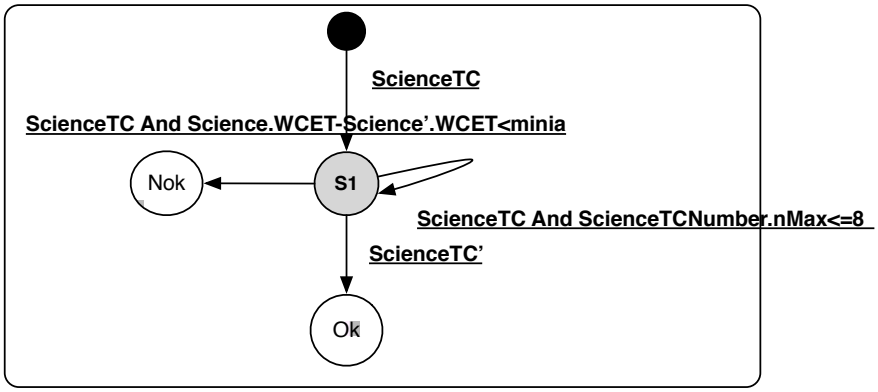


Figura 7.24: AFT RapiCheck que verifica el patrón de activación cuando un telecomando de ciencia es recibido.

Capítulo 8

Conclusiones y trabajo futuro

Ha sido presentado un proceso de VV cuyo objetivo es la reducción del coste en términos de recursos y tiempo. Para ello han sido presentados y vertebrados mediante MICOBS tres modelos:

- `Un modelo de análisis de planificabilidad`, valida que las restricciones temporales se cumplen.
- `Un modelo de análisis de rendimiento`, comprueba que la caracterización del sistema está cercana al tiempo disponible en los bancos de prueba de integración.
- `Un modelo de verificación`, comprueba que las propiedades presentes en el modelo de análisis transaccional corresponden con el sistema final que es desplegado.

La generación automática de modelos de análisis desde el modelo de diseño reduce los costes del proceso, estando éste asentado en un proceso en V, lo que ha permitido especificar el orden de las transformaciones en los distintos modelos.

El modelo transaccional de análisis tiene la propiedad de ser el punto de apoyo para la generación automática de los modelos de análisis. El modelo transaccional de análisis permite ser anotado con las propiedades extrafuncionales y es el punto de partida para la generación de los modelos anteriormente

mencionados. Este modelo permite realizar transformaciones dentro del dominio de análisis de propiedades extrafuncionales. El modelo es anotado con el WCET durante la campaña de pruebas unitarios. Todo el proceso ha sido vertebrado con técnicas MDE, permitiendo la cohesión de las etapas del proceso. Este modelo facilita la definición de nuevos modelos de análisis.

Un conjunto de estrategias han sido adoptadas para la caracterización del WCET. Estás estrategias están basadas en las propiedades de *composability* y *compositionality*, dando lugar a que los tiempos de ejecución puedan ser caracterizados y anotados en pruebas que estresen diferentes partes del sistema. De tal manera, que por una lado, podemos usar pruebas unitarias, las cuales tiene un carácter más exhaustivo, donde cada componente y librería de servicio pueden ser instrumentados y ejecutados caracterizando de esta manera los tiempos de ejecución. Por otro lado, tenemos las pruebas de integración, las cuales comprueban partes del sistema. Éstas son menos exhaustivas que las pruebas unitarias, pero tienen la ventaja a diferencia, de las anteriores de revelar las restricciones semánticas, lo que implica la detección de posibles caminos de ejecución no válidos.

El análisis de planificabilidad permite comprobar las restricciones de tiempos de ejecución. Para ello ha sido generado automáticamente desde el modelo transaccional. Los tiempos anotados en este modelo son el resultado de un análisis de WCET de cada elemento. Además, este modelo permite realizar el análisis de sensibilidad de cada elemento.

El modelo de análisis de rendimiento nos permite calcular la desviación de los diferentes tiempos de respuesta, cuantificando la calidad del análisis de verificación basado en evidencias. Para esto se utilizan técnicas de grado de confiabilidad o caracterizar elementos que están por encima del valor 3σ de la dispersión. Una de las transformaciones habilitadas es la de análisis de rendimiento, descrita usando Palladio Component Model (PCM). La herramienta Palladio Workbench no soportaba el análisis de sistemas de tiempo real. **Como parte de esta Tesis Doctoral se ha descrito una nueva extensión de PCM que permite este análisis**. Esta extensión da la posibilidad de **establecer la definición en PCM de políticas de planificación de sistemas de prioridades fijas con desalojo y protocolos de acceso a recursos compartidos**. Esta nueva extensión de PCM ha sido integrada en el proceso.

Para que los modelos de análisis de planificabilidad y de rendimiento fuesen correctos se ha propuesto el modelo de análisis de verificación. Este modelo de verificación **genera un conjunto de evidencias que tenían como objetivo comprobar que el comportamiento descrito en el modelo transaccional, junto con sus propiedades de *composabi-***

***lity* y *compositionality* en los tiempos de ejecución, se verifican en la integración del sistema**. Para esto han sido generado de manera automática (como antes) **un conjunto de perfiles de restricciones en la herramienta RapiCheck**.

Se ha mostrado un proceso de verificación automática de las restricciones temporales mediante la ejecución del sistema en el *target*. Las restricciones temporales son definidas desde los atributos descritos en el modelo de diseño, mientras que el código se obtiene a partir del modelo de diseño. El proceso genera evidencias adicionales para garantizar que las transformaciones sean correctas y que la implementación sea la esperada. Este trabajo se ha centrado particularmente en verificar las asunciones relativas al análisis de planificabilidad.

Se espera que con los datos resultantes de este proceso se pueda comprobar:

- Las restricciones de tiempo real que no se cumplen. El modelo de análisis de MAST las indicará, además, indicará de una manera cuantitativa el porcentaje que tiene que reducirse para que se satisfaga la restricción.

- También se comprueban mayores contribuciones de tiempo de respuesta, de tal manera que identifica los elementos del software que contribuyen principalmente a estos tiempos de respuesta globales del sistema, optimizando los elementos más significativos del software. Esta información es parte de la herramienta MAST.

- Se muestran qué elementos del software aumentan la dispersión de los tiempos de respuesta. Mediante la herramienta de análisis de rendimiento PCM podemos caracterizar cuál es la dispersión de los diferentes elementos que conforman el software. Esto nos permite adoptar estrategias diferentes en el modelo de diseño para reducir la dispersión de los tiempos de respuesta, es decir, aumentar la facilidad con la que caracterizar el sistema.

- Comprobamos que las asunciones en el modelo de análisis o transformación son incorrectas. El modelo de verificación RapiCheck nos mostrará cuáles de las asunciones en los modelos anteriormente citados se cumplen en el despliegue del sistema.

Conclusiones sobre el caso de uso

El proceso descrito se ha utilizado en el desarrollo del software de aplicación de una unidad de control del instrumento EPD, embarcado en el satélite Solar Orbiter. Con ello se comprueba que el proceso reduce los costes en las fases de verificación y validación de software embarcado en satélite, quedando falsada la hipótesis de partida. Para este fin se ha utilizado en la unidad de control el modelo de verificación propuesto. Se ha mostrado un proceso para la generación automática de las restricciones de tiempo. El modelo ha sido usado en la campaña de pruebas de integración. Esto ha permitido corroborar que **las propiedades de *composability* y *compositionality* en el WCET y las reacciones descritas en el modelo transaccional de análisis, corresponden con el software desplegado en la unidad de control, aumentando la calidad de los resultados del análisis de planificabilidad y de rendimiento**. Con ello, **los modelos de análisis de planificabilidad y análisis de rendimiento son más fiables al realizar este proceso de verificación**.

Se ha presentado un conjunto de escenarios para la validación de la nueva extensión de PCM-RT. Estas pruebas se encargan de validar el comportamiento del planificador de prioridades fijas con desalojo y los protocolos de acceso a los recursos compartidos.

En el caso de uso se han verificado los diferentes perfiles propuestos. En especial, nos hemos centrado en los perfiles de eventos singulares de ciencia y los perfiles nominales de funcionamiento. Los datos obtenidos no han sido muy reveladores, debido, principalmente, a la falta de madurez de la campaña de pruebas.

Una vez desarrollado el proceso objeto de investigación de esta Tesis Doctoral se podrían extraer datos que apuntan a los siguientes efectos:

- El análisis de planificabilidad satisfará las restricciones del tiempo de respuesta.

- Tras un análisis de sensibilidad, como es de esperar, en un proceso como el descrito, existen elementos del software susceptibles de mejora. Véase el gestor de telecomandos de la unidad de control, donde determinadas reacciones del software pueden ser objeto de optimización.

- Tras un análisis de rendimiento, también, se pueden encontrar reacciones del software mejorables, centrándose especialmente en la gestión de excepciones de la unidad de control.

- Se podría esperar que los resultados arrojados por el análisis de verificación mostrasen asunciones erróneas en la plataforma, en forma de bloqueos en el bus, tiempos de ejecución por encima de las restricciones de *jitters* en la gestión de la plataforma patológicos que invalidan el peor caso, etc.

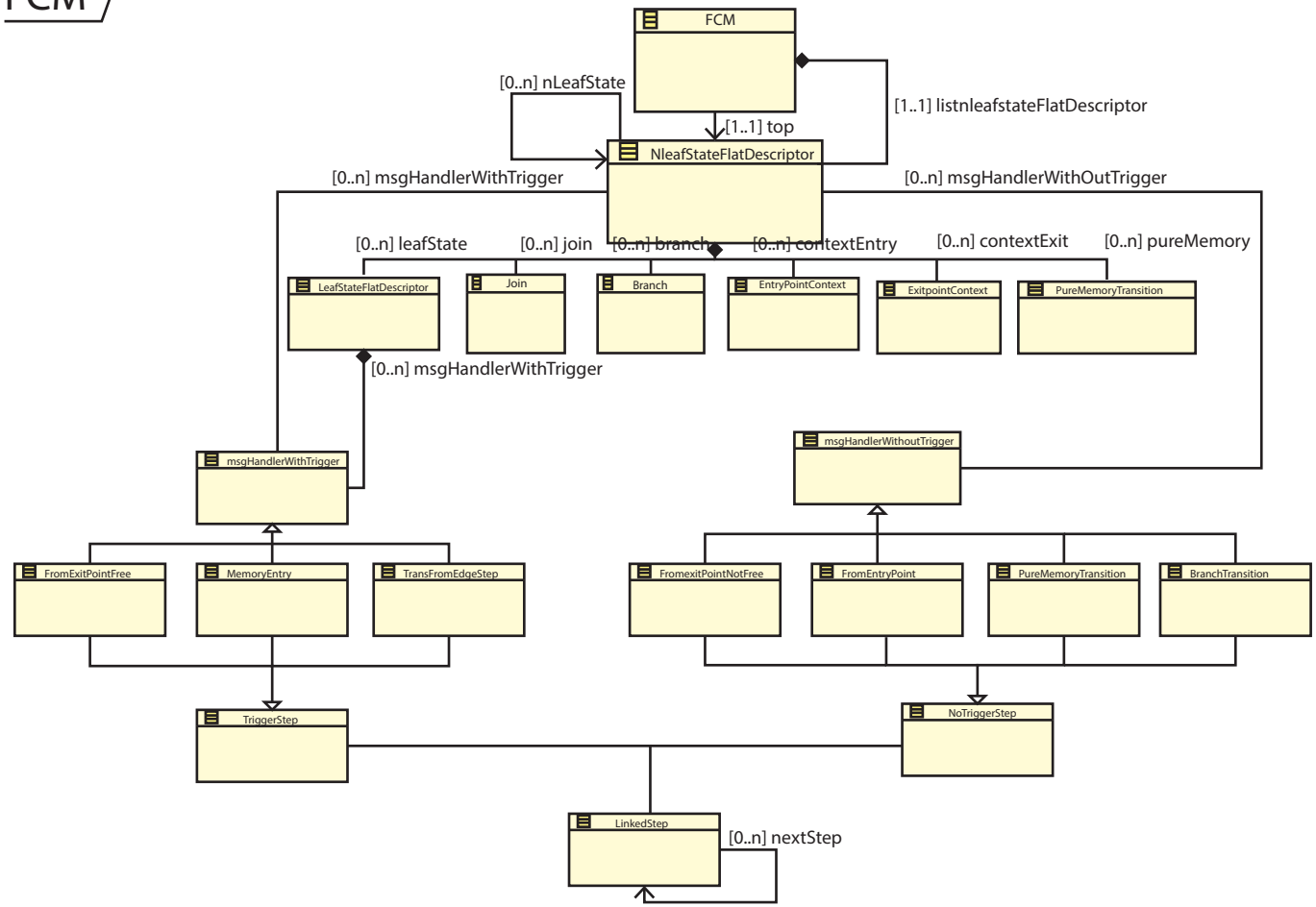
Resultados de la Tesis Doctoral

La propuesta ha sido desarrollada e integrada en MICOBS en forma de un conjunto de *plug-ins* de Eclipse. Los resultados parciales obtenidos han sido publicados en congresos (Fernández-Salgado et al., 2013a; Fernádez-Salgado et al., 2013). Éstas publicaciones presentan la definición de los modelos transaccionales, su anotación y transformación al modelo de análisis de planificabilidad de MAST. Además, se ha contribuido al diseño del software de aplicación de la unidad de control de EPD, que constituye el caso de uso de esta Tesis Doctoral, y participado en el artículo Sánchez et al. (2013), que lo describe. Otra publicación en la cual participé como coautor fue Polo et al. (2012b), ésta trata del uso de técnicas CBSE para mitigar los riesgos del desarrollo del software embarcado en nanosatélites. En la publicación Fernández-Salgado et al. (2015) ha sido mostrado en el análisis de verificación mediante evidencias. Otros resultados pendientes de publicar son el modelo de sistemas de tiempo real y la transformación correspondiente al modelo Palladio Component Model (PCM), y la descripción de la transformación del modelo de componentes EDROOM a los modelos de análisis transaccionales.

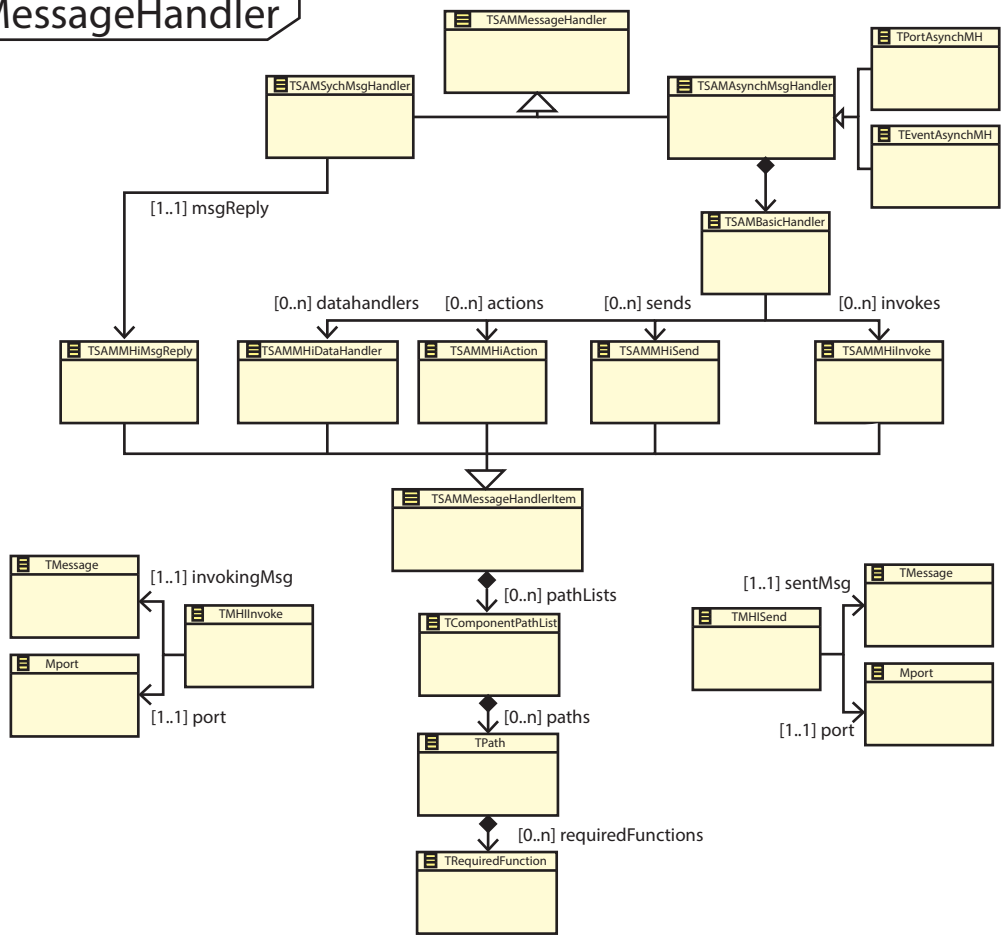
Enmarcados en el trabajo de Tesis Doctoral, se han realizado también informes técnicos, entregados a la ESA, como el diseño del software de aplicación de la unidad de control del instrumento EPD, las pruebas de integración iniciales con los sensores y los requisitos de parte de los sensores de los cuales está formado EPD.

Descripción de los modelos UML

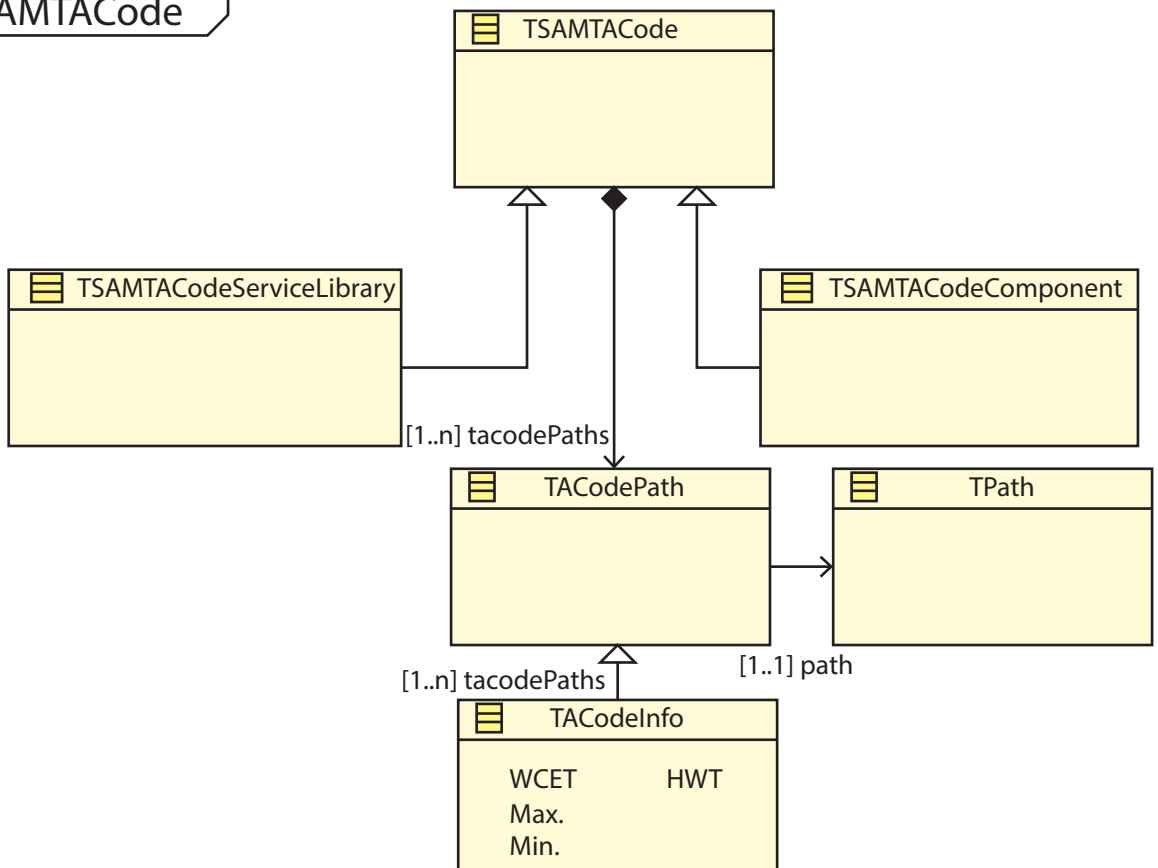
FCM


FCM
[0..n] nLeafState
[1..1] top
[1..1] listnleafstateFlatDescriptor
NleafStateFlatDescriptor
[0..n] msgHandlerWithTrigger
[0..n] msgHandlerWithOutTrigger
[0..n] leafState
[0..n] join
[0..n] branch
[0..n] contextEntry
[0..n] contextExit
[0..n] pureMemory
LeafStateFlatDescriptor
Join
Branch
EntryPointContext
ExitpointContext
PureMemoryTransition
[0..n] msgHandlerWithTrigger
msgHandlerWithTrigger
msgHandlerWithoutTrigger
FromExitPointFree
MemoryEntry
TransFromEdgeStep
FromexitPointNotFree
FromEntryPoint
PureMemoryTransition
BranchTransition
TriggerStep
NoTriggerStep
LinkedStep
[0..n] nextStep

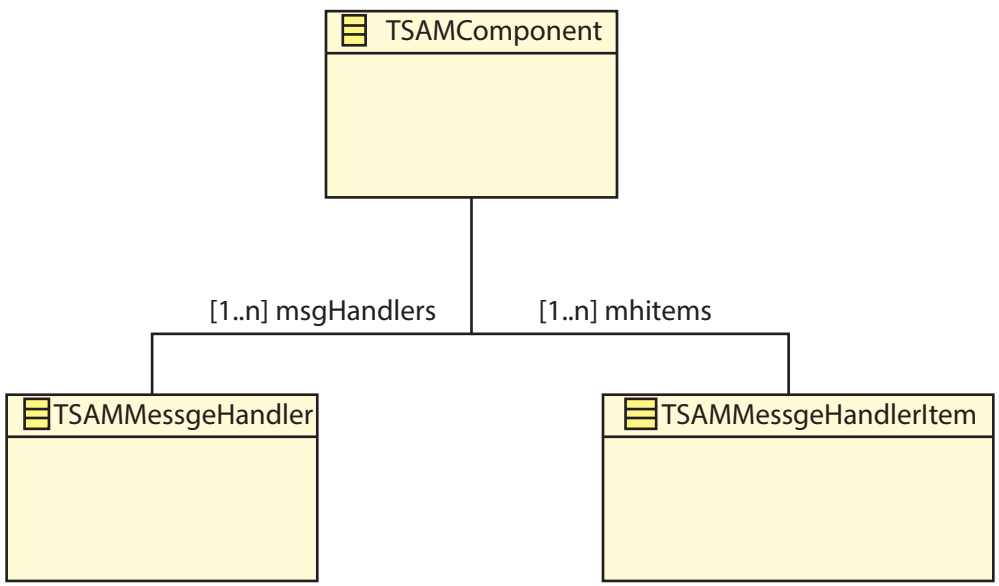
TSAMMessageHandler

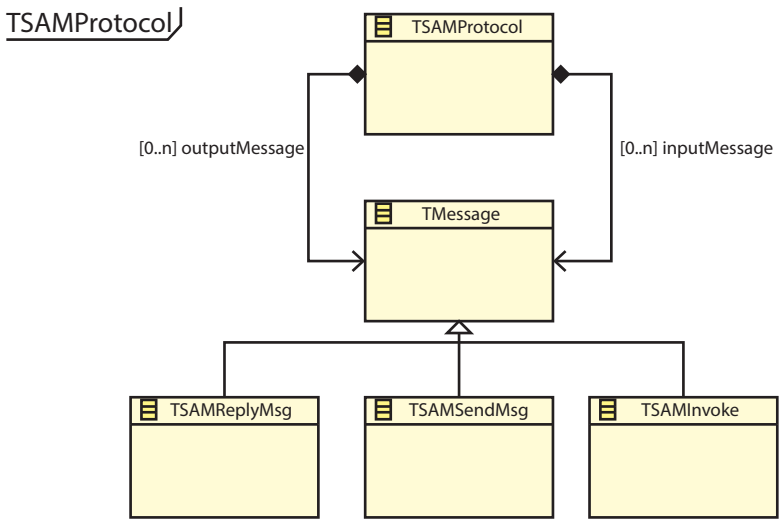


TSAMTACode

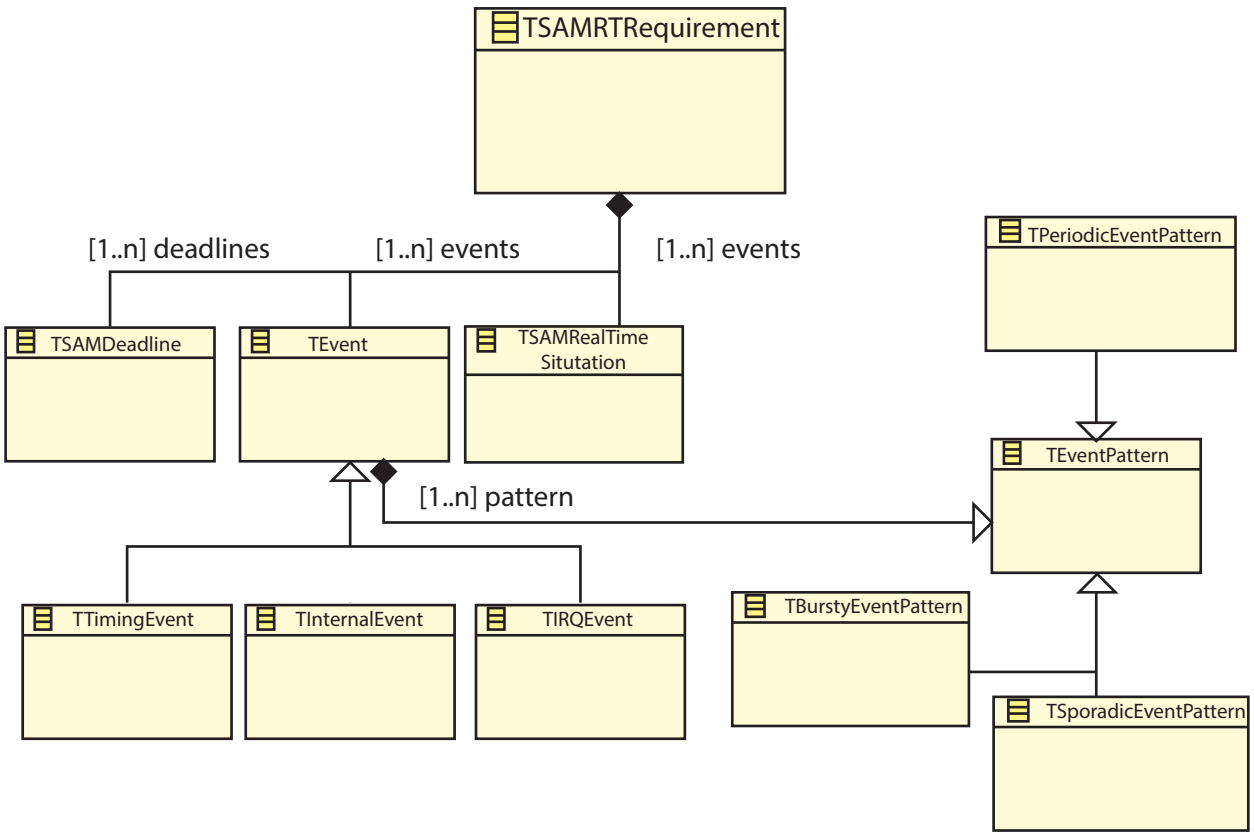

TSAMTACode
TSAMTACodeServiceLibrary
TSAMTACodeComponent
[1..n] tacodePaths
TACodePath
TPath
[1..n] tacodePaths
[1..1] path
TACodeInfo
WCET
HWT
Max.
Min.

TSAMComponent

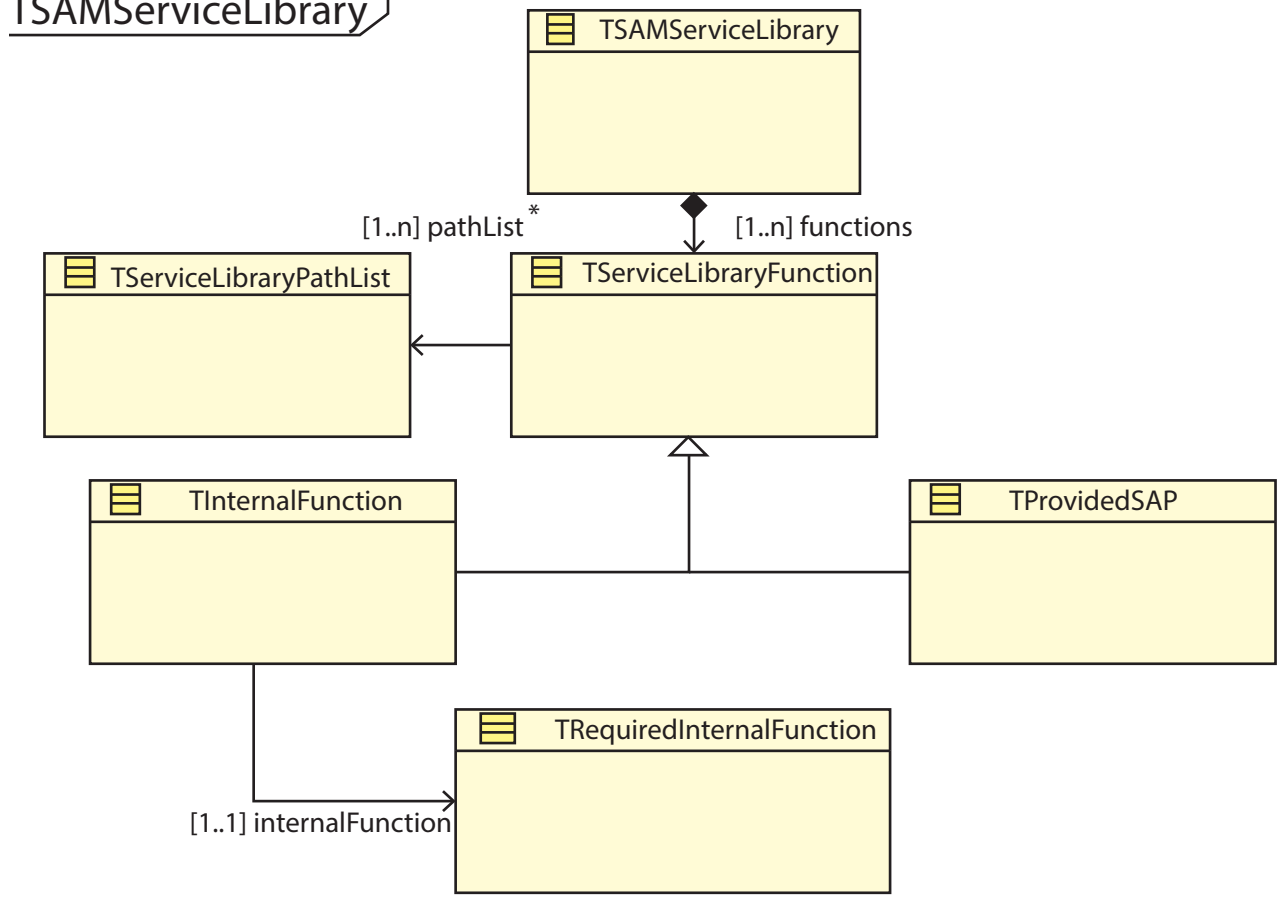



TSAMProtocol
TSAMProtocol
[0..n] outputMessage
[0..n] inputMessage
TMessage
TSAMReplyMsg
TSAMSendMsg
TSAMInvoke

TSAMRTRequirement



TSAMServiceLibrary



*Descripción idéntica a TMsgHandler

Lista de abreviaturas

AA *Application Architect*

AADL *Architeture Analysis and Design Language*

ACE *Advanced Composition Explorer*

ACS *Attitude Control System*

ACT *Abstract Component Technology*

AD *Application Developer*

ADA Lenguaje de Programación ADA

ADD *Architecture Design Document*

ADL *Architectural Description Language*

AFT Automata Finito Temporal

AlarmClock *MAST AlarmClock*

AMH-OC *Asynchronous Message Handler-Operation Composition*

AOCS *Attitude and Orbit Control System*

AOD *Application-Oriented Domain*

AODEM *Analysis-Oriented Deployment Model*

AODOM *Analysis-Oriented Domain Model*

AOEM *Analysis-Oriented Element Model*

AOM *Analysis-Oriented Model*

AOPM *Analysis-Oriented Platform Models*

API *Application Programming Interface*

APID *Application Process Identifier*

AST *Abstract Syntax Tree*

AT *Application Tester*

AVG *Average Execution Time Profile*

BasiComponent *PCM Basic Component*

BH-SOP *Basic Handler-Simple Operation*

BOTTOM-HALVES BOTTOM-HALVES

BranchPoint *Branch Point*

BranchTransition *Branch Transition*

BurstyEvent *MAST Bursty Event*

CAU Universidad Christian Albrecht

CBSE *Component-Based Software Engineering*

CCM *CORBA Component Model*

CCSDS *Consultative Committee for Space Data Systems*

CDL *Component Description Language*

CDPU *Control Process Data Unit*

CIDL *Component Implementation Definition Language*

CIF *CCM Implementation Framework*

CIM *Computation Independent Model*

COM *Component Object Model*

CompositeOperation *MAST Composite Operation*

COTS *Commercial-Off-The-Shelf*

CP *Component Provider*

CPU *Central Process Unit*

CRC *Cyclic Redundancy Check*

CRT *Component Run-Time*

CSP *Communicating Sequential Processes*

CT *Component Tester*

CTR Grupo de Computadores y Tiempo Real

DataHandler *Data Handler*

Deadlines *Dead Lines*

DMA *Direct Marketing Association*

D&C *Deployment and Configuration Specification*

DO-178-C *Software Considerations in Airborne Systems and Equipment Certification 178-C*

DS Desviación estándar

DSML *Domain-Specific Modelling Language*

EBNF *Extended Backus-Naur Form*

Eclipse *Eclipse Framework*

ECM *EDROOM Component Model*

Ecore *Meta Model Ecore*

ECSS-E-ST-40C *European Cooperation for Space Standardization, Space Engineering, Software*

EDF *Earliest Deadline First*

EDP *Embedded Development Platform*

EDROOM *ED Real-Time Object Oriented Modeling*

EDROOM-RT *EDROOM Run-Time*

EDROOMSL *EDROOM Service Library*

EEBB estáncias breves en el extranjero

EEPROM *Electrically Erasable Programmable Read-Only Memory*

EFP *Extra-Functional Properties*

EGSE *Electrical Ground Support Equipment*

EJB *Enterprise Java Beans*

EM *Engineering Model*

EMF *Eclipse Modelling Framework*

EMP *Eclipse Modelling Project*

EntryPoint *Entry Point*

EntryPointContext EntryPointContext*Entry Point Context*

EPD *Energetic Particle Detector*

EPDManager *Energy Particle Detector Manager*

EPT *Electron Proton Telescope*

ESA *European Space Agency*

ETP *Execution Time Profile*

EventGroup *Event Group*

EventHandler *Event Handler*

ExitPoint exitPoint*Exit Point*

ExitPointContext ExitPointContext*Exit Point Context*

ExternalEvent *MAST External Event*

FA *Framework Architect*

FCM *FLat Component Model*

FCS *Flat Component System*

FDIR *Fault Detection, Isolation and Recovery*

FIFO *First In First Out*

FireStateChange *Fire State Change*

FLATCMP *Flat Component Model*

FLATMCAD *Model Flat Model Architecture Desing*

FM *Flight Model*

FP *Functional Properties*

FPPS *Fixed Priority Preemptive Scheduling*

FPU *Floating-Point Unit*

FreeRTOS *Free Real-Time Operation System*

FromEntryPoint *From Entry Point*

FromExitPointFree *From Exit Point Free*

FromexitPointNotFree *From Exit Point Not Free*

FromLeafState *From Leaf State*

GEF *Graphical Editing Framework*

GMF *Graphical Modelling Framework*

GN&C *Guidance, Navigation and Control*

HardDeadline *MAST Hard Deadline*

HET *High Energy Telescope*

HKFDIR *House Keeping Fault and Detection Isolation and Recovery*

HVT *High Water Mark Analitical Time Profile*

HW Hardware

ICM *Intermediate Constructive Model*

ICU *Instrument Control Unit*

ICUSW *Intrument Control Unit Software*

ICUSWMAST *Intrument Control Unit Software MAST*

ICUSW_PCM *Instrument Control Unit Software Palladio Component Model*

IDE *Integrated Development Environment*

IDL *Interface Description Language*

IMA *Integrated Modular Avionics*

IOD *Implementation-Oriented Domain*

InmediateCeilingMutex *MAST Inmediate Ceiling Mutex*

InternalAction *Internal Action*

InternalEvent *Internal Event*

ISA *Instruction Set Architecture*

ISO-2626 *Road Vehicles – Functional Safety 2626*

ISR *Interrupt Service Routine*

ISRSwitch *MAST ISR Switch*

ISS *Instruction Set Simulator*

ITEA *Information Technology for European Advancement*

Join Join

JoinPoint *Join Point*

JoinTransition *Join Transition*

JSimMAST *Java Simulator MAST*

KIT Centro Tecnológico de Karlsruhe

Koala *Komponent Organiser and Linking Assistant*

KobrA *Komponenten Basierte Anwendungsentwicklung*

LeafState `Leaf State`

LeafStateFlatDescriptor *Leaf State Flat Descriptor*

LocalDeadline *Local Deadline*

LVPS *Low Voltage Slow Control*

LwCCM *Lightweight CCM*

M2M *Model-to-Model Transformation*

M2T *Model-to-Text Transformation*

MAST *MAST: Modeling and Analysis Suite for Real-Time Applications*

MARTE *Modeling and Analysis of Real-Time and Embedded Systems*

MASTPlatform *MAST Platform*

MCAD *Multi-Platform Component Architecture and Deployment Model*

MCLEV *MICOBS Composition Level*

MDA *Model-Driven Architecture*

MDE *Model-Driven Engineering*

MemoryEntry *Memory Entry*

MemoryTransition *Memory Transition*

MESP *Multi-platform Embedded Software Packaging Model*

MICOBS *Multi-platform Multi-Model Component-Based Software Development Framework*

MOF *Meta-Object Facility*

msgHandler `Message Handler`

msgHandlerItem `Message Handler Item`

msgHandlerWithOutTrigger *Message Handler WithOut Trigger*

msgHandlerWithTrigger *Message Handler With Trigger*

MutexInheritanceProtocol *MAST Mutex Inheritance Protocol*

MutualExclusionResource *MAST Mutual Exclusion Resource*

NASA *National Aeronautics and Space Administration*

NFP *Non-Functional Property*

NLeafState `No Leaf State`

NLeafStateFlatDescriptor *No Leaf State Flat Descriptor*

NoK *Not Valid State*

OBDH *On-Board Data Handling*

OBSW *On-Board Software*

OCL *Object Constraint Language*

Ok *Valid State*

OMG *Object Management Group*

Palladio *Palladio WorkBench*

PasivateThread *Pasivate Thread*

PassiveResource *Passive Resource*

PCM *Palladio Component Model*

PCM-RT *Palladio Component Model Real-Time*

PDL *Platform Description Language*

PDR *Preliminary Design Review*

PEC *Process Execution Control*

PECT *Prediction-Enabled Component Technology*

PeriodicEvent *MAST Periodic Event*

PIM *Platform Independent Model*

PLC *Mast Platform Configuration*

POSIX *Portable Operating System Interface*

PPS *Pulse per Second*

PrimaryScheduler *MAST Primary Scheduler*

ProcessingResource *Processing Resource*

PROM *Programmable Read-Only Memory*

ProvidedRole *Provided Role*

PSM *Platform Specific Model*

PureMemoryTransition *Pure Memory Transition*

PUS *Packet Utilization Standard*

QM *Qualification Model*

QoS *Quality of Service*

QVTO *Query, View, Transformation Language*

RapiCheck *RapiCheck Verification Tool*

RDSEFF *Resource Demanding Service Effect Specifications*

RegularProcessor *MAST Regular Processor*

RequiredRole *Required Role*

RIDL *Robocop Interface Definition Language*

RMS *Rate Monotonic Analysis*

ROOM *Real-Time Object-Oriented Modeling*

RTA *Response Time Analysis*

RTC *Run-To-Completion*

RTEMS *Real-Time Executive for Multiprocessor Systems*

RTOS *Real-Time Operating System*

RTRequirement *Real-Time Requirement*

RVS *Rapita Verification Suite*

SAE *Society of Automotive Engineers*

SAI *Service Access Interface*

SAM *System Analysis Model*

SAP *Service Access Point*

ScenarioBehaviour *Scenario Behaviour*

scheduleNextEvent *Schedule Next Event*

SDRAM *Synchronous DRAM*

SEFF *Service Effect Specification*

SEPT *Solar Electron and Proton Telescope*

SIES *Symposium on Industrial Embedded Systems*

SimpleOperation *MAST Simple Operation*

SimuCom *Simulation PCM*

SinkEvent *Sink Event*

SinkPort *Sink Port*

SoC *System on Chip*

SOFA *SOFtware Appliances*

SourcePort *Source Port*

SpaceWire *Space Wire*

SPL *Software Product Lines*

SporadicEvent *MAST Sporadic Event*

SRG *Space Research Group*

STEIN *SupraThermal Electrons, Ions, and Neutrals*

STEREO *Solar TErrestrial RElations Observatory*

STF *Sensor Transport Frame*

Sysml *Systems Modeling Language*

SystemProvidedRole *System Provided Role*

SystemRequiredRole *System Required Role*

TACode *Transactional Analisys Timing Annotated Code*

TBusrtyEventPatter *Transaccional Analysis Busrty Event Pattern*

TDSM *Transactional Design System Model*

TDeadLine *Transactional Analysis Deadline*

TEvent *Transactional Event*

ThreadScheduler *MAST Thread Scheduler*

Ticker *MAST Ticker*

Timer *MAST Timer*

TimingRequirement *Timming Requirements*

TInvokeMsg *Transactional Invoke Message*

TLM *Transaction Level Modelling*

TMAlarmClock *Tansaccional Analysis Alarm Clock*

TMessage *Transacctional Message*

TMMutexCeiling *Transactional Analysis MutexCeiling*

TMPrimaryScheduler *Transactional Analysis Primary Scheduler*

TM/TC *Telemetry and Telecommand*

TMTicker *Transaccional Analysis Ticker*

TPeriodicEventPattern *Transactional Analysis Periodic Event Pattern*

TransFromEdgeStep *Transition From Edge Step*

TReplyMsg *Transactional Reply Message*

TSAM *Transactional System Analysis Model*

TSAMAsynchMsgHandler *Transaccional Asynchronous Message Handler*

TSAMBasicHandler *Transactional System Analysis Basic Handler*

TSAMCMP *Transactional Analysis Component Model*

TSAMComponent *Transactional System Analisys Component*

TSAMEvent *Transaccional Analysis Event*

TSAMMessageHandler *Transactional Analysis Message Handler*

TSAMMessageHandlerItem *Transactional Message Handler Item*

TSAMMHiAction *Transaccional Analysis Message Handler Item Action*

TSAMMHiDataHandler *Transaccional Analysis Message Handler Item Data Handler*

TSAMMHiInvoke *Transaccional Analysis Message Handler Item Invoke*

TSAMMHiReply *Transaccional Analysis Message Handler Item Reply*

TSAMMHiSend *Transaccional Analysis Message Handler Item Send*

TSAMPlatform *Transaccional System Analisys Platform Model*

TSAMProtocol *Transactional System Analysis Model Protocol*

TSAMRealTimeSituation *Transactional Real-Time Situation*

TSAMRTRequirement *Real-Time Requirement Model*

TSAMSAI *Transactional System Analysis Service Access Interface*

TSAMSAP *TSAM Service Access Point*

TSAMSharedResource *Transaccional Analysis Shared Resource*

TSAMSL *Transactional System Analysis Model Service Library*

TSAMSynchMsgHandler *Transaccional Synchronous Message Handler*

TSDM *Transactional System Design Model*

TSendMsg *Transactional Send Message*

TSporadicEventPatter *Transactional Analysis Sporadic Event Pattern*

UAH Universidad de Alcalá

UART *A Universal Asynchronous Receiver/Transmitter*

UC/OS-II *Controller Operation System - II*

ULEIS *Ultra Low Energy Isotope Spectrometer*

UML *Unified Modeling Language*

UML-2 *Unified Modeling Language 2*

UMM *Unified Meta-component Model*

URI *Universal Resource Identifier*

Usagescenario *Usage Scenario*

VCF *Vienna Component Framework*

VHDL *VHSIC Hardware Description Language*

VV verificación y validación

VxWorks *VxWorks Real Time Operating System*

WCET *Worst-Case Execution Time Profile*

WRT *Worst-Case Response Time*

XML *Extensible Markup Language*

Índice Alfabético

Bibliografía

Accident, F. Controlled flight into terrain, american airlines flight 965, boeing 757-223, n651aa, near cali, colombia. *Aeronautica Civil of the Republic of Colombia.*, 1996.

Balp, H., Borde, É. y Haïk, G. Automatic composition of aadl models for the verification of critical component-based embedded systems. En *Engineering of Complex Computer Systems, 2008. ICECCS 2008. 13th IEEE International Conference on*, páginas 269–274. IEEE, 2008.

Balsamo, S., Di Marco, A., Inverardi, P. y Simeoni, M. Model-based performance prediction in software development: A survey. *IEEE Trans. Softw. Eng.*, vol. 30(5), páginas 295–310, 2004. ISSN 0098-5589.

Becker, S. *Coupled model transformations for QoS enabled component-based software design.*. Tesis Doctoral, Carl von Ossietzky University of Oldenburg, 2008. Http://d-nb.info/989923983.

Becker, S., Koziolek, H. y Reussner, R. The palladio component model for model-driven performance prediction. *Journal of System and Software (JSS)*, vol. 82(1), páginas 3–22, 2009. ISSN 0164-1212.

Betts, A. y Bernat, G. Tree-based wcet analysis on instrumentation point graphs. En *En Ninth IEEE International Symposium on Object-Oriented Real- Time Distributed Computing (ISORC 2006)*, páginas 558–565. Gyeongju, Korea, 2006.

Black, B. y Shen, J. Calibration of microprocessor performance models. *Computer*, vol. 31(5), páginas 59–65, 1998. ISSN 00189162.

Boehm, B. W. Verifying and validating software requirements and design specifications. *Software, IEEE*, vol. 1(1), páginas 75–88, 1984.

Börger, E., Cavarra, A. y Riccobene, E. *Algebraic Methodology and Software Technology: 8th International Conference, AMAST 2000 Iowa City, Iowa, USA, May 20–27, 2000 Proceedings*, capítulo An ASM Semantics for UML Activity Diagrams, páginas 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-45499-1.

Bruni, R., Lluch Lafuente, L. y Montanari, E., U. y Tuosto. Style-based reconfigurations of software architectures with qos constraints. 2007. Recuperado de http://citeseerx.ist.psu.edu /viewdoc/download ?doi=10.1.1.71.7006 &rep=rep1&type =pdf.

Brémaud, P. *Markov chains : Gibbs fields, Monte Carlo simulation and queues*. Texts in applied mathematics. Springer, New York, 1999. ISBN 0-387-98509-3.

Buttazzo, G. C. *Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series)*. Springer-Verlag TELOS, Santa Clara, CA, USA, 2004. ISBN 0387231374.

Cancila, R., D. an Passerone. Functional and structural properties in the model-driven engineering approach. En *En Proceedings of 13th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, Hamburg, Germany*, páginas 809–816. 2008.

Cantabria, I. U. D. Mast 2 meta-models. 2015.

Carloni, M., Ferrante, O., Ferrari, A., Massaroli, G. y Orazzo, L., A. y Velardi. Contract modeling and verification with formalspecs verifier tool-suite - application to ansaldo sts rapid transit metro system use case. En *Computer Safety, Reliability, and Security - SAFECOMP 2015 Workshops, ASSURE, DECSoS, ISSE, ReSA4CI, and SASSUR, Delft, The Netherlands, September 22, 2015, Proceedings*, páginas 178–189. 2015.

CCSDS. Packet telemetry, CCSDS 102.0-b-5, blue book. Informe técnico, CCSDS, 2000.

Chapman, R. Correctness by construction: a manifesto for high integrity software. *workshop on Safety critical systems and software*, páginas 5–8, 2006.

Chung, L. y Prado Leite, J. C. On non-functional requirements in software engineering. En *Conceptual Modeling: Foundations and Applications* (editado por A. Borgida, V. Chaudhri, P. Giorgini y E. Yu), vol. 5600 de *Lecture*

Notes in Computer Science, páginas 363–379. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-02462-7.

Cisco. Advanced qos services for the intelligent internet. White Paper, 2003. Getting Started with RTEMS.

Cobham Gaisler, A. Recuperado en http://www.gaisler.com. 2015.

Crnkovi, I. Component-based software engineering new challenges in software development. *Journal of Computing and Information Technology*, vol. 11(3), páginas 151–161, 2003.

Crnkovic, I., Larsson, M. y Preiss, O. Concerning predictability in dependable component-based systems: Classification of quality attributes. *Architecting Dependable Systems III*, páginas 257–278, 2005. LNCS 3549.

Crnkovic, I., Malavolta, I. y Muccini, H. A model-driven engineering framework for component models interoperability. En *En Proceedings of the 12th International Symposium on Component-Based Software Engineering*, CBSE '09, páginas 36–53. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-02413-9.

Crnkovic, I., Malavolta, I. y Muccini, H. A model-driven engineering framework for component models interoperability. *Based Software Engineering*, páginas 1–18, 2009a.

De Alfaro, L. y Henzinger, T. A. Interface automata. En *ACM SIGSOFT Software Engineering Notes*, vol. 26, páginas 109–120. ACM, 2001.

Dearle, T. A., A y Henzinger. Software deployment, past, present and future. *ACM SIGSOFT Software Engineering Notes*, 2001.

ECSS Secretariat. Telemetry and telecommand packet utilization. ECSS-E-ST-40C, 2009.

ESA. The assert project. http://www.esa.int/ TEC/Software_engineering_-and_standardisation /TECJQ9UXBQE_0.html, 2008.

Feldt, R., Torkar, R. y Ahmad, B., E. y Raza. Challenges with software verification and validation activities in the space industry. En *Proceedings of the 2010 Third International Conference on Software Testing, Verification and Validation*, ICST '10, páginas 225–234. IEEE Computer Society, Washington, DC, USA, 2010. ISBN 978-0-7695-3990-4.

Fernández-Salgado, J., Parra, P., R-Polo, O., Nilas-Nilas, B., García, I. y Sánchez-Prieto, S. Propuesta de cadena de herramientas para la automatización del análisis de planificabilidad del software del instrumento epd a bordo del solar orbiter. 2013.

Fernández-Salgado, J., Parra, P., Hauck, M., Hellín, A. M., Sánchez-Prieto, S., Krogmann, K. y Polo, O. R. Integration of a preemptive priority based scheduler in the palladio workbench. *J. Syst. Softw.*, vol. 114(C), páginas 20–37, 2016. ISSN 0164-1212.

Fernández-Salgado, J., Parra, P., Nilas, B., García, I., Sánchez, S. y Óscar, R. P. Schedulability analysis of on-board satellite software based on model-driven and compositionality techniques. En *Proceedings of the IEEE Internal Symposium on Industrial Embedded Systems (SIES)*, páginas 178–187. IEEE Computer Society, 2013a.

Fernández-Salgado, J., Parra, P., Sánchez Prieto, S., R. Polo, O. y Bernat, G. Automatic verification of timing constraints for safety critical space systems. En *DAta System In Aerospace (DASIA) 2105*. 2015.

Fernández-Salgado, J., Parra Espada, P., Rodríguez polo, O., Nilas, B., García Tejedor, I. y Sánchez Prieto, S. Propuesta de cadena de herramientas para la automatización del análisis de planificabilidad del software del instrumento epd a bordo del solar orbiter. En *CONGRESO ESPAÑOL DE INFORMÁTICA (CEDI)*, páginas 178–187. ÇONGRESO ESPAÑOL DE INFORMÁTICA (CEDI) / IV Jornadas de Computación Empotrada, 2013b.

Friedenthal, S., Moore, A. y Steiner, R. *A Practical Guide to SysML: Systems Modelling Language*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2008. ISBN 9780080558363, 9780123743794.

González Harbour, M., Gutiérrez García, J. J., Palencia Gutiérrez, J. C. y Drake Moyano, J. M. Mast: Modeling and analysis suite for real time applications. *En Real-Time Systems, 13th Euromicro Conference on 2001*, vol. 0, páginas 0125–134, 2001.

Goodenough, J. B. y Sha, L. The priority ceiling protocol: A method for minimizing the blocking of high priority ada tasks. *Ada Lett.*, vol. 7(7), páginas 20–31, 1988. ISSN 1094-3641.

Gössler, G. y Sifakis, J. Composition for component-based modeling. *Science of Computer Programming*, 2005.

Grassi, V., Mirandola, R. y Sabetta, A. Filling the gap between design and performance/reliability models of component-based systems: A model-driven approach. *Journal of System and Software (JSS)*, vol. 80(4), páginas 528–558, 2007.

Grønmo, R. y Møller-Pedersen, B. *Theory and Practice of Model Transformations: Third International Conference, ICMT 2010, Malaga, Spain, June 28-July 2, 2010. Proceedings*, capítulo From Sequence Diagrams to State Machines by Graph Transformation, páginas 93–107. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-13688-7.

Grotker, T., Liao, S., Martin, G. y Swan, S. *System Design With SystemC*. Springer, Dordrecht, 2002.

Gustafsson, J., Lisper, B., Sandberg, C. y Bermudo, N. A tool for automatic flow analysis of C-programs for WCET calculation. *Proceedings of the Eighth International Workshop on Object-Oriented Real-Time Dependable Systems, 2003. (WORDS 2003).*, páginas 106–112, 2003.

Happe, J. *Predicting software performance in symmetric multi-core and multiprocessor environments.*. Tesis Doctoral, Carl von Ossietzky University of Oldenburg, 2009. Http://d-nb.info/99670132X.

Harel, D. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, vol. 8(3), páginas 231–274, 1987. ISSN 0167-6423.

Hawkins, R. D. Using safety contracts in the development of safety critical object-oriented systems. *University of York*, 2006. (Tesis Doctoral). Recuperado de https://www-users.cs.york.ac.uk/rhawkins/papers/thesis2sides.pdf.

Heineman, G. T. y Councill, W. T. *Component-based software engineering: putting the pieces together*. Addison-Wesley Longman Publishing, Boston, MA, USA, 2001. ISBN 0-201-70485-4.

Hissam, S., Wallnau, K., Wood, W., Hudak, J., Ivers, J., Klein, M., Larsson, M., Moreno, G., Northrop, L., Plakosh, D. y Stafford, J. Predictable assembly of substation automation systems: An experiment report. Informe Técnico CMU/SEI-2002-TR-031, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 2003.

Holzmann, G. J. The model checker spin. *IEEE Trans. Softw. Eng.*, vol. 23(5), páginas 279–295, 1997. ISSN 0098-5589.

Hugues, J., Perrotin, M. y Tsiodras, T. Using mde for the rapid prototyping of space critical systems. En *En Proceedings of the 2008 The 19th IEEE/IFIP International Symposium on Rapid System Prototyping*, páginas 10–16. IEEE Computer Society, Washington, DC, USA, 2008. ISBN 978-0-7695-3180-9.

Jin, Y. y Li, S. Model-level wcet analysis of real-time system based on fpn. *Journal of Communication and Computer*, vol. 5(1), páginas 67–73, 2008.

Kent, S. Model driven engineering. En *Integrated Formal Methods* (editado por y. K. S. E. En M. Butler, L. Petre), vol. 2335 de *Lecture Notes in Computer Science*, páginas 286–298. Berlin, Heidelberg: Springer, 2002a. ISBN 978-3-540-43703-1.

Kent, S. Model driven engineering. En *En Proceedings of the Third International Conference on Integrated Formal Methods*, IFM '02, páginas 286–298. London, etc.: Springer-Verlag, 2002b. ISBN 3-540-43703-7.

Koziolek, H., y Reussner, R. Peropteryx: Automated application of tactics in multi-objective software architecture optimization. En *En Proceedings of the Joint ACM SIGSOFT Conference – QoSA and ACM SIGSOFT Symposium – ISARCS on Quality of Software Architectures – QoSA and Architecting Critical Systems – ISARCS*, QoSA-ISARCS '11, páginas 33–42. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0724-6.

Koziolek, H., Becker, S. y Happe, J. Predicting the performance of component-based software architectures with different usage profiles. *Proceedings of the 3rd International Conference on the Quality of Software Architecture, QoSA2007*, 2007.

Laplante, P. A. *Real-Time Systems Design and Analysis: An Engineer's Handbook*. IEEE Press, Piscataway, NJ, USA, 1992. ISBN 0780304020.

Lau, K. K. y Wang, Z. Software component models. *IEEE Transactions on Software Engineering*, vol. 33(10), páginas 709–724, 2007. ISSN 0098-5589.

Leveson, N. *SafeWare: System Safety and Computers*. Computer Science and Electrical Engineering Series. Reading, Massachusetts: Addison Wesley, 1995. ISBN 9780201119725.

Li, H., Puaut, I. y Rohou, E. Traceability of flow information: Reconciling compiler optimizations and wcetp estimation. En *En Proceedings of the 22Nd*

International Conference on Real-Time Networks and Systems, RTNS '14, páginas 97:97–97:106. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2727-5.

Lions, J. Ariane 5 flight 501 failure: Report by the inquiry board. *European Space Agency*, 1996. Recuperado de http://ravel.esrin.esa.it/docs//esa-x-1819eng.pdf.

Liu, J. W. S. W. *Real-Time Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edición, 2000. ISBN 0130996513.

López, P., Drake, J. M., Pacheco, P. y Medina, J. L. Ada-ccm: Component-based technology for distributed real-time systems. En *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, CBSE '08, páginas 334–350. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-87890-2.

Ltd., R. S. Worst-case execution time analysis. user guide. Informe técnico, Rapita Systems. Ltd., 2007.

Ltd., R. S. Rapicheck user guide version alpha. Informe técnico, Rapita Systems. Ltd., 2014., 2014.

Ltd., R. S. Atlas house, osbaldwick link road, york yo10 3jb. Recuperado de https://www.rapitasystems.com, 2015.

Lundqvist, T. A WCET Analysis Method for Pipelined Microprocessors with Cache Memories. *Computer Engineering*, 2002.

Mangeruca, L., Ferrante, O. y Ferrari, A. Formalization and completeness of evolving requirements using contracts. En *8th IEEE International Symposium on Industrial Embedded Systems, SIES 2013, Porto, Portugal, June 19-21, 2013*, páginas 120–129. 2013.

Marick, B. How to misuse code coverage. *En Proceedings of the 16th Interational Conference*, páginas 1–13, 1999.

Matic, S. *Compositionality in deterministic real-time embedded systems*. Electrical Engineering and Computer Sciences, University of California. Berkeley, 2008. (Technical Report No UCB/EECS-2008-12).

Mazzini, S., Puri, S. y Vardanega, T. Composability for high integrity real-time embedded systems. En *En Procceedings 1st Workshop on Compositional Theory and Technology for Real-Time Embedded Systems*. 2008.

McDermid, J. y Kelly, T. Software in safety critical systems? achievement and prediction. *Nuclear Energy*, vol. 2(3), páginas 140–146, 2006. ISSN 0140-4067.

McDermid, J. y Pumfrey, D. Software Safety: Why is there no Consensus? *York university*, 2001.

Miles, R. y Hamilton, K. *Learning UML 2.0: a pragmatic introduction to UML*. Sebastopol, California: O'Reilly Media, 2006. ISBN 0596009828.

Montecchi, L. y Lollini, A., P. y Bondavalli. Towards a mde transformation workflow for dependability analysis. *En 16th IEEE International Conference on Engineering of Complex Computer Systems*, páginas 157–166, 2011.

Niemann, B. y Haubelt, C. Assertion-based verification of transaction level models. 2006. Recuperado de http://www.imd.uni-rostock.de/fileadmin/IEF_IMD/veroeff/2006_MBMV06_NH.pdf.

Nilas, B. *Generación automática de instancias de modelos de análisis destinados a la verificación de las restricciones temporales de sistemas software empotrados.*. UAH, 2014.

Object Management Group. *CORBA Component Model (CCM), v.3.1*. OMG document formal/08-01-04, 2008.

Object Management Group. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*. OMG document formal/09-11-02, 2009.

OMG. Needham, ma, usa: Omg. Recuperado de http://www.omg.org/, 2015.

Parra, P., Polo, O., Knoblauch, M., García, I. y Sánchez, S. MICOBS: multi-platform multi-model component based software development framework. En *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, CBSE '11, páginas 1–10. ACM, New York, NY, USA, 2011.

Perrotin, M., Conquet, E., Delange, J. y Schiele, T., A. y Tsiodras. Taste: A real-time software engineering tool-chain overview, status, and future. En *SDL 2011: Integrating System and Software Modeling*. Berlin, Heidelberg: Springer, 2012.

Plazar, S., Lokuciejewski, P. y Marwedel, P. A retargetable framework for multi-objective wcet-aware high-level compiler optimizations . 2013.

Polo, O., Girón Sierra, J. M. y Esteban San Román, S. Edroom. automatic c++ code generator for real time systems modelled with room. *New Technologies for Computer Control*, 2002.

Polo, Ó., Parra, P., Knoblauch, M., García, I., Fernández-Salgado, J., Sánchez, S. y Angulo, M. Component based engineering and multiplatform deployment for nanosatellite on-board software. En *Proceedings of the DASIA 2012 Conference*. 2012a.

Polo, O., Parra, P., Knoblauch, M., García, I. y Sánchez-Prieto, M., Sebastian y Prieto. Component-based Engineering and Multi-Platform Deployment for Nanosatellite On-Board Software. En *Proceedings of the DASIA 2012 Conference*. 2012b.

Polo, O., Segundo, E. y De la Cruz, J. M. Control code generator used for control experiments in ship scale model. En *Proceedings of the CAMS2001 IFAC Conference*. 2001.

Racu, R., Hamann, A. y Ernst, R. A formal approach to multi-dimensional sensitivity analysis of embedded real-time systems. *18th Euromicro Conference on Real-Time Systems (ECRTS'06)*, páginas 3–12, 2006.

RapiTime. *Worst-case execution time analysis. User Guide*. York, United Kingdom: Rapita Systems, 2007. Recuperado de https://www.rapitasystems.com/products/rapitime.

Rathfelder, C. *Modelling Event-Based Interactions in Component-Based Architectures for Quantitative System Evaluation*, vol. 10 de *The Karlsruhe Series on Software Design and Quality*. KIT Scientific Publishing, Karlsruhe, Germany, 2013.

Rathfelder, C., Klatt, B., Sachs, K. y Kounev, S. Modeling event-based communication in component-based software architectures for performance predictions. *Software & Systems Modeling*, 2013. ISSN 1619-1366.

Reussner, R., Becker, S., Happe, S., Koziolek, H., Krogmann, K. y Kuperberg, M. The Palladio Component Model. Informe técnico, Universität Karlsruhe (TH), 2007.

Sánchez, S., Prieto, M., Polo, O., Parra, P., Castillo, R. y Fernández-Salgado, J. Instrument control unit for the energetic particle detector on-board solar orbiter. En *En Proceedings of the 39th COSPAR Scientific Assembly*, vol. 1667. 2012.

Sánchez, S., Prieto, M., Polo, O., Parra, P., Da Silva, A., Gutiérrez, O., Castillo, R., Fernández-Salgado, J. y Rodríguez-Pacheco, J. Hw/sw co-design of the instrument control unit for the energetic particle detector on-board solar orbiter. *Advances in Space Research*, vol. 52(6), páginas 989–1007, 2013. ISSN 0273-1177.

Sandberg, C., Ermedahl, A., Gustafsson, J. y Lisper, B. Faster WCET flow analysis by program slicing. *Proceedings of the 2006 ACM SIGPLAN/-SIGBED conference on Language, compilers and tool support for embedded systems - LCTES '06*, página 103, 2006.

Sauser, B. J., Reilly, R. R. y Shenhar, A. J. Why projects fail? How contingency theory can provide new insights - A comparative analysis of NASA's Mars Climate Orbiter loss. *International Journal of Project Management*, vol. 27(7), páginas 665–679, 2009. ISSN 0263-7863.

SCADE. Scade critical systems and software development solutions. Recuperado de http://www.esterel-technologies.com/products/scade-system, 2015.

Selic, B., Gullekson, G. y Ward, P. T. *Real-time object-oriented modeling*. New York: John Wiley and Sons, New York NY USA, 1994. ISBN 0-471-59917-4.

Sha, L., Abdelzaher, T., Arzén, K. E., Cervin, A., Baker, T., Burns, A. y Mok, A. K. Real time scheduling theory: A historical perspective. *Real-Time Systems*, vol. 28(2/3), páginas 101–155, 2004. ISSN 0922-6443.

Sha, L., Rajkumar, R. y Lehoczky, J. P. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Trans. Comput.*, vol. 39(9), páginas 1175–1185, 1990. ISSN 0018-9340.

Shaw, A. C. Reasoning about time in higher-level language software. *IEEE Transactions on Software Engineering*, vol. 15(7), páginas 875–889, 1989. ISSN 00985589.

Stafford, J. y Wallnau, K. A technology for predictable assembly from certifiable components. *Universitaet Karlsruhe, June/September*, vol. 3(April), 2003.

Storey, N. R. *Safety Critical Computer Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1996. ISBN 0201427877.

Study, . E. M. Embedded market study. En *Then Now: What's Next*. Recuperado de http://bd.eduweb.hhs.nl/es/2014-embedded-market-study-then-now-whats-next.pdf, 2014.

Thomasian, A. Concurrency control: Methods, performance, and analysis. *ACM Comput. Surv.*, vol. 30(1), páginas 70–119, 1998. ISSN 0360-0300.

VectorCast. Gmv españa. Recuperado de https://www.vectorcast.com/software-testing-products, 2015.

Viana, A., Polo, O., López, O., Knoblauch, M., Sánchez, S. y Meziat, D. EDROOM: A free tool for the uml2 component based design and automatic code generation of tiny embedded real time systems. En *En Proceedings of the 3rd European Congress on Embedded Real-Time Software ERTS*. 2006.

White, R. T., Healy, C. A., Whalley, D. B., Mueller, F. y Harmon, M. G. Timing analysis for data caches and set-associative caches. *Proceedings Third IEEE Real-Time Technology and Applications Symposium*, páginas 192–202, 1997.

Wilhelm, R., Engblom, J., Ermedahl, A., Holsti, N., Thesing, S., Whalley, D., Bernat, G., Ferdinand, C., Heckmann, R., Mitra, T., Mueller, F., Puaut, I., Puschner, P. y Staschulat, P., J. y Stenström. The worst-case execution-time problem – overview of methods and survey of tools. *ACM Trans. Embed. Comput. Syst.*, vol. 7(3), páginas 36:1—-36:53, 2008. ISSN 1539-9087.

Wilhelm, R., Engblom, J., Thesing, S. y Whalley, D. Industrial Requirements for WCETTools. En *Answers to the ARTIST Questionnaire*. Recuperado de http://www.artist-embedded.org/docs/Events/2003/ICD/Trends/Dave2003.