Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

DISEÑO DE LA ARQUITECTURA SOFTWARE DEL SISTEMA DE A BORDO DEL SATÉLITE UPMSAT-2

TRABAJO FIN DE MÁSTER

Jorge Garrido Balaguer

2013

Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingenieros de Telecomunicación

Máster Universitario en
Ingeniería de Redes y Servicios Telemáticos

TRABAJO FIN DE MÁSTER

DISEÑO DE LA ARQUITECTURA SOFTWARE DEL SISTEMA DE A BORDO DEL SATÉLITE UPMSAT-2

Autor
Jorge Garrido Balaguer

Director
Alejandro Alonso Muñoz

Departamento de Ingeniería de Sistemas Telemáticos

2013

Resumen

A lo largo de la historia del software, tanto la industria como posteriormente también la comunidad científica han perseguido una metodología definitiva para el diseño e implementación de los sistemas informáticos. Sin embargo, múltiples son los ejemplos de estas metodologías que durante estas últimas décadas han sido propuestas, estandarizadas, desarrolladas herramientas para implementarlas y finalmente abandonadas. Múltiples son también las causas del fracaso de cada una de ellas, siendo el exponencial crecimiento de la industria y sus crecientes necesidades factor común en el abandono de cada una de las propuestas.

Los sistemas de tiempo real y sistemas empotrados, si bien se caracterizan por los especiales requisitos funcionales y no funcionales, también presentan necesidades especiales en su ciclo de vida. Ya sea por necesidades del servicio, o por mandato de normativas al respecto, también el diseño y el desarrollo de estos sistemas deben seguir un especial proceso para su validación.

La Agencia Espacial Europea, como líder en este tipo de sistemas puso en marcha en 2004 un proyecto denominado ASSERT (Automated proof-based System and Software Engineering for Real-Time systems), cuyo principal propósito era el estudio y propuesta de una metodología específica para el diseño y desarrollo de estos sistemas. Posteriormente, este proyecto fue ampliado con el proyecto TASTE (The Assert Set of Tools for Engineering), cuyo objetivo era la implementación de un entorno de desarrollo que permitiera la puesta en práctica de la metodología propuesta en ASSERT.

La presente memoria recoge el trabajo realizado por el autor en el proceso de diseño e implementación del software del satélite UPMSat-2 mediante el uso de la metodología ASSERT/TASTE.

El proyecto UPMSat-2 es un proyecto de la Universidad Politécnica de Madrid que tiene por objetivo el diseño, implementación, lanzamiento, operación y mantenimiento de un microsatélite que sirva de plataforma de demostración y validación de diversos dispositivos y experimentos.

Abstract

During the last three decades, methodologies for designing and developing software have been proposed, implemented and abandoned endlessly. Time-to-market requirements, budgets constraints and a voracious industry have continuously led to new methodologies, based on different paradigms without finding a final solution for general software lifecycles.

Real-time and embedded systems are not only special in the functional and no-functional requirements of the final implementations of the systems. Furthermore, specific methodologies and thorough quality processes comprising the full system lifecycle are many times required by standards and normative for these systems to be validated.

The European Space Agency, as a leader in this kind of systems, launched in 2004 an ambitious project, called ASSERT (Automated proof-based System and Software Engineering for Real-Time systems), aimed to propose a specific methodology for developing real-time embedded systems. Furthermore, after the success of the former project, a new project was launched to provide a developing environment where the ASSERT process was easily followed. As a result, TASTE, The Assert Set of Tools for Engineering was developed.

This report reflects and summarizes the work done by the author in the design and implementation of the UPMSat-2 software. The UPMSat-2 is a project of the Universidad Politécnica de Madrid, leaded by the Ignacio da Riva Institute, aimed to design, build, launch and operate an academic microsatellite. The Real-Time system and Telematic Services Architecture group, of which the author is member, is responsible of the design, implementation and maintenance of the software and telecommunications system of the satellite.

Índice general

Índice de figuras

Índice de tablas

Índice de listados

Siglas

AADL - Architecture Analysis and Design Language

AAS - Anomalía del Atlántico Sur

ACN - ASN.1 Control Notation

ADC - Analog-Digital Converter

ADCS - Attitude Determination and Control System

ASN.1 - Abstract Syntax Notation One

ASSERT - Automated proof-based System and Software Engineering for Real-Time systems

BCD - Binary Coded Decimal

BER - Basic Encoding Rules

bps - Baudios por segundo

CASE - Computer-Aided Software Engineering

CCSDS - Consultative Committee for Space Data Systems

CTM - Control térmico

EEPROM - Electrically Erasable Programmable Read-Only Memory

ECSS - European Cooperation for Space Standardization

ESA - Agencia Espacial Europea

ESTEC - European Space Research and Technology Centre

FIFO - First In First Out

FPGA - Field Programmable Gate Array

FPU - Floating Point Unit

GUI - Graphic User Interface

I2C - Inter-Integrated Circuit

IDR - Instituto Ignacio da Riva

ITU-T - International Telecommunication Union, Telecommunication Standardization Sector

MARTE - Modeling and Analysis of Real-Time and Embedded systems

MDA - Model Driven Architechture

MDE - Model-driven engineering

MGM - Magnetómetro

MHz - Megaherzios

MM - Magnetómetros

MRAD - Monitorización del efecto de la radiación

MT - Magnetopares

MTS - MicroThermal Switch

OBC - On-Board Computer

OMG - Object Management Group

ORK - Open Ravenscar Real-time Kernel

PER - Packet Encoding Rules

PIM - Platform Independent Model

PSM Platform Specific Model

PUS - Packet Utilization Standard

SAE - Sociedad de Ingenieros de Automoción

SCT - Solar Cell Technology

SDL - Specification Description Language

SID - Source Identificator

SMA - Shape Memory Alloys

SOC - System On a Chip

SPARC - Scalable Processor ARChitecture

SPI - Serial Peripheral Interface

SRAM - Static Random Access Memory

SSS - software system specification

STRAST - Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos

TASTE - The Assert Set of Tools for Engineering

TBD - To Be Defined

TC - Telecomando

TM - Telemetría

TMTC - Telemetría y telecomando

UART - Universal Asynchronous Receiver-Transmitter

UHF - Ultra High Frequency

UI - Unnumbered Information frame

UML - Unified Modelling Language

UPM - Universidad Politécnica de Madrid

VHDL - Combinación de VHSIC y HDL, donde VHSIC es el acrónimo de Very High Speed Integrated Circuit y HDL es a su vez el acrónimo de Hardware Description Language.

WCET - Worst Case Execution Time

1 Introducción

1.1 Contexto del trabajo

El desarrollo del presente Trabajo de Fin de Máster se enmarca en el contexto del proyecto UPMSat-2[1]. Este consiste en el diseño, construcción, puesta en órbita y operación de un micro-satélite. Este satélite tiene como objetivo ser una plataforma de demostración tecnológica en órbita, llevando como carga de pago (*payload*) diferentes experimentos para su operación en el espacio. El proyecto abarca todas las etapas de la vida del satélite, desde su diseño e incluyendo su construcción, validación, lanzamiento y operación en órbita. Durante esta última fase se pondrán en marcha diversos experimentos científicos e industriales de diferente índole.

El proyecto está liderado por el Instituto Ignacio da Riva (IDR)[2] de la Universidad Politécnica de Madrid. En él participan además otros grupos de esta universidad, así como diferentes empresas del sector aeroespacial europeo.

El UPMSat-2 es un microsatélite de 50kg de peso, con estructura de cubo, con un tamaño de 0.5m de lado. El periodo de operación estimado es de dos años, siguiendo una órbita heliosíncrona a 600km de altura.

El trabajo del grupo STRAST abarca todo el ámbito telemático del proyecto. Así, el grupo es responsable tanto del desarrollo del software de los segmentos de vuelo y tierra de la misión, como del diseño del sistema de comunicación del satélite.

El autor tiene una implicación activa en el desarrollo del UPMSat-2. Su trabajo de Fin de Grado consistió en el desarrollo de manejadores de dispositivos para el computador de a bordo [1]. En la memoria de dicho trabajo se describe el proceso llevado a cabo para el diseño, implementación y prueba de los manejadores para los diferentes dispositivos hardware del satélite, entre ellos, la línea serie que comunicará el ordenador de a bordo con la radio, como se explicará más adelante. Tanto en el mencionado trabajo como en el presente, se hace uso de tecnología previamente desarrollada por el grupo, como es el núcleo para sistemas de tiempo real ORK[3] desarrollado dentro del mismo. El núcleo fue diseñado para ejecutar sobre computadores de la familia LEON, un conjunto de procesadores de 32 bits basados en la arquitectura SPARC, especialmente desarrollada para el ámbito espacial.

[1] Este proyecto ha sido parcialmente financiado por el Ministerio de Economía y Competitividad (MINECO) con el identificador TIN2011-28567-C03-01 (HI-PARTES).

[2] www.idr.upm.es.

[3] www.dit.upm.es/ork

1.2 Motivación

El trabajo expuesto en la presente memoria recoge el proceso llevado a cabo por el autor, apoyado por el resto del grupo de investigación, para complementar labor del grupo en el proyecto, diseñando e implementando el software de alto nivel del satélite.

El mismo se ha llevado a cabo siguiendo metodologías y haciendo uso de las herramientas de actualidad en el ámbito de este tipo de sistemas, sirviendo de demostración para las mismas.

1.3 Objetivos del trabajo

Los objetivos generales del trabajo han sido los siguientes:

- Realizar un estudio y diseño del sistema de comunicaciones del satélite, incluyendo las características del radioenlace y los protocolos de gestión del mismo.
- Seguir la metodología ASSERT haciendo uso de las herramientas TASTE para el diseño, implementación y validación del sistema telemático UPMSat-2, demostrando la capacidad de las mismas para el desarrollo de un proyecto real.

Para llevar a cabo estos objetivos generales, se han definido una serie de objetivos detallados:

- Colaborar en la redacción de los diferentes requisitos concernientes al software del UPMSat-2.
- Evaluar las implicaciones de los requisitos relativos al software del satélite UPMSat-2.
- Definir los servicios del sistema de a bordo del satélite UPMSat-2 a partir de la especificación de requisitos.
- Modelar, mediante las herramientas propias del entorno TASTE, los diferentes tipos de datos a emplear en el software del UPMSat-2. Entre ellos, se encuentran los mensajes de telecomando y telemetría de los servicios previamente definidos.
- Siguiendo la metodología del ASSERT generar, a partir de la propuesta existente, una arquitectura software para el sistema de a bordo satélite.
- En base a los diseños generados hacer uso de las funcionalidades de TASTE para generar los esqueletos de código que faciliten la posterior implementación del sistema.
- Implementar un subconjunto de los componentes del software del sistema de acuerdo a la especificación de requisitos actual.
- Realizar las pruebas pertinentes a las partes implementadas.

1.4 Estructura del documento

La presente memoria se estructura de la siguiente manera: la sección 2 presenta el estado de la técnica en el diseño basado en modelos, y se realiza un pequeño estudio de la metodología y conjunto de herramientas ASSERT/TASTE. En la sección 3 se introduce el proyecto UPMSat-2.

Las siguientes secciones abordan las diferentes fases del trabajo: la sección 4 presenta el trabajo realizado en el modelado de datos del satélite, incluyendo la aplicación de los estándares pertinentes y la definición de los servicios de a bordo. La sección 5 presenta el modelo de alto nivel del software del satélite, desarrollado en la herramienta, sirviendo de documentación también del mismo. La sección 6 presenta el modelo de despliegue sobre la plataforma, la sección 7 aborda la implementación de parte del software del satélite y la sección 8 presenta un útil método de validación del sistema.

Finalmente la sección 9 presenta las conclusiones del trabajo realizado, las dificultades encontradas en durante el desarrollo del mismo, así como líneas de trabajo futuro.

2 Revisión tecnológica

2.1 Ingeniería basada en modelos

La ingeniería basada en modelos (o Model-driven engineering, MDE [2]) es una metodología de desarrollo de software enfocada a la generación de modelos como base del diseño de software. Estos modelos son abstracciones de las actividades, objetos y entidades involucradas en un determinado ámbito donde el software debe desarrollar una función concreta.

El objetivo de esta metodología es maximizar la compatibilidad entre sistemas y la reducción de costes, al reusar modelos estandarizados, simplificar el proceso de diseño, aplicando patrones al dominio y generar un ambiente de colaboración entre desarrolladores, dada la facilidad ofrecida para la reutilización.

La metodología MDE está claramente influenciada por las herramientas conocidas como CASE (Computer-Aided Software Engineering) desarrolladas en los 80's. En la actualidad existe una organización, el Object Management Group (OMG)[4], que desde 2001 trata de unificar procesos y lenguajes de esta nueva tendencia bajo la definición de Model Driven Architechture (MDA).

MDA sigue el concepto de abstracción del modelo a la plataforma tecnológica que soporte la solución. Para ello se debe generar un modelo de dominio, también conocido como modelo independiente de plataforma o PIM (Platform Independent Model). Este modelo debe representar únicamente lógica de negocio. Este modelo debe completarse con un modelo específico de la plataforma, o Platform Specific Model (PSM), en el cual se aplica el modelo de dominio a un modelo de plataforma concreto, obteniéndose así un modelo completo a partir de diferentes abstracciones reutilizables.

A pesar de los esfuerzos por parte de la OMG de estandarizar y unificar las metodologías basadas en modelos, su enfoque, centrado en un único lenguaje (UML, Unified Modelling Language), no ha sido uniformemente adoptado en todos los ámbitos de la industria del software y de la informática en general. Uno de los ámbitos en los cuales han surgido alternativas a esta metodología es la industria de los sistemas empotrados y de tiempo real.

Los sistemas informáticos empotrados son aquellos que operan como una parte de un sistema ingenieril más grande al cual supervisan y controlan. Es el caso del sistema informático del satélite UPMSat-2, en el cual el valor no reside en el sistema informático sino en sistema más amplio, el satélite. La interacción entre ambos sistemas

[4] www.omg.org

se lleva a cabo por medio de sensores y actuadores, dispositivos de hardware que permiten analizar el comportamiento del sistema y operar sobre él.

Los sistemas de tiempo real son sistemas que se caracterizan por sus requisitos temporales. En ellos, su correcto funcionamiento no sólo depende de la corrección lógica del resultado, sino también del instante de tiempo en el que se manifiesta. De este modo, un resultado lógicamente correcto obtenido fuera del plazo de respuesta no es válido. Un ejemplo sencillo de un sistema de tiempo real es el ABS de un coche. Este sistema debe accionar y liberar el freno de cada rueda cuando se realiza un frenazo, y un retraso de centésimas de segundo puede hacer que la rueda patine y el vehículo colisione.

Estos tipos de sistemas tienen características y requisitos diferentes de los sistemas informáticos convencionales. A pesar de que UML ofrece una extensión para el modelado de estos sistemas, MARTE (Modeling and Analysis of Real-Time and Embedded systems) [3], determinados sectores de la industria decidieron desarrollar sus propias metodologías y herramientas para el facilitar el ciclo de vida de estos sistemas. Un importante actor en el sector que tomó esta iniciativa fue la Agencia Espacial Europea (ESA).

2.1.1 ASSERT - TASTE

El proyecto ASSERT[4,5](Automated proof-based System and Software Engineering for Real-Time systems), liderado por la Agencia Espacial Europea y formado por un consorcio de 28 socios del sector aeroespacial tuvo como principal objetivo el desarrollo de métodos más fiables y robustos para el desarrollo de sistemas de software embarcado. Esta metodología se conoce como ASSERT process, siendo un conjunto de métodos y directrices con los que abordar el desarrollo, dirigido por modelos, de software embarcado que asegure su validez respecto a la especificación de requisitos.

TASTE es un conjunto de herramientas de código abierto para el desarrollo de sistemas empotrados y de tiempo real. Estos sistemas suelen tener requisitos de tiempo real crítico y, como parte de su validación, hay que realizar un análisis de planificabilidad para demostrar que tendrán un comportamiento temporal correcto. TASTE incluye un conjunto de restricciones que aseguran que los sistemas desarrollados son compatibles con el perfil de Ravenscar [6,7]. El perfil de Ravenscar es un conjunto de restricciones a la parte concurrente del lenguaje de programación Ada, a fin de que el código generado sea analizable temporalmente y tenga un comportamiento determinista. Aunque se definió originalmente para el lenguaje Ada, su modelo computacional se puede extrapolar a otros paradigmas de programación concurrente como POSIX.

El conjunto de herramientas TASTE está enfocado al desarrollo basado en modelos [2], siguiendo la metodología desarrollada en el proyecto ASSERT. En ella se describen las siguientes fases:

- Modelado del sistema.
- Transformación del sistema modelado a una arquitectura software de tiempo real.
- Análisis de factibilidad del sistema.
- Generación automática de código, implementación y compilado
- Validación de la implementación del sistema.

En las siguientes subsecciones se presentan en mayor detalle cada una de estas fases y las tecnologías implicadas en cada una de ellas.

Fase de modelado

La primera fase del desarrollo propuesta por TASTE es el modelado del sistema. El objetivo de esta fase es generar un modelo homogéneo para el sistema en su conjunto, obviando la naturaleza heterogénea del hardware y los artificios software necesarios para su implementación. El proceso de modelado por tanto debe basarse únicamente en una arquitectura software preliminar como la presentada en la figura 1, fruto del análisis conjunto del sistema objetivo por parte de los ingenieros del sistema y expertos en software.

Esta fase del proceso es posiblemente la más compleja. Por ello, se hace uso de diversos lenguajes como se presentará a continuación.

El nexo de unión entre los diferentes lenguajes usados el desarrollo es AADL (Architecture Analysis and Design Language), un lenguaje de descripción de arquitecturas estandarizado por la Sociedad de Ingenieros de Automoción (SAE). AADL está definido por un núcleo del lenguaje que define una notación única tanto para características de hardware, software y comportamiento del sistema. AADL presenta una gran flexibilidad a la hora de modelar sistemas concretos, al poderse declarar características específicas usando propiedades (*properties* en el lenguaje). Además, AADL se puede extender mediante dos métodos: extendiendo el conjunto de propiedades del sistema con propiedades definidas por el usuario y mediante la adición de anexos al lenguaje. Por el momento existen cuatro anexos, todos ellos relevantes en el ámbito de los sistemas que se pretende desarrollar con TASTE:

- *Behaviour annex*, que incluye soporte para máquinas de estados.
- *Error-model annex*, que especifica diversos aspectos sobre el tratamiento y propagación de errores.
- ARINC653 *annex*, que define patrones de modelado para sistemas de aviónica.
- *Data-Model annex* para el modelado de tipos de datos mediante AADL.

Sin embargo, el conocimiento del lenguaje AADL no es necesario para el usuario de TASTE, ya que, aunque es posible hacerlo, no es necesario manipularlo directamente. Por contra, las diferentes herramientas de modelado transforman automáticamente a

AADL las características del sistema especificadas por el usuario a medida que se editan los diferentes modelos o vistas.

Las vistas son, en el conjunto de herramientas de TASTE, cada una de las abstracciones para el modelado del sistema. Existen cuatro: *Data view*, *Interface view*, *Deployment view* y *Concurrency view*.

- *Data view*: en un primer paso se propone modelar los tipos de datos a usar en el sistema. Si bien sólo se exige estrictamente el modelado de aquellos datos que se vayan a usar posteriormente como parámetros en las diferentes interfaces, resulta altamente recomendable le especificación temprana de todos aquellos tipos a usar relacionados con la lógica del sistema. Esta especificación es abstracta y no impide que cada módulo de software use los tipos propios del lenguaje en el que finalmente se implemente.
  El lenguaje seleccionado en TASTE para llevar a cabo el proceso de modelado fue ASN.1 [8]. ASN.1 es un lenguaje estandarizado por ISO y la ITU-T que permite el modelado de tipos y estructuras de datos, tanto desde el punto de vista semántico como de codificación. Las especificaciones generadas mediante ASN.1 son independientes de los lenguajes de programación, de las plataformas hardware e incluso de la codificación de las estructuras de datos definidas. ASN.1 se encarga por defecto de generar una codificación óptima a los tipos especificados. Sin embargo, también es posible, mediante la extensión ACN, definir el formato de codificación a nivel de bit. Esta cualidad resulta de alto interés en este tipo de sistemas, en especial para el manejo de las interfaces con sensores y actuadores.
- *Interface view*: en un segundo paso se propone el modelado de las interfaces software mediante la vista de interfaz. Haciendo uso de una herramienta gráfica, se definen las diferentes funciones del sistema. Estas funciones pueden ser agrupadas en contenedores, que si bien no representan ninguna información efectiva, hacen más sencilla e intuitiva la edición del modelo. Las interacciones entre funciones se modelan mediante interfaces provistas y requeridas. Estas últimas se caracterizan por un conjunto de propiedades no funcionales (entre ellos los requisitos temporales de la tarea). Este conjunto de propiedades no funcionales depende del tipo de tarea que representen. Así, por ejemplo, en el caso de tareas cíclicas se ha de definir su periodo, mientras que el caso de tareas esporádicas se debe dar un valor de tiempo mínimo entre llegadas.
  Del mismo modo que en el caso de la edición del modelo de datos el modelo se convierte a AADL siendo accesible para uso en el resto de herramientas del entorno.
- *Deployment view:* Una vez definidas las diferentes funciones de software y sus dependencias entre ellas se debe proceder a describir la arquitectura hardware del sistema. Mediante una herramienta gráfica similar a la utilizada en el paso anterior se despliegan diferentes elementos sobre el modelo. Los dos elementos básicos de este modelo son los *Processor Boards* y los *buses* que interconectan a los primeros entre sí. Cada *processor board* tiene

al menos un procesador (caracterizado por su arquitectura) y un módulo de memoria principal. Adicionalmente, un *processor board* puede tener más de un módulo de procesador o memoria, además de varios módulos de *Drivers* mediante los cuales se conecta a los diferentes *buses*. Dentro de cada procesador se define al menos una partición, caracterizada por el sistema operativo o núcleo que ejecutará sobre ella. Finalmente, el usuario asigna las funciones software declaradas en la Interface view a las particiones en las que se desplegarán.

- *Concurrency view:* analizando automáticamente los modelos generados tanto en el *Interface view* como el *Deployment view*, TASTE genera una nueva especificación AADL que recoge las características de concurrencia y rendimiento del sistema, pudiendo analizar al planificabilidad del mismo mediante la herramienta integrada Cheddar[5].

Generación de código

Una vez modelados los diferentes módulos o funciones software del sistema en la *Interface view*, TASTE permite generar automáticamente tanto los esqueletos de código de cada función como los *scripts* necesarios para su posterior compilación. De igual modo, genera las definiciones de los tipos declaradas en el *Data view* para cada uno de los diferentes lenguajes soportados por TASTE. Estos lenguajes son los siguientes:

- SDL (Specification Description Language)[15], lenguaje de especificación de sistemas complejos mediante la comunicación entre módulos independientes por medio de señales. SDL está estandarizado por la ITU-T en el estándar Z.100.
- Simulink[6], un entorno de modelado, simulación y análisis de sistemas dinámicos. TASTE permite la implementación de funciones mediante modelos generados con esta herramienta.
- Ada [11], lenguaje de programación orientado a objetos y fuertemente tipado, usado principalmente en el entorno de los sistemas de tiempo real y sistemas empotrados.
- C, lenguaje de programación de propósito general.
- SCADE[7], herramienta de diseño y modelado de aplicaciones de alta criticidad para sistemas empotrados.

Además de estos lenguajes para la implementación de software, las funciones de la *Interface view* pueden ser declaradas con otros lenguajes. Uno de ellos es VHDL, permitiendo dentro del propio entorno definir elementos hardware en este lenguaje, integrados con el resto del sistema.

[5] http://beru.univ-brest.fr/~singhoff/cheddar/
[6] http://www.mathworks.es/products/simulink/
[7] http://www.esterel-technologies.com/products/

Compilación

Mediante los *scripts* generados automáticamente por TASTE se puede compilar el código de las diferentes funciones con los compiladores incluidos en la herramienta y enlazarlos para producir los ejecutables, uno por cada partición declarada en el *Deployment view*.

En esta fase, se hace uso de un middleware específico, conocido como PolyORB-HI encargado de relacionar las diferentes primitivas del código generado a aquellas ofrecidas por el sistema operativo seleccionado para cada partición. Este middleware se ofrece en dos variantes: una en Ada, que se implementa mediante un *run-time system* que da soporte al perfil de Ravenscar y por tanto el compilador y el *run-time* realizan comprobaciones del código generado para asegurar que se cumplen las restricciones, y otra variante en C/POSIX, en el cual las restricciones las comprueba el propio PolyORB-HI. El uso de una u otra variante depende de los requisitos específicos de cada proyecto en términos de uso de memoria, rendimiento, disponibilidad de drivers, etc.

Validación

TASTE ofrece diferentes herramientas para la validación del sistema generado. En primer lugar, TASTE puede generar automáticamente pruebas de codificación y decodificación de los diferentes tipos definidos. Dado que a lo largo del sistema un mismo valor puede pasar por elementos software desarrollados en diferentes lenguajes, y por elementos hardware con diferente representación en memoria de valores numéricos, esta característica resulta de gran utilidad.

Por su parte, en el ámbito de la *Interface view* se pueden declarar funciones del tipo GUI. Estas funciones generan un ejecutable extra, consistente en una interfaz gráfica en la cual se pueden tanto mostrar los valores recibidos por las interfaces ofrecidas como enviar diferentes valores por las interfaces requeridas. En esta interfaz gráfica se puede, además almacenar las interacciones con el sistema, generando un conjunto de pruebas automático basado en las pruebas realizadas previamente de forma manual, característica de utilidad para la realización de pruebas de regresión. El uso habitual que se hace de esta funcionalidad es la de simular la estación de tierra, declarando una interfaz ofrecida para recibir la telemetría del satélite, y una interfaz requerida para enviar telecomandos al mismo. De esta manera se puede interactuar manualmente con el sistema en desarrollo y analizar su comportamiento. Un ejemplo de esta funcionalidad se presenta en la sección 8.

Otra interesante característica para la validación del sistema es la disponibilidad de la herramienta PeekPoke. Mediante esta herramienta se puede monitorizar en tiempo de ejecución las diferentes variables del sistema. Esta herramienta, en conjunción con la presentada anteriormente, forman una potente combinación para el análisis del comportamiento del sistema.

3 Características del satélite UPMSat-2

3.1 Misión

El UPMSat-2 es un proyecto de la Universidad Politécnica de Madrid que tiene el objetivo de desarrollar un micro-satélite satélite experimental. Su función es la de servir como demostrador tecnológico de las capacidades técnicas de los diversos grupos involucrados en el proyecto. El Instituto Universitario de Microgravedad "Ignacio da Riva" lidera el proyecto. El Grupo de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST) es el encargado del diseño e implementación del software, tanto del segmento de tierra como del segmento de vuelo. Éste último ejecutará sobre un único OBC del tipo LEON3[13]. El software hará uso de la cadena de compilación GNATforLEON incluyendo Open Ravenscar Real-time Kernel (ORK)[10].

El proyecto cuenta con un antecedente natural, el satélite UPMSat-1 [11], lanzado el 7 de julio de 1995, desarrollado por un grupo de profesores del Laboratorio de Aerodinámica de la Escuela Técnica Superior de Ingenieros Aeronáuticos de la UPM. Con un objetivo similar al del actual satélite, si bien a un nivel de complejidad menor técnica y organizativamente, el proyecto fue un éxito. El satélite fruto de este proyecto fue puesto en órbita en el vuelo V75 de un lanzador Ariane IV-40, como carga secundaria, y se mantuvo operativo durante 213 días, siguiendo una órbita de similares características a la del UPMSat-2, inscrito en el registro español ROLEU y en el de las Naciones Unidas como UPM-Sat1/ROLEU 4.

La carga de pago del satélite, o *payload* está compuesta por un conjunto de experimentos propuestos por diversas empresas, así como por los diferentes grupos de investigación de la UPM involucrados en el proyecto. Estos experimentos están en su mayoría enfocados al estudio del comportamiento de diferentes componentes en el espacio, para adquirir experiencia en vuelo en el caso de los productos más maduros, y obtener datos para el desarrollo de los menos evolucionados.

A continuación se describe brevemente cada uno de ellos, a fin de ilustrar la misión del satélite, así como justificar su influencia en el diseño que se presentará a continuación:

- MTS MicroThermal Switch: Este experimento, denominado "MicroThermal Switch" (MTS), ha sido propuesto y está siendo construido por IberEspacio. Su objetivo es demostrar el funcionamiento de un interruptor térmico en versión miniaturizada. Su función es evacuar el calor del componente al que se adhiere, haciendo uso de un radiador. De esta forma, se trata de evitar que el componente supere una cierta temperatura máxima, ajustada de antemano.

- SCT Solar Cell Technology: El departamento TEC-EPG Solar Generator Section del Centro de Tecnología e Investigación de la Agencia Espacial Europea (ESTEC) ha propuesto un experimento consistente en instalar en una de las caras del UPMSat-2 un conjunto de cinco células solares de nueva generación. Estas células solares de triple unión serán probadas en órbita (calibración, pruebas de degradación por rayos ultravioletas y oxígeno atómico) y formarán parte del subsistema de potencia del satélite.
- MGM Magnetómetro: Se ha llegado a un acuerdo con la empresa Bartington para probar este magnetómetro en vuelo. El experimento consistirá en medir el campo magnético terrestre y comparar los resultados con otras medidas tomadas con magnetómetros cualificados y células solares. Estos equipos forman parte del subsistema de control de actitud del satélite.
- MRAD Monitorización del efecto de la radiación: El experimento tiene por objetivo observar el efecto de la dosis de radiación recibida en órbita sobre el hardware del satélite. Para ello se hará un chequeo de errores en una parte de la memoria del computador cada cierto intervalo de tiempo y se mantendrá un registro de las posiciones de memoria dañadas, obteniéndose una estadística del número de daños en función del tiempo. Una vez procesados los datos, y deducida la posición del satélite en función del tiempo, se tratará de observar la conocida como anomalía del Atlántico Sur (AAS). Esta anomalía es consecuencia de la depresión del campo magnético terrestre en la zona y su efecto más notable es el aumento de la radiación. Esta radiación es variable según la altura, presentado en la altitud prevista para el satélite una intensidad intermedia, inferior a la presentada en capas inferiores. Como consecuencia de esta radiación, los satélites con inclinaciones orbitales entre 35º y 60º han de estar especialmente protegidos frente a esta radiación, siendo el ejemplo más notable la Estación Espacial Internacional[8].
- SMA Actuadores: La empresa ARQUIMEA desea validar en vuelo una carga útil en la que combinará actuadores (Pin-Puller y REACT) basados en aleaciones con memoria de forma (SMA, Shape Memory Alloys) para demostración en órbita su funcionamiento en la actuación del despliegue de dos booms. Los booms son dispositivos encargados de desplegar diferentes segmentos de los satélites, una vez estos han sido expulsados del vehículo lanzador (P.e. paneles solares en el caso de algunos satélites).
- RW Rueda de reacción: La empresa SSBV ha proporcionado al satélite, con el fin de ser validado en órbita, una rueda de reacción en miniatura, del tamaño adecuado para su aplicación en el control de actitud de pequeños satélites.
- SS6 Sensores solares: El Instituto de Energía Solar de la Universidad Politécnica de Madrid está desarrollando unas células solares especiales, de gran estabilidad, para emplearlas como sensores de Sol sencillos y económicos. El experimento consiste en medir la corriente generada por las mencionadas células, dispuestas sobre cada una de las caras del satélite, para

[8] http://en.wikipedia.org/wiki/International_Space_Station

determinar el ángulo que forma cada cara del satélite con la dirección del Sol. Los resultados se compararán con las medidas realizadas por los magnetómetros, a fin de comprobar su exactitud.

- CTM Control térmico: Este experimento también forma parte de un programa interno del IDR/UPM y consiste en obtener datos de funcionamiento del satélite y de su comportamiento térmico durante el periodo de operación. De este modo, el grupo de investigación relacionado podrá refinar los métodos de cálculo y diseño térmico en los que el Instituto Ignacio da Riva cuenta con una amplia experiencia.
- BOOM: El IDR/UPM ha desarrollado un Boom, que se quiere calificar y dar experiencia de vuelo, además de emplearlo para separar un magnetómetro del cuerpo del satélite una distancia del orden de 0.2 m, y así reducir el posible efecto de contaminación magnética en las medidas.
- MAC Control de actitud: Este experimento forma parte de un programa interno del IDR/UPM, para desarrollar esquemas robustos de control de actitud basados en el campo magnético, para aplicar en próximos vuelos. En el caso del UPMSat-2, este experimento controlará la actitud del satélite en periodos de tiempo reducidos y controlados, desactivándose el control de actitud nominal del satélite.

El satélite está previsto que esté listo para ponerse en órbita en otoño de 2014 y su fecha de lanzamiento definitiva dependerá de la disponibilidad de espacio como carga secundaria en lanzamientos comerciales o en lanzamientos de prueba de los nuevos lanzadores Vega de la Agencia Espacial Europea. Una vez puesto en órbita, el satélite se espera que tenga un periodo operativo de dos años.

Durante ese tiempo, el satélite describirá una órbita heliosíncrona a 700 kilómetros de altitud, con una inclinación entre 96,5º y 102,5º. Dadas las características, el periodo orbital del satélite es de aproximadamente 100 minutos, sobrevolando las proximidades de la estación de tierra cada 12 horas.

3.2 Características del hardware

Como se ha mencionado previamente, el software del satélite se ejecutará sobre un único computador, conocido como On-Board Computer (OBC). El hardware del mismo está basado en la familia de computadores LEON3, de la arquitectura SPARC, incluyendo memoria del tipo SRAM, una EEPROM para almacenamiento persistente, así como diversos dispositivos periféricos para la interacción con sensores y actuadores.

La familia LEON de procesadores implementa la arquitectura SPARC V8 [14] desarrollados inicialmente por la Agencia Espacial Europea (ESA) y posteriormente por Gaisler Research. Estos procesadores se encuentran descritos en lenguaje VHDL para su configuración y uso en los conocidos como circuitos integrados o *system on a*

chip (SOC) en los cuales la mayoría (o totalidad) de los componentes del sistema se encuentran en un mismo circuito integrado o chip.

El desarrollo de estos procesadores comenzó en el año 1997 por parte de la ESA con el fin de conseguir un diseño de procesador abierto, portable y no privativo, capaz de alcanzar los requisitos de rendimiento, compatibilidad y costes en los proyectos a llevar a cabo durante las siguientes décadas. Posteriormente, la empresa Gaisler Research, renombrada como Aeroflex Gaisler, paso a ser la encargada del desarrollo de nuevos modelos de procesadores LEON, entre ellos el LEON3, el procesador seleccionado para su uso en el UPMSat-2.

El procesador LEON3 supuso un avance frente a sus predecesores. Entre sus novedades más destacadas se halla el aumento en las etapas de su pipeline, que paso de las 5 etapas de su inmediato predecesor, el LEON2, a 7. Igualmente, la distribución incorporó un gran número de módulos sintetizables, que lo hacen de alto interés para el proyecto UPMSat-2. Entre estos módulos, los de mayor interés son el controlador de memoria SDRAM de 32 bits, los controladores para SPI, I2C y UART con colas FIFO, así como unidades de reloj, controladores de interrupción y puertos de entrada/salida.

Las características más relevantes del computador de a bordo son:

- Procesador LEON3 con FPU (unidad de coma flotante).
- 41 canales de entrada analógicos.
- Memoria SDRAM (tamaño por definir).
- Memoria no volátil de tipo EEPROM
- Comunicación interna
    - Bus SPI.
    - Bus I2C.
    - Línea serie RS-232 a 115200 baudios
- Comunicación con tierra
    - Frecuencia: UHF banda 400 MHz, con potencia reducida.
    - Capacidad de enlace: 19200 bps.

A pesar de que la versión de vuelo del OBC descrita sigue en proceso de desarrollo, se dispone de una versión de ingeniería en la cual se están probando los diferentes elementos tanto software como hardware para el modelo de vuelo. Este modelo de ingeniería está también compuesto de un procesador LEON3 sintetizado sobre una FPGA VIRTEX 5 de XILINX. Al desarrollarse sobre una FPGA, la mayor parte de los componentes se pueden sintetizar de forma idéntica a la versión de vuelo, con la salvedad de que el modelo de ingeniería no está protegido externamente a la radiación como sí lo estará el modelo de vuelo.

3.3 Características del software del satélite UPMSat-2

El software de a bordo está compuesto por los subsistemas que se muestran en la Figura 1. Éstos son:

- Monitor de la plataforma: este subsistema se encarga de controlar el estado de los diferentes componentes del satélite mediante chequeos periódicos. Éstos incluyen la medida de los voltajes y corriente de las baterías y paneles solares, la temperatura en diferentes puntos de la estructura y componentes del satélite, etc. El periodo de las medidas a tomar dependen tanto del estado de operación en el que se encuentre el satélite, como del tipo de medida a tomar.
- Telemetría y telecomando (TMTC): subsistema encargado de interactuar con el equipo hardware de telecomunicaciones para el envío de mensajes (telemetría) y recepción de órdenes (telecomandos) desde la estación de tierra.
- Sistema de determinación y control de actitud: subsistema encargado de analizar la orientación actual del satélite respecto a la tierra y calcular las posibles medidas a tomar para mantenerlo en los parámetros consignados. Este proceso se lleva a cabo mediante el procesado de las medidas del campo magnético de la Tierra en los tres ejes del satélite provistas por unos sensores específicos conocidos como magnetómetros (MM). Los correspondientes actuadores de este subsistema son los magnetopares (MT), electroimanes que generan un campo magnético que interactúa con el de la Tierra haciendo girar al satélite sobre su propio eje.

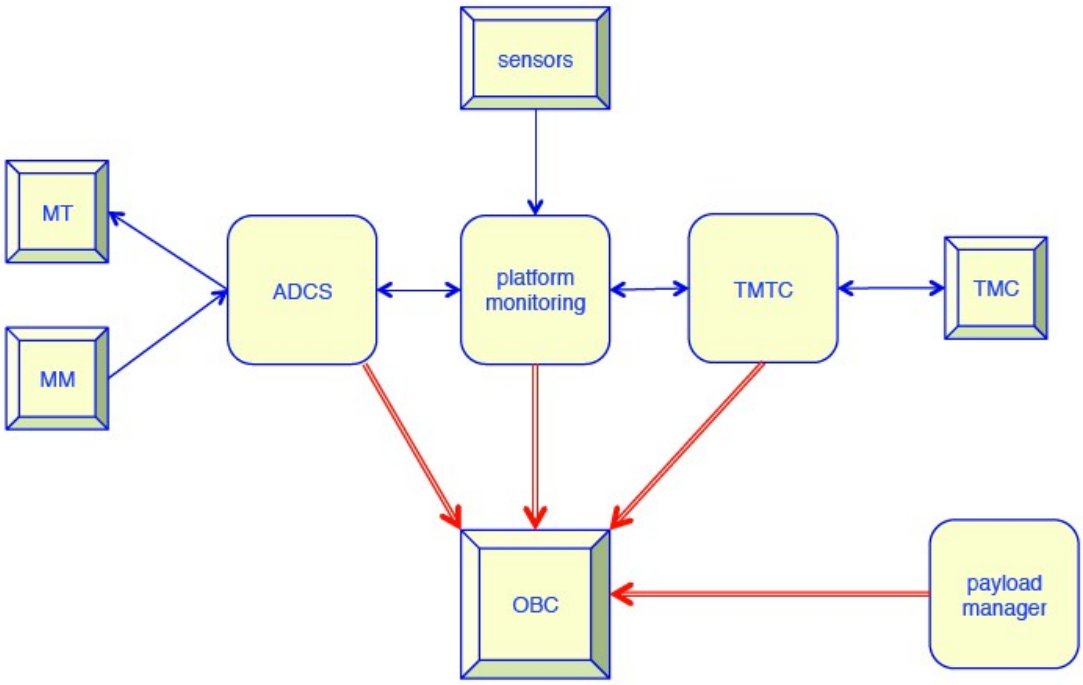

Figura 1 – Arquitectura software del satélite UPMSat-2.

- Controlador de experimentos: subsistema encargado de gestionar la puesta en marcha, ejecución, toma de datos y detención de los diversos experimentos a bordo del satélite, que constituyen su carga de pago ó *payload*, presentado previamente.

4 Modelado de datos

Como se describió anteriormente, la primera fase descrita en la metodología propuesta en el proyecto ASSERT es el modelado de los datos. El principal objetivo de esta fase es la definición de los diferentes tipos de datos que se intercambiarán en futuros modelos que representen otras abstracciones del software. Mediante la combinación de ASN.1 y su extensión ACN, esta fase de modelado puede subdividirse en dos etapas, siendo la segunda de ellas opcional.

4.1 ASN.1

En una primera fase, haciendo uso del lenguaje ASN.1 (Abstract Syntax Notation One), brevemente presentado previamente, se definen los diferentes tipos de datos mediante su definición semántica. De este modo, puede abstraerse toda la complejidad relacionada con la codificación de los datos, compatibilidad entre diferentes lenguajes y plataformas, etc.

En este lenguaje existen tres tipos de datos, a partir de los cuales se pueden definir tipos de la complejidad requerida por el usuario para modelar correctamente su sistema. Estos tipos son:

- Tipos primitivos: tipos que almacenan un único valor, entre los cuales encontramos:
  - El tipo INTEGER, usado para representar números enteros.
  - El tipo REAL, que representa tipos de datos de números decimales.
  - El tipo BOOLEAN, que representa datos que sólo pueden tomar los valores verdadero o falso.
  - El tipo OCTET STRING, consistente en una cadena de bytes.
  - El tipo BIT STRING, o cadena de bits.
  - El tipo NULL que representa la ausencia de valor.
- Tipos compuestos: tipos generados a partir de composiciones de los tipos primitivos y otros tipos compuestos. Los más habituales son:
  - SEQUENCE: lista ordenada de tipos de datos primitivos heterogéneos. En la práctica guarda gran similitud con estructuras de datos como *struct* del lenguaje C.
  - SEQUENCE OF: lista ordenada de datos del mismo tipo. Ofrece gran facilidad para declarar estructuras de gran tamaño, al no tener que declarar uno por uno los tipos. Por ejemplo, si quisiéramos declarar una tabla, mediante el tipo SEQUENCE declararíamos el formato de una fila de la tabla y mediante SEQUENCE OF (tipo de la fila declarado con SEQUENCE) se declara la agrupación de todas las filas para representar la tabla.

    - SET: es una lista no ordenada, en la que no se permiten tipos de datos repetidos para evitar ambigüedades.
    - SET OF: lista no ordenada de tipos de datos iguales.
    - CHOICE: tipo de dato para el cual se declaran diferentes tipos que puede adoptar en tiempo de ejecución. Por lo general, este tipo adoptado en tiempo de ejecución suele depender del valor que tomen otros campos de estructuras del tipo SEQUENCE.
- Tipos definidos: el propio lenguaje incluye tipos definidos para facilitar el proceso de declaración de los tipos. Estos tipos definidos en realidad no son más que casos concretos de los tipos presentados anteriormente, con un nombre más significativo. Dado que el lenguaje fue desarrollado y estandarizado por la ITU, estos tipos describen estructuras comunes en el ámbito de las telecomunicaciones, como por ejemplo:
    - IpAddres: representa una dirección IPv4. Por tanto su tamaño es de 4 bytes y su definición es un OCTET STRING de tamaño 4.
    - Counter: tipo entero al que sólo se le permite aumentar su valor, y una vez alcanzado el valor máximo declarado, un posterior incremento devuelve su valor a 0.
    - TimeTicks: entero de 32 bits que representa las centésimas de segundo transcurridos desde un evento concreto.

Los tipos presentados aquí no son todos los recogidos por el estándar. La lista completa de los mismos puede encontrarse en el estándar ITU-T X.680. [12].

El lenguaje ASN.1 es de fácil aprendizaje, dado que el conjunto de tipos que comúnmente se usan es reducido, y las reglas para la declaración de tipos son bastante sencillas. Un documento de ASN.1 comienza con la declaración del nombre del paquete que representa, seguido del número de declaraciones de tipos requeridos, concluyendo con la palabra reservada END.

Para definir tipos de datos primitivos la sintaxis es la siguiente: en primer lugar se declara el nombre del tipo, que ha de comenzar por letra mayúscula, seguido del símbolo de asignación ::= . A la derecha del símbolo de asignación se declara el tipo primitivo seguido si es necesario de información adicional sobre el mismo. Para los tipos primitivos presentados anteriormente se requiere la siguiente información adicional: para los tipos INTEGER y REAL se requieren los valores mínimo y máximo que pueden tomar, escribiéndose entre paréntesis y separados por dos puntos seguidos (P.e. (0..255) para INTEGER ó (0.0..255.0) para REAL). Para los tipos BIT STRING y OCTECT STRING, aunque el lenguaje no lo obliga, es recomendable, y en muchos casos necesario el dar un tamaño máximo, mediante SIZE (X) donde X es el tamaño en bit o bytes dependiendo del tipo. Finalmente los tipos NULL y BOOLEAN no requieren de información adicional.

En el caso de la declaración de tipos compuestos, se repite la estructura de los tipos primitivos hasta el nombre del tipo incluido. En el caso de tipos compuestos de un único tipo, como SEQUENCE OF y SET OF, se ha de definir el tamaño máximo mediante la fórmula SIZE (X), seguido del identificador del tipo del que están

compuestos. Este tipo que compone la estructura puede ser tanto un tipo definido por el lenguaje, como un tipo definido por el usuario. Por su parte, si se está declarando un tipo compuesto por elementos de varios tipos diferentes, como en el caso de SEQUENCE, SET o CHOICE, seguido de este identificador se abre corchetes, en cuyo ámbito se declaran cada elemento contenido en la estructura mediante su nombre (comenzando en este caso por minúscula) seguido por un espacio y el tipo del elemento. Nuevamente el tipo de cada elemento puede ser tanto un tipo definido por el lenguaje como por el usuario. Los diferentes elementos que componen la estructura se separan entre sí por comas, terminándose la definición de la estructura con un cierre de corchetes.

El conjunto completo de normas que rigen el lenguaje ASN.1 se pueden encontrar igualmente en el estándar ITU-T X.680 [12].

4.2 ACN

Como se ha mencionado con antelación, ASN.1 permite definir tipos y estructuras de datos abstrayendo cualquier tipo de información con respecto a cómo estos tipos son codificados. Por lo general, los compiladores para ASN.1, que convierten la especificación en código fuente para diversos lenguajes, hacen uso de reglas predefinidas para determinar la codificación de los tipos definidos en ASN.1. Así, lo más común es que estas herramientas permitan codificar bien siguiendo el estándar BER (Basic Encoding Rules)[17] o el estándar PER (Packet Encoding Rules)[18]. Estas reglas predefinidas limitan la capacidad de modelado de datos. En muchas casos es necesaria una codificación concreta al transformar la especificación a un lenguaje concreto. Esto puede suceder cuando se desea comunicar un sistema nuevo con otro existente, cuyos datos no han sido modelados con ASN.1, como por ejemplo un manejador de dispositivo o *driver*. Otro caso bastante corriente del ámbito de las telecomunicaciones es el modelado de datos acorde a protocolos. En la siguiente subsección se ilustrará esta utilidad con más detalle, aplicado al ámbito del satélite UPMSat-2.

Fruto de la necesidad de ofrecer también información sobre la codificación de los datos, se desarrolla en el ámbito del proyecto TASTE el lenguaje ACN (ASN.1 Control Notation). Este sencillo lenguaje permite al usuario la definición de la codificación deseada para cada tipo. Así, un documento ACN consta de los tipos definidos en ASN.1, uno por línea, seguidos de corchetes cuadrados ([ ]) dentro de los cuales se definen las características deseadas, separadas por comas.

La información que ACN permite incluir sobre la codificación es la siguiente:

- Tamaño: mediante el comando *size* seguido del tamaño (las unidades dependen del tipo en cuestión) se puede definir el tamaño deseado para el tipo de dato. Es responsabilidad del compilador impedir tamaños inferiores al mínimo necesario para el tipo de dato.

- Codificación de enteros y reales: diferentes tipos de codificación existen para valores numéricos. Para números enteros, ACN permite elegir entre: *pos-int*, valor positivo representado por la codificación binaria, *twos-complement*, o complemento a dos, ASCII, en el cual se codifica el valor en cuestión mediante el valor ASCII del símbolo positivo o negativo, seguido del valor de cada digito en ASCII o BCD (Binary Coded Decimal) en el cual cada dígito se codifica mediante 4 bits con su valor en binario. Para números decimales, se ofrecen las codificaciones estandarizadas IEEE754-1985-32 e IEEE754-1985-64.
- Codificación numérica a nivel de byte: aplicable a los enteros de tamaño fijo, define el orden en el que se codificarán los correspondientes bytes. Como es natural, las posibles opciones son *Big* o *Little endian*.
- Alineamiento: se puede forzar a que determinados valores se alineen al siguiente *byte* (8 bits), palabra (16 bits, *word*) o doble palabra (32 bits, *dword*).
- Codificación de valores: para los tipos ENUMERATED se pude forzar a que se codifique el valor del enumerado y no su índice (opción por defecto).
- Valores positivo y negativo: se puede forzar a que los valores positivo y negativo de los tipos booleanos se codifiquen mediante cadenas de bits en lugar del valor 1 para *true* y 0 para *false*.
- Indicador de presencia: para las estructuras CHOICE, permite definir la condición por la cual el tipo toma una u otra de las posibles especificaciones.

4.3 Modelado de datos aplicado al UPMSat-2

Haciendo uso de los lenguajes anteriormente descritos, y siguiendo con las directrices de TASTE, se ha realizado el modelado de los datos para el satélite UPMSat-2. Este modelado se ha hecho en base a las siguientes fuentes:

- Documento de especificación de requisitos software (SSS)[20] del proyecto UMSat-2.
- Manuales de referencia del hardware preseleccionado para el computador de a bordo.
- ECSS-E-70-41A, estándar sobre el software para misiones espaciales [16].
- Propuesta del autor para los mensajes de telemetría y telecomando a nivel de aplicación.

4.3.1 Telemetría y Telecomando en el satélite UPMSat-2

Uno de los principales objetivos del proyecto es la máxima adopción a lo largo del mismo de metodologías, técnicas, estándares y materiales de uso en el ámbito de la industria aeroespacial. El subsistema de telemetría y telecomando tanto del satélite

como de la estación de tierra persiguen igualmente este objetivo. De este modo se han seleccionado protocolos de amplio uso en el sector.

Nivel de enlace. Protocolo AX.25

En el ámbito de las comunicaciones entre satélite y la estación tierra, en proyectos a menor escala como el presente, es común hacer uso del protocolo AX.25 [19] para los niveles más bajos, físico y de enlace.

El protocolo AX.25 es un derivado de la suite de protocolos X.25. A pesar de haber sido desarrollado en la década de 1970 como un protocolo para radioaficionados. Fue estandarizado en 1984, resultando de gran utilidad para la comunidad. El protocolo ofrece varios beneficios para su uso en proyectos como el UPMSat-2: en primer lugar, al haber sido estandarizado hace ya 30 años existe gran cantidad de material referente al mismo. Por un lado existe gran cantidad de equipamiento hardware de calidad y a bajo coste disponible en el mercado. Por otro lado existen diferentes implementaciones disponibles para su uso. En segundo lugar el protocolo es bastante flexible en su utilización. El protocolo contempla tanto conexiones tanto *half* como *full-duplex*, además de más de un canal de comunicación, si el dispositivo hardware lo soporta. Además, el protocolo define tanto modos de funcionamiento orientados a conexión como modos no orientados a conexión. En el caso de radioenlaces como el del satélite UPMSat-2 lo común es hacer uso del modo no orientado a conexión mediante el intercambio de tramas UI (*Unnumbered Information frame*). Este modo de operación resulta bastante simple y sencillo, y tiene una baja sobrecarga u *overhead*, por lo que es muy adecuado para radioenlaces de baja capacidad como es el caso del UPMSat-2

Dado que el presente trabajo se centra en el diseño de alto nivel, y que existen diferentes implementaciones del protocolo, no se considera necesario entrar en mayor detalle en la memoria.

Paquetes de Telemetría y Telecomando. Estándar ECSS-E-70-41A.

Para la transferencia de telemetría y telecomando *end-to-end* entre aplicaciones del satélite y de la estación de tierra se hará uso de los paquetes estandarizados en el documento ECSS-E-70-41A[16]. Esta norma es el resultado de la puesta en común de los estándares publicados por el CCSDS y el Packet Utilization Standard (PUS) de la ESA. En él no sólo se detallan las estructuras tanto de los paquetes de telemetría (Figura 2 y Figura 3) y telecomando (Figura 4 y Figura 5), sino también define un interfaz para diferentes servicios, de los cuales también se hará uso para el modelado de la información de aplicación a transmitir.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Source Data	Spare (Optional)	Packet Error Control (Optional)
Version Number (=0)	Type (=0)	Data Field Header Flag	Applica-tion Process ID	Grouping Flags	Source Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	(see Note 2)

Figura 2 – Estructura de paquetes de telemetría estandarizado por ECSS. Adoptado en el ámbito del proyecto UPMSat-2. Extraído de [16].

Spare	TM Source Packet PUS Version Number	Spare	Service Type	Service Subtype	Packet Sub-counter	Destination ID	Time	Spare
Fixed BitString (1 bit)	Enumerated (3 bits)	Fixed BitString (4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Unsigned Integer (8 bits)	Enumerated	Absolute Time	Fixed BitString (n bits)
						← Optional →	← Optional →	← Optional →	← Optional →

Figura 3 – Detalle del *Packet Data Field* de los paquetes de telemetría. Extraído de [16].

En lo que a la estructura de telecomando se refiere, en las Figura 4 y Figura 5 se muestra la estructura de los paquetes definidos en el estándar y adoptadas en el ámbito del proyecto UPMSat-2. Adicionalmente, sirviéndose de la estructura de telecomandos definida en el estándar, se presenta un extracto tanto de los documentos ASN.1 como ACN que describen esta estructura.

Packet Header (48 Bits)							Packet Data Field (Variable)			
Packet ID				Packet Sequence Control		Packet Length	Data Field Header (Optional) (see Note 1)	Application Data	Spare	Packet Error Control (see Note 2)
Version Number (=0)	Type (=1)	Data Field Header Flag	Applica-tion Process ID	Sequence Flags	Sequence Count					
3	1	1	11	2	14					
16				16		16	Variable	Variable	Variable	16

Figura 4 – Estructura de paquetes de telecomando estandarizado por ECSS. Extraído de [16].

CCSDS Secondary Header Flag	TC Packet PUS Version Number	Ack	Service Type	Service Subtype	Source ID	Spare
Boolean (1 bit)	Enumerated (3 bits)	Enumerated(4 bits)	Enumerated (8 bits)	Enumerated (8 bits)	Enumerated (n bits)	Fixed BitString (n bits)
					← Optional →	← Optional →

Figura 5 – Detalle del *Packet Data Field* de los paquetes de telecomando. Extraído de [16].

Siguiendo la metodología propuesta, en primer lugar se debe especificar el modelo ASN.1. En él, para la definición de los diferentes campos, se ha hecho uso tanto de tipos de datos básicos (booleanos y enteros), como estructuras de datos complejas. Adicionalmente se requiere del tipo SEQUENCE de ASN.1. Igualmente, se hace uso del tipo CHOICE, una colección de tipos posibles para una variable, de los cuales se selecciona uno en tiempo de ejecución, generalmente en función del valor de otras variables. Como su puede ver en el código correspondiente a la estructura de los paquetes de telecomando, este tipo es de gran utilidad para modelar campos de tamaño variable, algo bastante corriente en la especificación de protocolos de comunicación. Así, en el ejemplo se ilustran algunos de los diferentes posibles contenidos para el *Application Data* en función del subsistema del satélite al que está dirigido el telecomando.

```
-- Telecommand structure

TelecommandPacketType   ::= SEQUENCE {

        tcPacketHeader TcPacketHeaderType,

        tcPacketDataField TcPacketDataFieldType

}

-- Packet Header

TcPacketHeaderType        ::= SEQUENCE {

        tcPHHeaderID TcPHHeaderIDType,

        tcPHSecuenceHeader TcPHSecuenceHeaderType,

        tcPHPacketLength TcPHPacketLengthType

}

-- Packet ID

TcPHHeaderIDType          ::= SEQUENCE {

        tcPHVersionNumber TcPHVersionNumberType,
```

```
        tcPHType TcPHTypeType,
        tcPHDataFielHeaderFlag TcPHDataFielHeaderFlagType,
        tcPHApplicationProcessID TcPHApplicationProcessIDType
}
TcPHVersionNumberType ::= INTEGER (0..7)
TcPHTypeType ::= INTEGER (0..1)
TcPHDataFielHeaderFlagType ::= INTEGER (0..1)
TcPHApplicationProcessIDType ::= INTEGER (0..2047)
…
-- Packet Data Field
TcPacketDataFieldType ::= SEQUENCE {
        tcDFH TcDFHType,
        tcApplicationData TcApplicationDataType,
        tcSpare TcSpareType,
        tcPacketErrorControl TcPacketErrorControlType
}
-- Data Field Header
TcDFHType ::= SEQUENCE {
        …
        Ack TcDFHTAckType
}
…
TcDFHTAckType ::= SEQUENCE {
                acceptanceAck BOOLEAN,
                startAck BOOLEAN,
                progressAck BOOLEAN,
                completionAck BOOLEAN
}
-- Application Data
TcApplicationDataType ::= CHOICE {
                TMRequest TMRequestType,
                ExperimentCommand ExperimentCommandType,
…
```

```
    }

    ...
```

Listado 1 – Extracto de la definición de la estructura de un paquete de telecomando mediante ASN.1.

Una vez definida la estructura del paquete, es conveniente completarla con directrices sobre su codificación. Dado que el protocolo define claramente el tamaño de cada campo, así como el significado de cada bit según su posición en el paquete, debemos asegurarnos que se codifican adecuadamente. En otros casos podemos simplemente confiar en la codificación automática del compilador de ASN.1, el cuál asegura que esta codificación hace uso del tamaño mínimo posible para cada tipo de dato declarado. Además, el conjunto de herramientas TASTE posee mecanismos para asegurar la consistencia de los valores que tomen cada tipo de datos, independientemente del lenguaje de programación y plataforma en los que se interpreten. Por tanto, en las situaciones en las que sea necesario, se pueden dar directrices de codificación para cada tipo de datos definido en el modelo ASN.1 mediante ACN. Estas directrices indican los valores deseados para el tamaño de los datos, el tipo de representación, o el tipo de representación de valores numéricos, entre otros. A continuación podemos observar un extracto del código ACN generado para el paquete de telecomandos.

```
    ...

      TcPHVersionNumberType[size 3, encoding pos-int]

      TcPHTypeType[size 1, encoding pos-int]

      TcPHDataFielHeaderFlagType[size 1, encoding pos-int]

      TcPHApplicationProcessIDType[size 11, encoding pos-int]

...
```

Listado 2 – Extracto de la definición de codificación en ACN de algunos de los tipos declarados en el documento ASN.1.

4.3.2 Definición de servicios del UPMSat-2. Telemetría y telecomando de nivel de aplicación.

Además de la estructura básica de los paquetes de telemetría y telecomando, existen gran cantidad de tipos a definir para el satélite. Un importante subconjunto de los mismos es la información que contendrán los mencionados paquetes de telemetría y telecomando en el campo *Application Data*, referente a cada uno de los diferentes servicios del satélite.

Buena parte de los servicios requeridos en el proyecto se encuentran ya descritos en el citado estándar ECSS-E-70-41A (Tabla 1). En ese caso, como ya se mencionó, se seguirá la propuesta del estándar. En aquellos casos en los que los servicios requeridos

por el satélite no se correspondan directamente con un servicio definido por el estándar, o en su descripción parte del mismo quede a decisión de diseño alguna de sus partes, se presenta la propuesta del autor para dichos servicios.

Cabe mencionar que los servicios definidos en el estándar, por lo general, están compuestos por un conjunto mínimo de subservicios a implementar y un conjunto opcional de funcionalidades avanzadas.

Service Type	Service Name
1	Telecommand verification service
2	Device command distribution service
3	Housekeeping & diagnostic data reporting service
4	Parameter statistics reporting service
5	Event reporting service
6	Memory management service
7	Not used
8	Function management service
9	Time management service
10	Not used
11	On-board operations scheduling service
12	On-board monitoring service
13	Large data transfer service
14	Packet forwarding control service
15	On-board storage and retrieval service
16	Not used
17	Test service
18	On-board operations procedure service
19	Event-action service

Tabla 1 – Servicios definidos en el estándar ECSS-E-70-41A.
El número de *Service Type* se corresponde con los campos de mismo nombre tanto en los paquetes de telemetría como de telecomando. Extraído de [16].

Housekeeping & diagnostic data reporting service

El primero de los servicios definidos en el estándar y de utilidad para el proyecto es el dedicado al envío de telemetría. Este servicio tiene el identificador de servicio número 3, como se puede ver en la Tabla 1. Su principal utilidad (y subconjunto mínimo o básico de implementación) es la de definir la estructura y contenido de los mensajes de telemetría referidos al *housekeeping* o monitorización del satélite. Adicionalmente se describe un número de subservicios para modificar parámetros de

la generación de mensajes de telemetría, definir nuevos tipos de mensaje, etc. En el caso del UPMSat-2 sólo se hará uso del subconjunto mínimo definido en el estándar, ampliado con dos subservicios propios para la petición de telemetría específica.

Housekeeping Parameter Report

El mencionado subconjunto mínimo está compuesto por un único subservicio: *Housekeeping Parameter Report*, con identificador de subservicio 25. En este subservicio, la composición del *Source Data* (paquetes de telemetría por tanto) definida por el estándar es la mostrada en la Figura 6. En ella se describen tres campos:

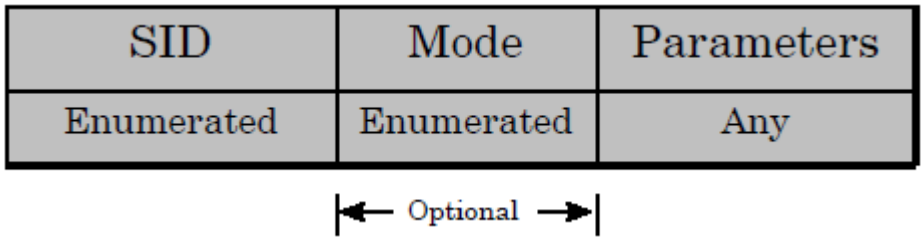

Figura 6 – Campos del Housekeeping Parameter Report.
Extraído de [16].

- SID: Campo enumerado que define el tipo de dato al que se refiere el paquete de telemetría. Su contenido es específico (y por tanto libremente definible) en el ámbito de cada misión. La propuesta del autor es mantener los tipos de telemetría descritos en el documento de requisitos. Únicamente tendría sentido subdividirlos en el caso de que el periodo de muestreo para diferentes valores del mismo tipo sea diferente. Por tanto, se consideran los siguientes valores:
  - Valor = 0 : Todas las medidas (solo en petición de telemetría en tiempo real definida más adelante).
  - Valor = 1 : Medidas de temperatura.
  - Valor = 2 : Medidas de tensión eléctrica.
  - Valor = 3 : Medidas de intensidad eléctrica.
  - Valor = 4 : Medidas de actitud de los magnetómetros.
  - Valor = 5 : Medidas de actitud de las células solares.
- Mode: Indica el modo en el que se ha generado el paquete de telemetría. Existen dos modos definidos en el estándar: periódico, en el cual los datos se almacenan para su envío por telemetría periódicamente, independientemente de su valor, y filtrado, en el cual se definen rangos de valores considerados normales, de modo que si todas las medidas de una determinada toma se encuentran dentro de estos rangos, ésta no es enviada a tierra, salvo que se durante un largo periodo de tiempo no se haya enviado medida alguna del tipo correspondiente. Los posibles valores son:
  - Valor = 0 : Modo periódico.
  - Valor = 1 : Modo filtrado. Valor fuera de rango.

    - Valor = 2 : Modo filtrado. Se ha excedido el tiempo mínimo entre medidas reportadas.
- Parameters: Campo en el cual se incluyen los valores de las medidas. Su contenido es de libre definición. Por tanto, la propuesta consiste en enviar un determinado número de medidas consecutivas (por definir) del mismo tipo. Siendo así, se requiere una reinterpretación del campo Mode previamente descrito. En el caso en el que el modo de operación sea periódico, el valor sigue siendo 0. Si se está operando en modo filtrado, el valor será 1 en el caso de que alguna de las medidas que se envían haya sido generada por encontrarse fuera del rango de valores previstos. Unicamente si todas y solo todas las medidas se hayan almacenado debido a que se ha cumplido el tiempo mínimo entre muestras almacenadas, el valor del campo Mode será 2.

  El contenido del campo será por tanto una sucesión de pares de la longitud (por definir) especificada de la siguiente estructura:
    - Time (32 bits): Valor del reloj de misión en el momento de toma de los valores de telemetría.
    - Values (dependiente del tipo de datos): Conjunto de valores para cada uno de los datos definidos en el documento de especificación de requisitos software (SSS).

Housekeeping Log Request

Subservicio añadido para la petición de telemetría de un determinado tipo. Siguiendo el estándar, el identificador de los subservicios específicos de misión deben estar comprendidos entre el 128 y el 255, proponiendo el autor el uso del identificador 128 para este subservicio en concreto. Dado que es posible que por iniciativa propia del software de a bordo no se envíe la totalidad de la telemetría disponible (por decisión de diseño para la correcta administración del tiempo de cobertura), se requiere poder solicitar el envío del log completo de datos de *housekeeping* de alguno de los tipos definidos previamente. Para ello se describe el contenido del campo *Application Data* de los mensajes de telecomando asociados a este subservicio:

- SID: Campo enumerado que define el tipo de dato al que se refiere el paquete de telemetría. Los posibles valores siguen aquellos definidos en el subservicio Housekeeping Parameter Report, con la salvedad del valor 0, que en este caso incluye a todos los tipos, solicitándose por tanto el envío de toda la telemetría disponible.
    - Valor = 0 : Todas las medidas.
    - Valor = 1 : Medidas de temperatura.
    - Valor = 2 : Medidas de tensión eléctrica.
    - Valor = 3 : Medidas de intensidad eléctrica.
    - Valor = 4 : Medidas de actitud de los magnetómetros.
    - Valor = 5 : Medidas de actitud de las células solares.

- Time (32 bits): Campo para solicitar la telemetría almacenada desde un instante de tiempo concreto. Su valor se comparará con el registrado en cada paquete de medidas de *housekeeping* y se enviarán aquellos con un valor posterior. En caso de que el campo Time tome el valor 0, se enviará toda la telemetría disponible.

Real-time Housekeeping Request

Subservicio añadido para la petición de telemetría en tiempo real. Se propone el uso del identificador 129. El contenido del campo *Application Data* de los mensajes de telecomando asociados a este subservicio:

- SID: Campo enumerado que define el tipo de dato al que se refiere el paquete de telemetría. Los posibles valores siguen aquellos definidos en el subservicio Housekeeping Parameter Report, con la salvedad del valor 0, que en este caso incluye a todos los tipos, solicitándose por tanto el envío de toda la telemetría disponible.
  - Valor = 0 : Todas las medidas.
  - Valor = 1 : Medidas de temperatura.
  - Valor = 2 : Medidas de tensión eléctrica.
  - Valor = 3 : Medidas de intensidad eléctrica.
  - Valor = 4 : Medidas de actitud de los magnetómetros.
  - Valor = 5 : Medidas de actitud de las células solares.

Event Reporting Service

Este servicio permite el envío de información sobre eventos que ocurren durante la operación del satélite. Su identificador de servicio es el 5 y su función es la de cubrir los requisitos de notificación de eventos que suponen la detección de una anomalía o fallo en algún elemento del satélite, así como de acciones autónomas del satélite (como puede ser el cambio de modo de operación). En el satélite UPMSat-2 se implementará el subconjunto básico, que define cuatro subservicios:

- Normal Progress Report: Subservicio con identificador 1, reporta acciones autónomas del satélite.
- Error Anomaly Report – Low/Medium/High Severity: Subservicios de notificación de errores, con identificadores del 2 al 4. La diferente gradación permite ofrecer diferentes niveles de prioridad para su envío y tratamiento en tierra.

La estructura del campo *Source Data* de los paquetes de telemetría asociados es la siguiente:

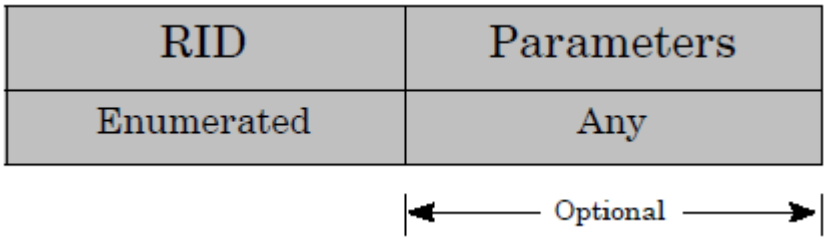

Figura 7 – Campos del Event Reporting Service.
Extraído de [16].

- RID: Report ID, junto con el identificador de aplicación y la severidad permite la identificación del evento en concreto.
- Parameters: Valores que añaden información sobre el evento en cuestión:
    - Time (32 bits): Valor del reloj de misión en el momento que se registra el evento.
    - Otros: dependiente de los tipos de evento descritos a continuación.

Por tanto, se requiere la identificación de los diferentes eventos que se pueden dar en los diferentes subsistemas, así como categorización de severidad y la definición de parámetros adecuados para su completa descripción:

- Orchestrator: para el orquestador de la plataforma se describen los siguientes eventos:
    - Normales:
        - StatusChange: Cambio de estado de operación del satélite
            - RID: 0
            - Parameters:
                - NewStatus (enumerado): Nuevo estado del satélite. Los posibles estados son: *launch, latency, initialization, commisioning, safe, beacon, nominal* y *experiment*.
                - Trigger (enumerado): Motivo por el cual se cambia de estado:
                    - TC(0) : ordenado por telecomando.
                    - LostCOMM(1) : Ha vencido el plazo máximo entre periodos de conexión.
                    - ExperimentTimer(2): Ha vencido el plazo dado al experimento.
                    - LowBattery(3): Se ha detectado estado bajo de baterías.
                    - CriticalBattery(4): Se ha detectado un nivel de baterías crítico.
                    - WatchdogTimer(5): El computador se ha reiniciado al expirar el temporizador de guardia.

                - LatencyTimer(6): Ha expirado el temporizador de estado de latencia.
                - SeparationTimer(7): Ha expirado el temporizador tras la separación del lanzador.
                - Other(8): Cualquier otra causa no contemplada por el resto de indicadores.
    - Medium Severity:
        - EventLogFull: Se he llenado la región de memoria dedicada a almacenar eventos pendientes de transmisión a tierra.
            - RID: 0
            - Parameters:
                - EventSeverity (enumerado): Cola de envío que se ha llenado:
                    - Nomal (0)
                    - LowSeverity (1)
                    - MediumSeverity (2)
                    - HighSeverity (3)

- Housekeeping Manager: El monitor de plataforma puede registrar los siguientes eventos:
    - Low Severity:
        - LowBattery: Se ha detectado nivel de baterías bajo.
            - RID: 0
            - Parameters:
                - BatteryLevel: valor concreto detectado.
    - Medium Severity:
        - SensorError: Se ha detectado un sensor que presenta un comportamiento erróneo.
            - RID: 0
            - Parameters:
                - SensorID (enumerado): Sensor que presenta el comportamiento anómalo.
                - ErrorID (enumerado): tipo de error.
                    - Timeout(0): Se ha excedido el tiempo de espera para conectar con el dispositivo.
                    - ValueOutOfRange(1): El dispositivo notifica valores fuera de rango de forma reiterada.
        - HousekeepingLogFull: Se ha llenado la región de memoria dedicada a un tipo de telemetría sin que sus datos hayan sido enviados a tierra. Por tanto se perderá información.
            - RID: 1
            - Parameters:

        - SID: Valor del SID definido en el Housekeeping Report Service para el tipo de telemetría cuyo Log se ha llenado.

    - High Severity:
        - CriticalBattery: Se ha detectado un nivel de baterías crítico. Se ha de cambiar de modo de operación del satélite de forma inmediata.
            - RID: 0
            - Parameters:
                - BatteryLevel: valor concreto detectado.
- ADCS:
- Experiment Manager: Gestor de los experimentos.
    - Normales:
        - ExperimentBegins: Notificación de inicio de experimento.
            - RID: 0
            - Parameters:
                - ExperimentID(enumerado).
        - ExperimentEnds:
            - RID: 0
            - Parameters:
                - ExperimentID(enumerado).
                - ResultValues: Valores resultado del experimento. Contenido a definir por cada responsable de experimento.
    - Medium Severity:
        - ExperimentError: Se ha detectado un error en alguno de los experimentos de a bordo.
            - RID: 0
            - Parameters:
                - ExperimentID (enumerado).
                - ErrorID (enumerado): Error detectado.
                    - ConnectionError(0): No se puede conectar con el experimento por la interfaz del mismo.
                    - ExperimentError(1): Se ha detectado una anomalía en el funcionamiento del experimento.
    - High Severity:
        - RWError: Dado que puede afectar gravemente al funcionamiento del satélite, un error que impida la desactivación/detención de la rueda de inercia del satélite se considera un error grave.
            - RID: 0
            - Parameters: Ninguno (aparte de Time).

- TMTC: Subsistema de comunicaciones.
  - Medium Severity:
    - LostCOMM: Ha expirado el tiempo máximo entre dos conexión con la estación de tierra.
      - RID: 0
      - Parameters:
        - LastConnection (32 bits): Valor del reloj de misión cuando se estableció comunicación con la estación de tierra por última vez.
    - NotTrustedSource: Se ha detectado la recepción de un telecomando del cual no se ha podido comprobar su origen.
      - RID: 0
      - Parameters: Ninguno (aparte de Time).
    - ReceptionErrors: Se ha detectado un número de paquetes de telecomando rechazados por ser considerados erróneos (al no corresponder con los códigos de redundancia) superior al máximo esperado (por definir):
      - RID: 0
      - Parameters:
        - Percentage: Porcentaje de paquetes descartados.

Memory Management Service

Este servicio, con identificador 6, permite tanto la modificación de áreas de memoria del computador de a bordo como el envío a tierra de áreas de memoria del mismo. En el estándar se describen dos posibles subconjuntos de implementación mínima, diferenciándose entre ellos en la forma en que estas áreas de memoria son direccionadas. Los posibles dos opciones son: mediante direcciones base previamente definidas y desplazamiento a partir de la dirección base o el uso de direcciones absolutas. Dada la reducida capacidad de la memoria del satélite, así como la mayor simplicidad de la implementación, se hará uso de las direcciones absolutas. Para implementar este subconjunto, se requieren los siguientes tres subservicios:

Load Memory using Absolute Addresses

Este subservicio (identificador 2) permite el envío desde la estación de tierra del contenido deseado para una o varias áreas de memoria en el computador de a bordo. Por tanto, los campos descritos a continuación serán el contenido del campo *Application Data* de los mensajes de telecomando referidos a este subservicio:

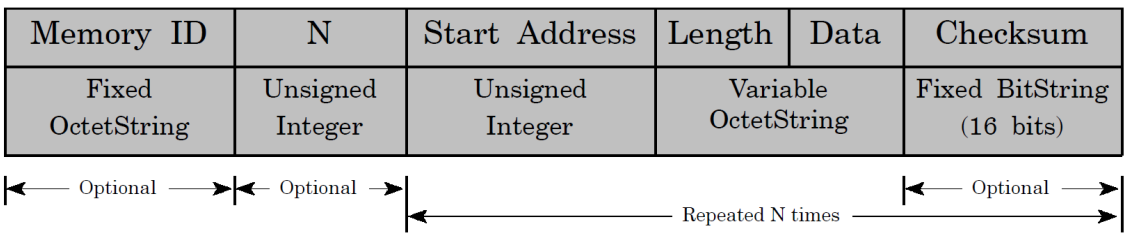

Figura 8 – Campos del subservicio Load Memory using Absolute Addreses.
Extraído de [16].

- Memory ID: Campo para identificar la memoria (dispositivo físico o lógico) a la que se refiere el telecomando. Dado que es opcional y la arquitectura del computador de a bordo del satélite UPMSat-2 maneja la entrada salida mapeada en memoria (es decir, existe un único espacio de memoria, compartida por la memoria principal y los diversos periféricos), este campo se omitirá, al no existir ambigüedad posible.
- N: indica el número de regiones de memoria a editar. Dado que es opcional, y este servicio debe usarse con extrema cautela, solo se contempla que se pueda enviar un único bloque de memoria por telecomando, omitiéndose por tanto este campo. Esto, además, simplifica enormemente la implementación del servicio.
- Start Addres: Dirección de memoria de comienzo de la zona de memoria a editar. Dado la arquitectura del computador, esta dirección debe estar alineada a byte.
- Length: Campo que describe la longitud (en bytes) del área de memoria a modificar (y por tanto la longitud del campo Data).
- Data: Contenido que debe insertarse en memoria.
- Checksum: Suma de comprobación del campo data. Dado que es opcional, que el paquete de telecomando en sí ya tienen *checksum* que incluye el Application Data, y que no se esperan regiones de memoria grandes en el campo Data, el campo Checksum se omitirá.

Dump Memory using Absolute Addresses

Subservicio (identificador 5) que permite solicitar desde la estación de tierra el envío de áreas de memoria del computador de a bordo. Por tanto, los campos descritos a continuación serán el contenido del campo *Application Data* de los mensajes de telecomando referidos a este subservicio:

Memory ID	N	Start Address	Length
Fixed OctetString	Unsigned Integer	Unsigned Integer	Unsigned Integer
← Optional →	← Optional →	← Repeated N times →	

Figura 9 – Campos del subservicio Dump Memory using Absolute Addreses.
Extraído de [16].

- Memory ID: Previamente se ha razonado la omisión de este campo.

- N: Del mismo modo que en el caso del subservicio Load Memory, solo se contempla una solicitud por telecomando, y por tanto, este campo es omitido.
- Start Address: Dirección de memoria base solicitada. Aplican las consideraciones previamente expuestas en el mismo campo del subservicio Load Memory.
- Lenght: Longitud (en bytes) de la petición de volcado de memoria.

Memory Dump using Absolute Addresses Report

Subservicio (identificador 6) que genera un mensaje de telemetría con el contenido de cierta área de la memoria del computador de a bordo. Por lo general, este servicio ejecuta como respuesta a una petición del subservicio Dump Memory using Absolute Addresses, pero también puede realizarse espontáneamente por el computador de a bordo, si ha sido configurado para ello. Estos son los campos del *Source Data* de los mensajes telemetría referidos a este subservicio:

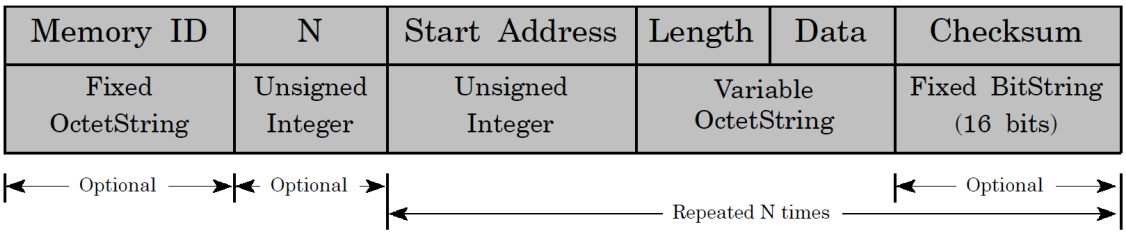

Figura 10 – Campos del subservicio Memory Dump using Absolute Addreses. Extraído de [16].

- Memory ID: Previamente se ha razonado la omisión de este campo.
- N: Previamente se ha razonado la omisión de este campo.
- Start Address: Mismo significado que en los anteriores subservicios de este servicio.
- Length: Mismo significado que en los anteriores subservicios de este servicio.
- Data: Mismo significado que en los anteriores subservicios de este servicio.
- Checksum: Previamente se ha razonado la omisión de este campo.

Function Management Service

Servicio (con identificador 8) que permite enviar desde tierra órdenes para la ejecución de acciones concretas de una aplicación determinada. En el ámbito del proyecto, este servicio resulta de utilidad para el envío de telecomandos al subsistema de experimentos, a fin de que ponga en marcha, detenga o genere mensajes de telemetría de un experimento.

El servicio solo define un subservicio, con el nombre de Perform Function e identificador de subservicio 1. La definición de los campos del *Application Data* de los mensajes de telecomando asociados a este servicio son los siguientes:

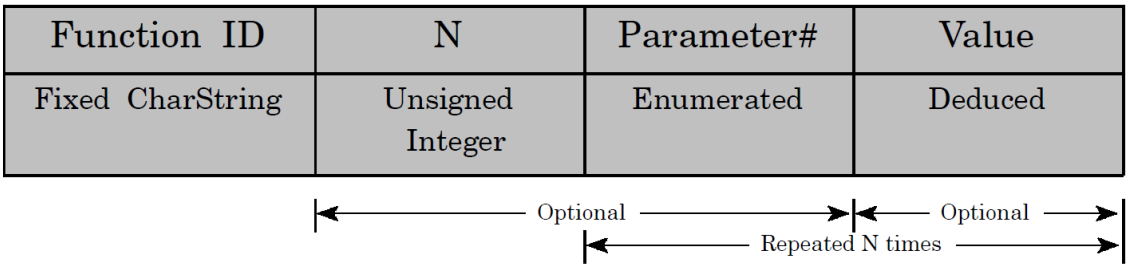

Figura 11 – Campos del servicio Function Management. Extraído de [16].

- Function ID: Campo que identifica la operación a realizar. Dado que el estándar ofrece total libertad para su definición, mediante este campo se comunicarán dos datos:
  - ExperimentID(enumerado): Identificador del experimento.
  - ExperimentOperation(enumerado): Operación a realizar. Los posibles valores son *start*, *stop* y *reportResults*.
- N: Número de parámetros que siguen en el mensaje.
- Parameter#: Identificador de parámetro del valor que se envía a continuación.
- Value: Valor del parámetro.

Con la actual definición de los diferentes experimentos, no se considera que vaya a ser necesario el envío de parámetros. Sin embargo, se documenta la estructura por si fuera necesario dada la evolución de la descripción de los experimentos.

OpenLink

Servicio con identificador 128 (el primero de los valores posibles para los servicios específicos de misión, no recogidos en el estándar.). Dada la órbita del satélite, el subsistema de telecomunicaciones solo tendrá cobertura con la estación de tierra durante determinados periodos de tiempo. Por tanto, debe establecerse un protocolo de adquisición de señal y comienzo del intercambio de mensajes de aplicación. Si bien la adquisición de señal es responsabilidad del protocolo de bajo nivel, AX.25, que incorpora mecanismos para ello, será la estación de tierra la que tome la iniciativa a nivel de aplicación, mandando el telecomando definido en este servicio. Una vez correctamente recibido este telecomando, el satélite considera la comunicación completamente establecida y comienza el envío de la telemetría básica definida en el documento de especificación de requisitos software, salvo que el contenido del telecomando indique lo contrario. Esto puede ser de utilidad en situaciones en las cuales se requiera de gran cantidad de tiempo de conexión, como puede ser la modificación de grandes áreas de memoria. Para ello, se define el campo del

Application Data de los mensajes de telecomando asociados a este servicio (el identificador de subservicio será el 1):

- SendTM (Booleano): Valor booleano que indica si se debe enviar la telemetría básica como respuesta al telecomando OpenLink.

Petición de reenvío de mensajes

Identificador de servicio 129. Debido a errores de transmisión en el mensaje, puede que o bien se detecten saltos en el número de secuencia de un telecomando o telemetría, o bien puede detectarse un error en el contenido del mensaje en el proceso de verificación de su suma de comprobación. En cualquiera de los casos, puede considerarse necesario la solicitud de la retransmisión de dicho paquete. En ese caso, debe mandarse un mensaje de telecomando o telemetría (dependiendo de si se trata de la estación de tierra o el satélite) con el siguiente contenido en el campo *Application Data* o *Source Data* (el identificador de subservicio será el 1):

- SequenceCount: Número de secuencia del mensaje perdido/deteriorado del cual se solicita su retransmisión.

Modificación de valores de configuración

Dada la especificación de requisitos, el sistema requiere de un servicio encargado de la modificación de valores de configuración software. Entre ellos, por ejemplo, se puede considerar el cambio de modo de operación del satélite. Para ello se define el presente servicio, que, siguiendo el estándar, debe tener un identificador entre el 128 y el 255 (se propone usar el 130, para dejar libres 128 y 129 a posibles servicios más genéricos).

Para la definición del servicio se seguirá la estructura de subservicios que presenta el estándar, definiéndose los siguientes subservicios:

Cambio de modo de operación

Dada la relevancia y especificidad de este parámetro de configuración, se describe un subservicio para el mismo. Este tendrá el indicador de subservicio 1. A continuación se detalla la propuesta para el campo *Application Data* de los mensajes de telecomando asociados a este subservicio.

- NewStatus (enumerado): Nuevo estado del satélite.

Nótese que se puede indicar el cambio de estado a cualquiera de los posibles estados operacionales del satélite. El software de a bordo debe comprobar que es posible, siguiendo el diagrama de posibles estados de la Figura 12, la transición de estados solicitada, descartándose esta si no es posible realizarse.

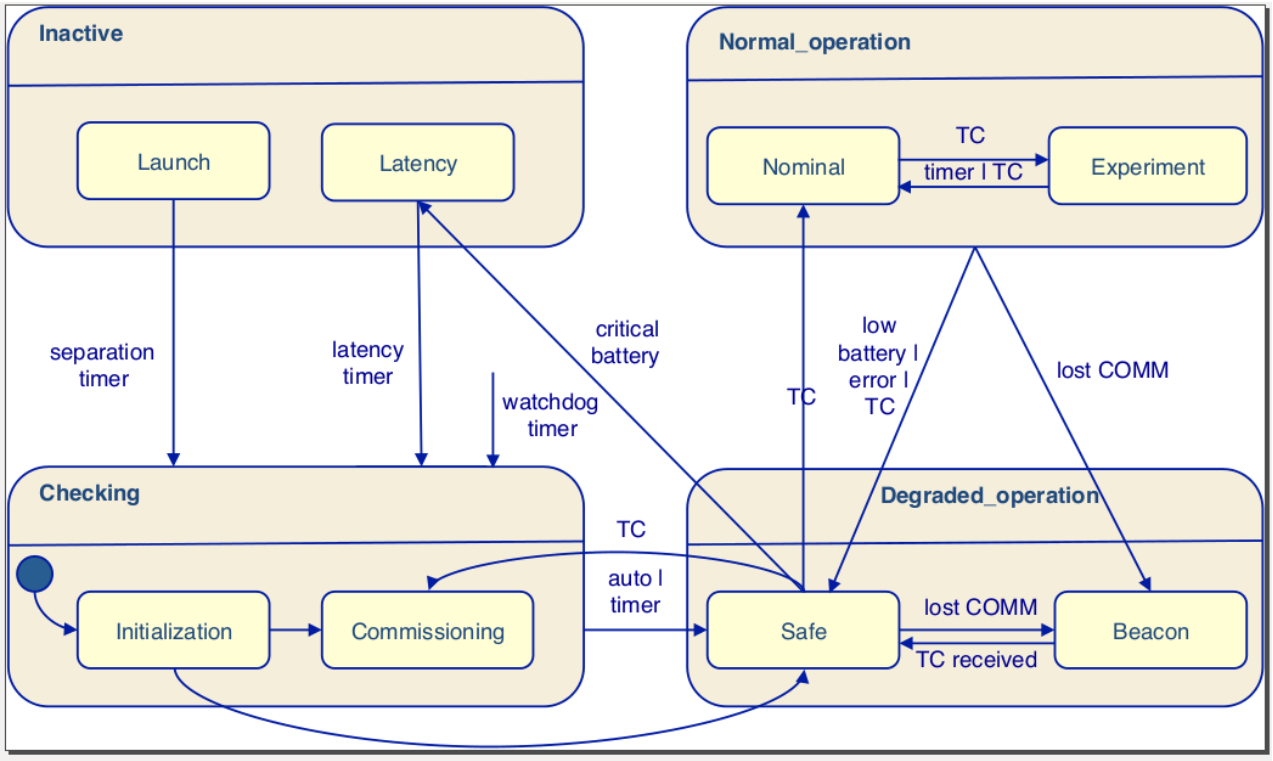

Figura 12 – Diagarama de estados de operación del satélite UPMSat-2.

Cambio de valor de parámetro de configuración

Subservicio que permite el cambio de valor de un parámetro de configuración software distinto del estado de operación del satélite. El identificador de este subservicio es el nº 2. A continuación se detalla la propuesta para el campo *Application Data* de los mensajes de telecomando asociados a este subservicio.

- ParameterID (enumerado): Identificador del parámetro de configuración a modificar. Los posibles dependen del documento de especificación software.
- Value (dependiente de ParameterID): Nuevo valor para el parámetro de configuración.

5 Modelado de interfaces

La segunda fase propuesta por la metodología ASSERT-TASTE es el modelado de las interfaces del sistema. En esta fase se declaran las diferentes funciones del sistema. En el modelado de interfaces de ASSERT/TASTE, las funciones son entidades software que interaccionan entre ellas ofreciendo diferentes servicios, asemejándose al concepto de componente de otras metodologías MDE. Estas funciones por tanto se caracterizan por sus interfaces, así como el lenguaje de programación o de modelado en el que se implementará. Las interfaces pueden ser provistas, si la función las implementa, o requeridas, si representan la llamada a un procedimiento implementado por otra función. Por lo tanto, las interfaces provistas son los puntos de activación de las diferentes tareas. Así, estas pueden ser:

- Cíclicas: Son funciones activadas por el vencimiento de un temporizador con un tiempo de expiración constante conocido como periodo.
- Esporádicas: Se activan al ser llamadas por otras funciones. Si bien, a diferencia de las cíclicas, no se activan bajo un patrón de tiempo constante, sí es posible asegurar un tiempo mínimo entre dos llamadas sucesivas, lo cual permite el análisis temporal del sistema.
- Protegidas: Se activan ante la llamada de otra función y sobre ellas no se puede ofrecer ninguna suposición sobre su esquema de activación. Este tipo de interfaz debe ejecutarse en exclusión mutua. Esto quiere decir que si existe alguna otra tarea de la función en ejecución, la llamada a la tarea protegida deberá esperar a que esta concluya para comenzar a ejecutar. Una vez esto suceda, cualquier activación de tarea de la función deberá esperar a que la tarea protegida concluya su ejecución.
- No protegidas: Se activan ante la llamada de otra función y sobre ellas no se puede ofrecer ninguna suposición sobre su esquema de activación. Además, no presentan ninguna restricción sobre la ejecución concurrente de otras tareas dentro de la función que las implementa.

Todos los tipos de tareas pueden caracterizarse además por su plazo de ejecución o *deadline* y su peor tiempo de ejecución (o WCET por sus siglas en inglés, Worst Case Execution Time). Esto permite que posteriormente pueda analizarse la planificabilidad del sistema, es decir, si es posible asegurar el cumplimiento de los requisitos temporales de todas las funciones.

Adicionalmente, dado que las interfaces provistas representan las llamadas a las funciones por parte de otras funciones, estas interfaces pueden tener parámetros. Estos parámetros pueden ser tanto de entrada como de salida y los tipos de los mismos pueden ser solamente aquellos definidos previamente en la fase de modelado de datos.

Finalmente, las funciones también pueden presentar lo que en la herramienta se conocen como *Functional States*, que son variables globales accesibles por todas las tareas de la función.

Para facilitar la edición del modelo, y hacer más sencilla su interpretación, la herramienta de edición gráfica permite la agrupación de funciones en contenedores (*containers* en la herramienta), que en este caso no tienen ninguna implicación funcional, son meras ayudas visuales.

Así, en la Figura 13 podemos ver el modelo de interfaces del satélite UPMSat-2. En ella, cada contenedor (bloques grises) representa cada uno de los diferentes subsistemas del satélite presentados en la figura. Dentro de cada contenedor pueden existir una o varias funciones (bloques amarillos). Estos bloques presentan flechas azules que representan las interfaces de la función, salientes en caso de las interfaces requeridas y entrantes en el caso de las interfaces provistas.

En las posteriores subsecciones se detalla cada subsistema, incluyendo una descripción de las funciones que lo componen y sus interfaces.

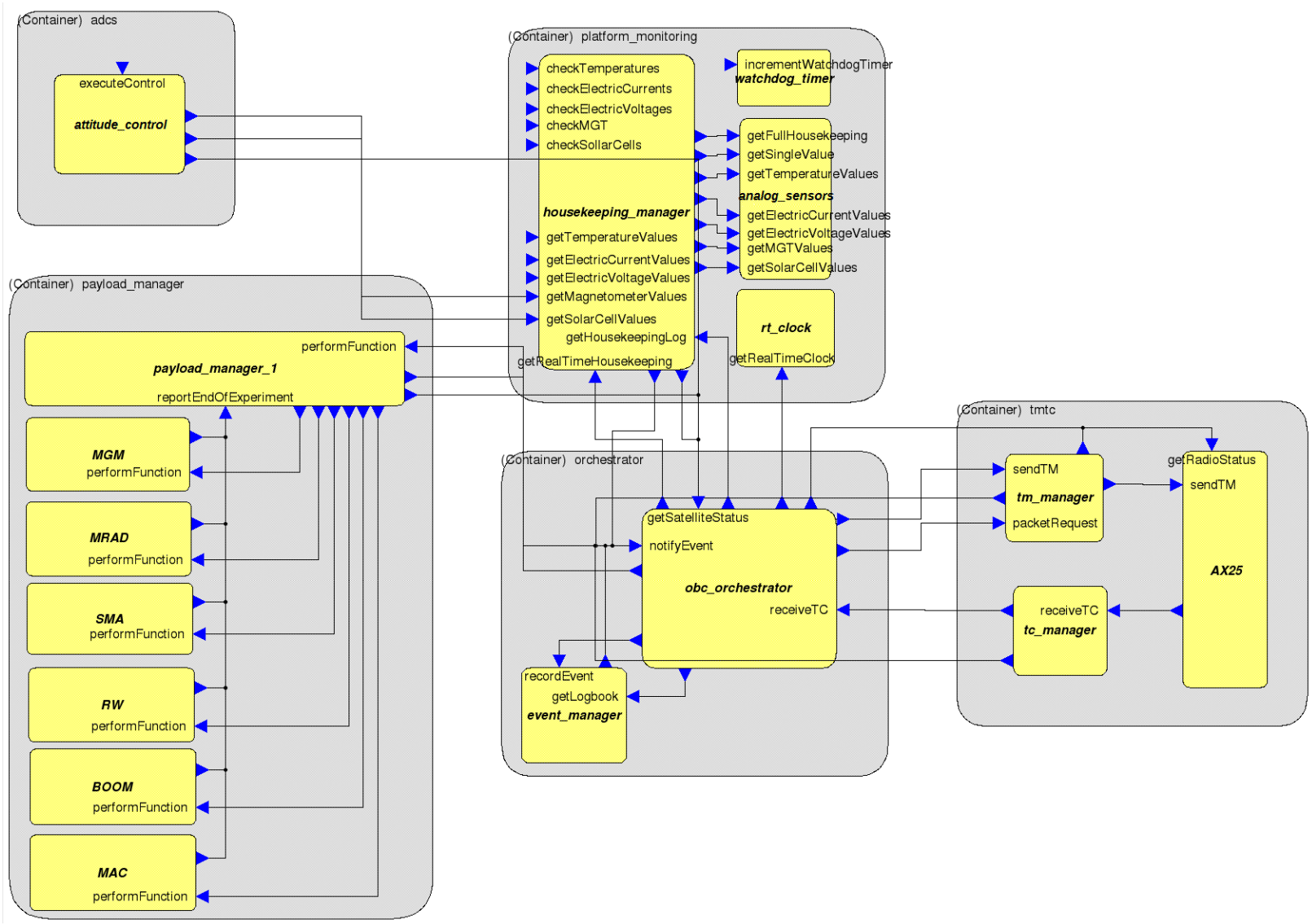

Figura 13 – Modelo de Interfaces del software de alto nivel del satélite UPMSat-2

5.1 Orchestrator

El sistema Orchestrator representa la entidad central y de control del satélite. A pesar de que en la medida de lo posible se delega a cada sistema la mayor cantidad posible de labores, resulta altamente conveniente tener una entidad que controle el correcto funcionamiento del resto, así como gestione algunos recursos compartidos. Entre ellos se encuentra el manejo del estado de operación del satélite, el cual determina el comportamiento del conjunto del sistema, y por tanto su manipulación ha de ser lo más estable y cuidadosa posible, evitando problemas de concurrencia.

Así, como se puede observar en la Figura 14, el sistema cuenta con dos funciones software: una central, y que servirá como interfaz al resto del sistema llamada *obc_orchestrator* y una auxiliar, *event_manager*, cuya función será la de administrar el registro de los eventos que vayan sucediéndose en el satélite.

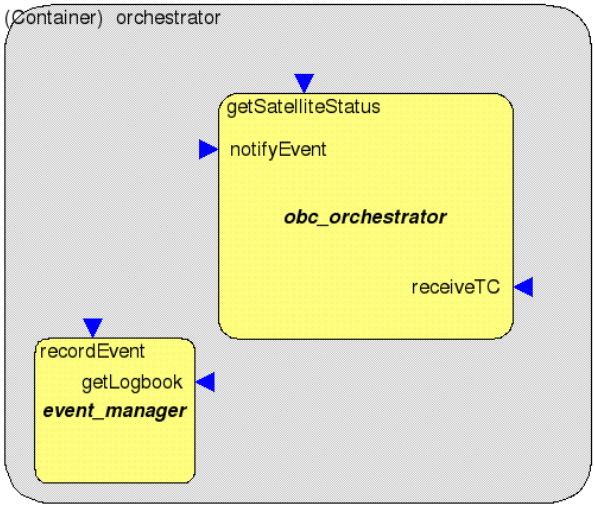

Figura 14 – Funciones e Interfaces Provistas por el sistema Orchestrator.

Obc-orchestrator

Esta función se encarga de organizar y orquestar (de ahí su nombre) el comportamiento del conjunto del satélite. Para ello provee tres interfaces al resto del sistema:

Nombre Interfaz	Tipo	Parámetros		
receiveTC	protected	receivedTC	TelecommandPacketType	IN
notifyEvent	protected	eventToRecord	LogbookEventType	IN
		eventSeverity	EventSeverityType	IN
getSatelliteStatus	protected	satelliteStatus	SatelliteStatusType	OUT

Tabla 2 – Interfaces Provistas por obc_orchestrator.

- receiveTC: Recibe los telecomandos directamente del subsistema de comunicaciones, los interpreta y desencadena las operaciones pertinentes.
- notifyEvent: Recibe la notificación de los eventos detectados tanto por el resto de sistemas, como de event_manager.
- getSatelliteStatus: Devuelve el estado actual del satélite. De este modo, se puede ofrecer un estado consistente del satélite, y se evita una errónea manipulación del mismo por parte del resto de sistemas.

Además, obc_orchestrator tiene variables internas, conocidas en TASTE como *Functionnal States*. Concretamente tiene una, el ya mencionado estado del satélite:

Nombre Variable	Tipo
satelliteStatus	SatelliteStatusType

Tabla 3 – Variables internas de obc_orchestrator.

- satelliteStatus: Representa el estado de operación del satélite. Solo debe ser consultado por medio de su interfaz y modificado por el obc_orchestrator como respuesta a telecomandos recibidos desde tierra o a eventos registrados a bordo.

Event_manager

Esta función se encarga de administrar los eventos registrados en el satélite. Para ello, almacena diferentes estructuras de datos para cada una de las severidades de eventos que pueden suceder a bordo del satélite, y ofrece dos interfaces para acceder a ellas:

Nombre Interfaz	Tipo	Parámetros		
getLogbook	protected	eventSeverityLogbook	EventSeverityType	IN
		logbook	LogbookBufferType	OUT
recordEvent	protected	eventToRecord	LogbookEventType	IN
		eventSeverity	EventSeverityType	IN

Tabla 4 – Interfaces Provistas por event_manager.

- getLogbook: Permite obtener el registro de eventos de una determinada severidad.
- recordEvent: Permite almacenar en el registro un determinado evento, dada su severidad. Debe determinarse el comportamiento en el caso en que se llene el registro de un determinado tipo. Las posibles opciones en ese caso es sobreescribir las eventos pasados no enviados a tierra más antiguos, o descartar los nuevos eventos entrantes.

  Además de lo presentado, el Event_manager deberá manejar cuatro registros de eventos o *Logbooks*, uno por cada gradación de los eventos de a bordo (*normal, low severity, medium severity* y *high severity*).

5.2 Platform Monitoring

Uno de los principales sistemas dentro de un satélite es dedicado al control del estado de la plataforma. En este tipo de ingenios, como es un satélite, se debe tener especial cuidado con las condiciones en las que opera el mismo. Debido al ambiente hostil en el que opera, así como la imposibilidad de realizar acción física alguna sobre él una vez puesto en órbita, las condiciones han de tenerse lo más bajo control posible. Así, periódicamente, se ha de comprobar que las diferentes partes del satélite se encuentran operando a la temperatura adecuada, que la intensidad y la tensión eléctrica que reciben los subsistemas es la adecuada, que las baterías tengan la carga suficiente para mantener el estado de operación, o que el satélite tenga la orientación adecuada, a fin de poder cargar las baterías por medio de los paneles solares y que la antena esté correctamente apuntada.

Además, no solo es de interés para el software de a bordo conocer todos estos parámetros. En general en cualquier misión, y más especialmente en una académica como es la del UPMSat-2, resulta de gran interés poder estudiar estos valores en tierra. Cabe recordar que gran parte de los experimentos están relacionados con estas medidas (control térmico, control de actitud, células y paneles solares, etc.) Por ello, se ha de conservar un registro de los valores muestreados (no necesariamente todos, como ya se explicó previamente) que permita el envío de los mismos una vez se disponga de cobertura con la estación de tierra.

Por todo lo expuesto anteriormente este sistema es uno de los más complejos del satélite, como se puede observar en la Figura 15:

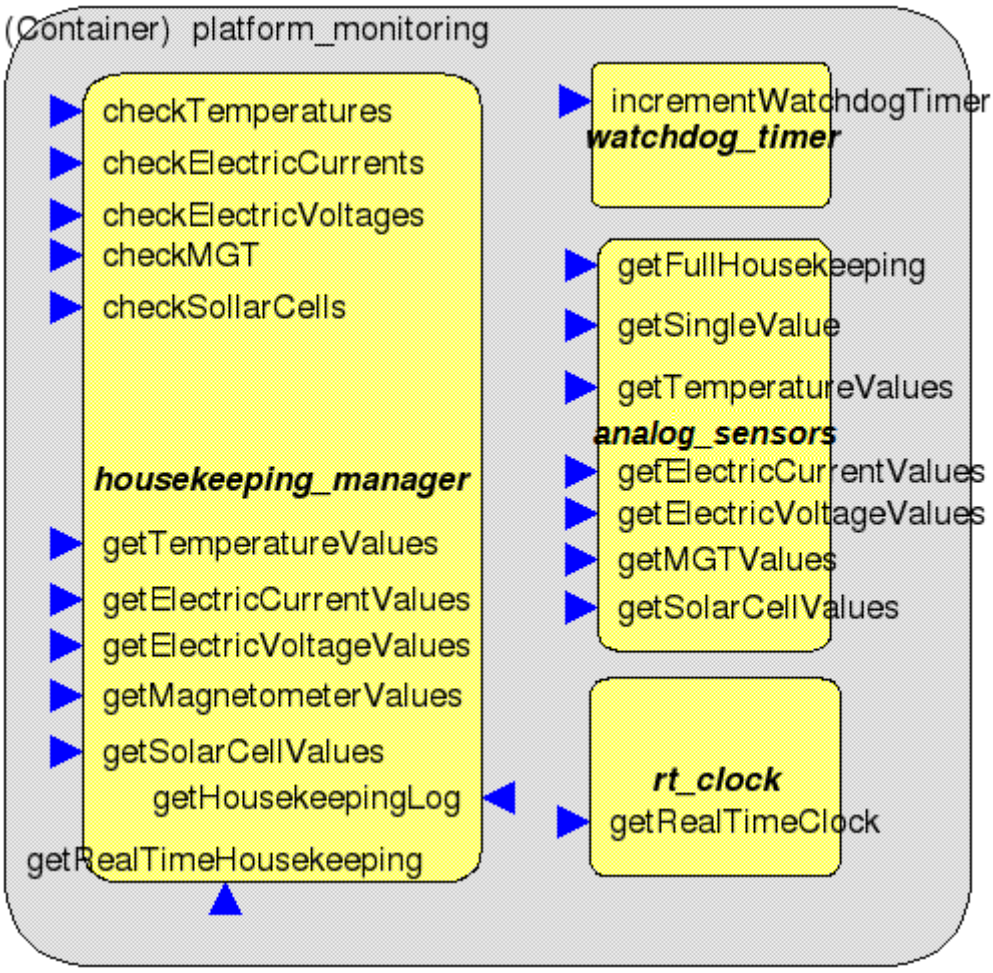

Figura 15 – Funciones e Interfaces Provistas por el sistema Platform Monitoring.

Housekeeping_manager

Esta función es la principal del sistema y ofrece, con la salvedad del reloj de tiempo real, la interfaz del sistema hacia el resto del satélite. Por tanto, presenta un elevado número de interfaces provistas. Además, dado que la monitorización de los diferentes datos es una actividad periódica, que no depende de otras funciones (únicamente del estado de operación del satélite), existe un número de interfaces provistas de activación cíclica. La descripción de estas interfaces es la siguiente:

Nombre Interfaz	Tipo	Parámetros		
getHousekeepingLog	protected	housekeepingRequest Parameters	HousekeepingLog RequestType	IN
		housekeepingReport	HousekeepingParameter ReportAppDataType	OUT
getRealTime Housekeeping	protected	housekeepingRequest Parameters	HousekeepingRealTime RquestType	IN
		housekeepingReport	HousekeepingParameter ReportAppDataType	OUT
getSolarCellValues	protected	solarCellValues	SolarCellConversion	OUT

			ValuesType	
getMagnetometer Values	protected	magnetometerAxis	MagnetometerAxisType	OUT
getElectric VoltageValues	protected	electricVoltageValues	ElectricVoltage ConversionValuesType	OUT
getElectric CurrentValues	protected	electricCurrentValues	ElectricCurrent ConversionValuesType	OUT
getTemperature Values	protected	temperatureValues	TemperatureConversion ValuesType	OUT
checkSollarCells	cyclic	Period	10 seconds[9]	-
checkMGT	cyclic	Period	5 seconds[9]	-
checkElectric Voltages	cyclic	Period	20 seconds[9]	-
checkElectric Currents	cyclic	Period	20 seconds[9]	-
CheckTemperatures	cyclic	Period	60 seconds[9]	-

Tabla 5 – Interfaces Provistas por housekeeping_manager.

- getHousekeepingLog: Permite recuperar la información de *housekeeping* de todos o cada uno de los tipos definidos (temperaturas, tensiones, intensidades, medidas de los magnetómetros, medidas de las células solares).
- getRealTimeHousekeeping: Permite solicitar la adquisición y reporte de datos de *housekeeping* en tiempo real de todos o alguno de los tipos definidos previamente.
- getSolarCellValues: Permite obtener la última medida de los valores registrados por las células solares.
- getMagnetometerValues: Permite obtener la última medida de los valores registrados por los magnetómetros.
- getElectricVoltageValues[10]: Permite obtener la última medida de los valores de intensidad registrados.
- getElectricCurrentValues[10]: Permite obtener la última medida de los valores de tensión eléctrica registrados
- getTemperatureValues[10]: Permite obtener la última medida de los valores de temperatura registrados.
- checkSollarCells: tarea periódica que realiza la adquisición de datos de las células solares, con un periodo de 10 segundos en modo nominal. Para el resto de modos aún no se encuentra definido. Se ha de definir específicamente el comportamiento cuando el registro asociado se encuentre completo.

[9] La periodicidad muestreada pertenece al modo de operación nominal del satélite. Para el resto de modos no está especificada por el momento.

[10] A pesar de que en el diseño actual esta interfaz no es requerida por ningún otro sistema, se ofrece por homogeneidad con el resto del sistema, así como para su posible uso y compatibilidad en futuros diseños.

- checkMGT: tarea periódica que realiza la adquisición de datos del campo magnético terrestre detectado por los magnetómetros, con un periodo de 5 segundos en modo nominal. Para el resto de modos aún no se encuentra definido. Se ha de definir específicamente el comportamiento cuando el registro asociado se encuentre completo.
- checkElectricVoltages: tarea periódica que realiza la adquisición de datos del voltaje en diferentes partes del circuito del satélite, con un periodo de 20 segundos en modo nominal. Para el resto de modos aún no se encuentra definido. Se ha de definir específicamente el comportamiento cuando el registro asociado se encuentre completo.
- checkElectricCurrents: tarea periódica que realiza la adquisición de datos de la tensión en diferentes partes del circuito del satélite, con un periodo de 20 segundos en modo nominal. Para el resto de modos aún no se encuentra definido. Se ha de definir específicamente el comportamiento cuando el registro asociado se encuentre completo.
- checkTemperatures: tarea periódica que realiza la adquisición de datos de la temperatura de diferentes partes de la estructura, así como de componentes del satélite, con un periodo de 60 segundos en modo nominal. Para el resto de modos aún no se encuentra definido. Se ha de definir específicamente el comportamiento cuando el registro asociado se encuentre completo.

Asimismo, cuenta con cinco estructuras de datos para, como se adelantó, cumplir con su función de gestión de los datos de *housekeeping* hasta su envío a tierra:

- temperatureLogbook: Registro de los valores de temperatura registrados en el satélite.
- electricCurrentLogbook: Registro de los valores de tensión registrados en el satélite.
- electricVoltageLogbook: Registro de los valores de intensidad registrados en el satélite.
- mGTLogbook: Registro de los valores de campo magnético registrados en el satélite.
- sollarCellLogbook: Registro de los valores de posición del sol registrados en el satélite.

Analog_sensors

Una pieza clave en la monitorización del conjunto del satélite son los sensores. Estos dispositivos son capaces de detectar una determinada magnitud física y transformar la medida en una variable eléctrica (generalmente el voltaje). Así, mediante estos sensores se mide la temperatura, la intensidad y voltaje de la corriente en los buses de potencia, el campo magnético terrestre y la exposición solar de las diferentes partes del satélite. Para que estas medidas puedan ser interpretadas por el computador de a bordo (un sistema digital), la salida de los sensores (un determinado voltaje, magnitud analógica) debe ser convertida a un valor digital.

El propósito de la función analog_sensors es la de servir de interfaz entre el resto del sistema de *housekeeping* y el citado conversor (o conversores, dependiendo del número de sensores). De este modo, mediante las interfaces provistas, esta función permite al housekeeping_manager obtener los valores de los diferentes sensores de forma sencilla, ocultando la complejidad del driver del citado conversor.

Nombre Interfaz	Tipo	Parámetros		
getFullHousekeeping	protected	fullHousekeeping	FullADCConversion ValueTper	OUT
getSingleValue	protected	aDCConnectedDevice	ADCConected DeviceType	IN
		aDCConversionValue	ADCConversion ValueType	OUT
getTemperature Values	protected	temperatureValues	TemperatureConversion ValuesType	OUT
getElectric CurrentValues	protected	electricCurrentValues	ElectricCurrent ConversionValuesType	OUT
getElectric VoltageValues	protected	electricVoltageValues	ElectricVoltage ConversionValuesType	OUT
getMGTValues	protected	magnetometerAxis	MGTConversionValuesT ype	OUT
getSolarCellValues	protected	solarCellValues	SolarCellConversion ValuesType	OUT

Tabla 6 – Interfaces Provistas por analog_sensors.

- getFullHousekeeping: Devuelve una estructura de datos con una lectura en tiempo real de todos los sensores del satélite.
- getSingleValue: Devuelve una lectura en tiempo real de uno de los sensores del satélite.
- getTemperatureValues: Devuelve una estructura de datos con las lecturas en tiempo real de los 23 sensores de temperatura del satélite.
- getElectricCurrentValues: Devuelve un conjunto con las lecturas en tiempo real de los 2 sensores de tensión del satélite.
- getElectricVoltageValues: Devuelve un conjunto con las lecturas en tiempo real de los 7 sensores de intensidad del satélite.
- getMagnetometerValues: Devuelve un conjunto con las lecturas en tiempo real de los 3 magnetómetros del satélite.
- getSolarCellValues: Devuelve un conjunto con las lecturas en tiempo real de las 6 células solares del satélite.

Esta función, intermediaria entre el gestor del housekeeping y el driver del conversor, no tiene variables propias, o *Functional States*.

Watchdog_timer

Un importante riesgo en los sistemas empotrados es que el software o el hardware del sistema fallen, de manera que el controlador quede fuera de servicio. Para evitarlo, estos sistemas se diseñan e implementan siguiendo metodologías específicas, y haciendo uso de materiales con especificaciones (en lo concerniente a disponibilidad y tiempo entre fallos) muy por encima de los sistemas informáticos convencionales, además de un concienzudo proceso de validación. A pesar de ello, ningún sistema puede considerarse cien por cien libre de fallos, y puede suceder que ocurra un error por el cual el sistema se bloquee. Como parece evidente, este tipo de sistemas no pueden ser reiniciados de forma sencilla por el usuario, como sería el caso de un sistema convencional. Para ello, estos sistemas cuentan con un temporizador de guardia, o *watchdog timer* en inglés que se encarga de reiniciar por un impulso eléctrico el sistema en ese caso. Su implementación es la de un temporizador convencional conectado al sistema de potencia del computador de a bordo, de manera que cuando el temporizador llega a cero, interrumpe momentáneamente la corriente del computador, reiniciando así el sistema. De esta manera, si la tarea periódica responsable de incrementar el temporizador no puede ejecutar y por tanto incrementar el valor, se interpreta que el sistema se ha bloqueado, y tras un tiempo determinado por el pertinente parámetro de configuración, el temporizador salta y resetea el computador.

Por lo tanto, esta función solo cuenta con una función periódica encargada de llevar a cabo esta tarea:

Nombre Interfaz	Tipo	Parámetros	
incrementWatchdogTimer	cyclic	Period	TBD

Tabla 7 – Interfaces Provistas por watchdog_timer.

- incrementWatchdogTimer: Tarea periódica que se encarga de aumentar el valor del *watchdog timer* cada un determinado periodo de tiempo (por definir) en un determinado número de *ticks* de reloj (por definir).

Para llevar a cabo esta tarea, no es necesario que la función disponga de ninguna variable interna.

Rt_clock

Como en la mayoría de sistemas de navegación, resulta de interés el llevar un registro de los eventos que van sucediendo durante su operación, como ya se ha mencionado. Para ello, se requiere una fuente que provea de un tiempo común al conjunto del sistema de navegación para su uso homogéneo. En el caso de los satélites, este reloj se conoce como Reloj de Misión. Este reloj está diseñado para ser arrancado en el momento en el que el computador de a bordo se ponga en marcha por primera vez, y no se detenga a pesar de que el computador se reinicie o bloquee.

Nombre Interfaz	Tipo	Parámetros		
getRealTimeClock	protected	realTimeClock	RealTimeClockType	OUT

Tabla 8 – Interfaces Provistas por rt_clock.

5.3 Payload Manager

El objetivo del satélite UPMSat-2 es ser un demostrador tecnológico en órbita, tanto del funcionamiento y tecnología del satélite en si como un demostrador de diferentes dispositivos y técnicas. Para la puesta en marcha y análisis de algunos de ellos es necesario el llevar a cabo diversas operaciones, de lo que se encarga el gestor de experimentos. Este interpreta las órdenes recibidas del Orchestrator (por medio de telecomandos) cuando el satélite se encuentra en estado de operación experimental, y las convierte en órdenes directas al gestor de cada uno de los experimentos.

Para ello, se cuenta con el mencionado gestor y una función gestora de experimento por cada uno que lo requiera, como se puede ver en la Figura 16.

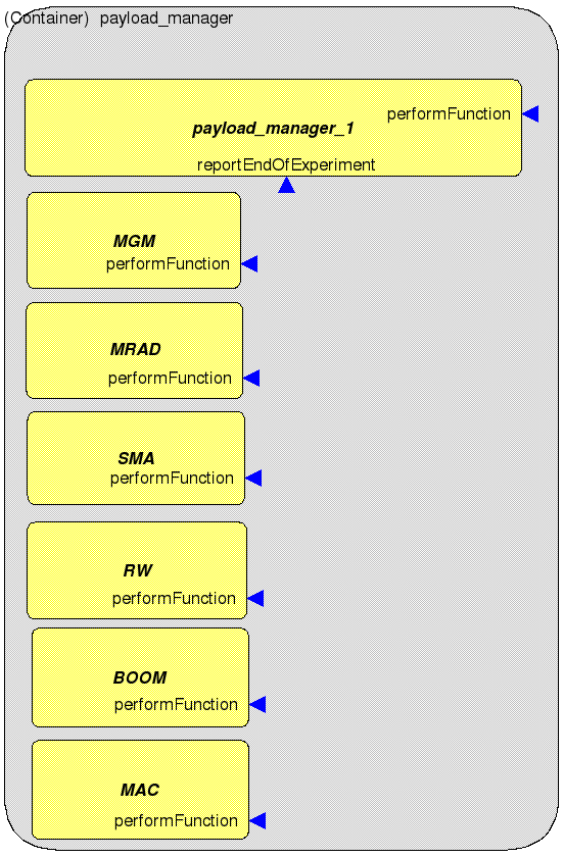

Figura 16 – Funciones e Interfaces Provistas por el sistema Payload Manager.

Payload_manager

Función que sirve de interfaz al Orchestrator y que gestiona la puesta en marcha y detención de cada uno de los experimentos. Su nombre en el modelo y en general dentro de todo el entorno TASTE es payload_manager_1, debido a que payload_manager es una palabra restringida dentro de la cadena de herramientas. La función ofrece dos interfaces:

Nombre Interfaz	Tipo	Parámetros		
performFunction	unprotected	performFunctionValues	FunctionManagement AppDataType	IN
reportEndOf Experiment	unprotected	experimentReported Values	EventReporting AppDataType	IN

Tabla 9 – Interfaces Provistas por payload_manager_1.

- performFunction: Interfaz para la recepción de órdenes desde el orquestador para poner en marcha y detener los diferentes experimentos de a bordo. Estas órdenes serán interpretadas y pasadas al gestor de cada experimento.
- reportEndOfExperiment: Permite que los experimentos reporten su consecución, ya sea satisfactoria o no, para su posterior notificación al orquestador.

Gestores de experimento

Cada uno de los diferentes experimentos contará con gestor que reciba las órdenes pertienentes y conozca y ejecute la lógica requerida para cada experimento. Este gestor será también el encargado de recopilar los datos resultado del experimento y notificarlo al gestor de experimentos. Cada una de las funciones presenta la siguiente interfaz:

Nombre Interfaz	Tipo	Parámetros		
performFunction	unprotected	operationToPerform	ExperimentOperation Type	IN

Tabla 10 – Interfaces Provistas por cada gestor de experimento.

Además, cada experimento podrá encontrarse en un estado concreto, o Functional State. Además del aquí definido, cada experimento podrá definir otros según su propia lógica de funcionamiento.

Nombre Variable	Tipo
experimentStatus	ExperimentStatusType

Tabla 11 – Variables internas de cada gestor de experimento.

5.4 Attitude Determination and Control System

La actitud de un satélite es su orientación respecto al centro de la Tierra. En el momento del lanzamiento, por cuestiones técnicas, el satélite adquiere una gran velocidad de giro al ser separado del vehículo lanzador. Una vez prudentemente alejado del mismo, deben tomarse las adecuadas medidas para reducir dicha velocidad de giro y orientar el satélite de forma adecuada, de manera que los paneles solares reciban la máxima exposición solar posible, y la antena se encuentre correctamente apuntada para permitir las comunicaciones con la estación terrestre.

Una vez en una órbita estable, diversos factores como el propio campo magnético de la Tierra, la radiación solar o la atmósfera residual alteran esta configuración, de manera que periódicamente se debe recalcular la orientación del satélite y operar los magnetopares para reorientar el satélite. De todo ello se encarga el Sistema de Determiandción y Control de Actitud, o ADCS por sus siglas en inglés (Attitude Determination and Control System).

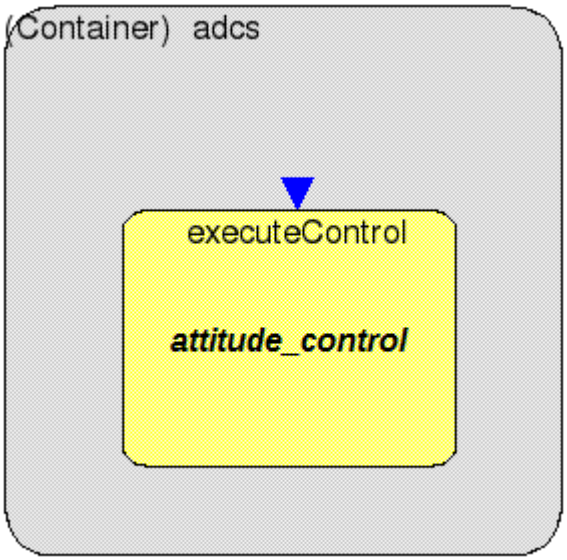

Figura 17 – Funciones e Interfaces Provistas por el sistema Attitude Determination and Control System.

Attitude_control

Una única función se encarga de todo el proceso, interpretar los valores recibidos por los magnetómetros y calcular la actuación necesaria a llevar a cabo mediante los magnetopares. Esto se realiza periódicamente (periodo por definir), por lo cual su interfaz provista es una tarea cíclica:

Nombre Interfaz	Tipo	Parámetros		
executeControl	cyclic	Period	TBD	-

Tabla 12 – Interfaces Provistas attitude_control.

- executeControl: Tarea de activación periódica que requiere de las medidas de los magnetómetros para calcular la orientación del satélite y poder calcular la posible actuación de los magnetopares para, interactuando con el campo magnético de la tierra, reorientar el satélite debidamente.

5.5 Telemetry and Telecommand System

Un satélite, por si mismo, sin comunicación con tierra no es más que cualquiera de las masas que orbitan en torno a la Tierra. Por ello, desde el primer satélite puesto en órbita, el Sputnik, todo ellos han contado con sistemas más o menos complejos de comunicación con una estación base en tierra.

En el caso del satélite UPMSat-2, el radioenlace se realiza en banda UHF 400 MHz a 19200 baudios. Para los protocolos de bajo nivel, como ya se ha discutido en la presente memoria, se hace uso del protocolo AX.25.

Para su gestión, se modelan las tres funciones representadas en la Figura 18.

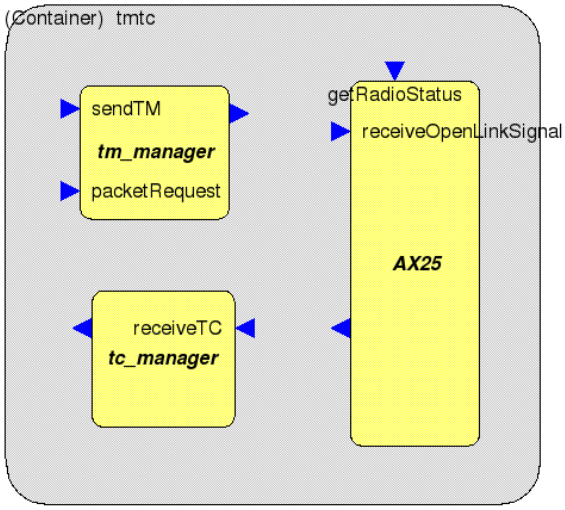

Figura 18 – Funciones e Interfaces Provistas por el sistema Telemetry and Telecomunications System.

Tm_manager

Función encargada de gestionar los mensajes de telemetría listos para ser enviados a tierra. Para ello, ofrece una interfaz al Orchestrator para recibir esos mensajes que posteriormente serán enviados vía el protocolo AX.25. Del mismo modo debe gestionar el reenvío de mensajes perdidos por lo cual debe almacenarlos durante un determinado periodo de tiempo así como numerarlos adecuadamente para ello.

Nombre Interfaz	Tipo	Parámetros		
sendTM	protected	sendingTM	TelemetryPacket Type	IN
packetRequest	protected	packetRequested	PacketRequestedApp DataType	IN

Tabla 13 – Interfaces Provistas por cada tm_manager.

- sendTM: Interfaz que permite al Orchestrator poner en la cola de envío los diferentes mensajes de telemetría que se vayan generando.
- packetRequest: Permite al Orchestrator solicitar al sistema de comunicaciones la retransmisión de un mensaje de telemetría perdido (a petición de la estación de tierra).

Nombre Variable	Tipo
sequenceCount	PHSequenceCountType

Tabla 14 – Variables internas de cada gestor de tm_manager.

- sequenceCount: Valor de secuencia del último mensaje de telemetría numerado. Una vez alcanzado el valor máximo, el siguiente mensaje volverá a reutilizar el número de secuencia 0.

Además, el Tm_manager deberá contar con un registro de los últimos mensajes enviados (almacenados en caso de que sea necesario reenviarlos) y pendientes de enviar (para enviarlos cuando sea posible)

Tc_manager

Del mismo modo que para la telemetría, se requiere una función que se encargue de gestionar los telecomandos dentro del sistema de telecomunicaciones antes de pasarlos al Orchestrator para su interpretación y ejecución. Además de recibir los mensajes de telecomando y almacenarlos hasta que el Orchestrator pueda procesarlos, se encarga de calcular y verificar las sumas de comprobación, de modo que solo los telecomandos correctamente recibidos sean procesados.

Nombre Interfaz	Tipo	Parámetros		
receiveTC	protected	receivedTC	TelecommandPacket Type	IN

Tabla 15 – Interfaces Provistas por cada tc_manager.

- receiveTC: Recibe los telecomandos una vez recibidos por los protocolos de niveles inferiores. Una vez recibidos, su suma de comprobación debe ser verificada, el telecomando almacenado en el buffer adecuado y solicitarse su procesamiento por Orchestrator.

Nombre Variable	Tipo
sequenceCount	PHSequenceCountType

Tabla 16 – Variables internas de cada gestor de tm_manager.

- sequenceCount: Valor de secuencia del último mensaje de telecomando recibido.

Como se ha mencionado, se requiere de un buffer de almacenamiento de los telecomandos recibidos y verificados a la espera de ser procesados por el Orchestrator.

AX.25

Función que representa el conjunto de protocolos de nivel de enlace y físico, que en el caso del satélite son el protocolo AX.25. Esta función tiene contacto directo por tanto con la radio del satélite para el envío y recepción de mensajes por el radioenlace.

Nombre Interfaz	Tipo	Parámetros		
getRadioStatus	unprotected	radioStatus	RadioLinkConnectedType	OUT
sendTM	protected	sendingTM	TelemetryPacketType	IN

Tabla 17 – Interfaces Provistas por cada AX.25.

- getRadioStatus: Permite conocer si se está en cobertura o no, para el envío de mensajes de Telemetría.
- sendTM: Ofrece la interfaz para el envío del siguiente mensaje de telemetría mediante el procesamiento recogido en el estándar AX.25.

Nombre Variable	Tipo
radioLinkConnected	RadioLinkConnectedType

Tabla 18 – Variables internas de cada gestor de AX.25

- radioLinkConnected: Variable que registra el estado actual del radioenlace.

6 Modelado de despliegue

Una vez declaradas los componentes o funciones del sistema, es posible generar un modelo de despliegue de las mismas, es decir, asignar las funciones a la plataforma donde ejecutarán. El resultado de esta fase por tanto es un PSM o modelo específico de plataforma presentado con anterioridad.

En este modelo se declaran las plataformas de ejecución del sistema. En el caso de existir más de una, estas pueden comunicarse mediante buses. No se requiere que estos buses existan físicamente, pues son una abstracción de cualquier canal que permita la comunicación entre diferentes placas (por ejemplo un radioenlace).

En cada plataforma puede existir uno o más procesadores. Estos procesadores a su vez pueden contener una o varias particiones, que alojan las funciones declaradas en el modelo de interfaces. Cada partición tiene como propiedad el sistema operativo o núcleo que ejecuta en ella.

En el caso del satélite UPMSat-2 solo existe un computador de a bordo, monoprocesador, o *monocore*, sobre el que ejecutará una única partición administrada por el núcleo ORK, desarrollado por el grupo. Por tanto, como se puede ver en la Figura 19, el modelo de despliegue en este caso es relativamente sencillo.

Como muestra la figura, no se contempla el software de la estación de tierra. Esto permite que este software se desarrolle con cierta independencia del software de a bordo. A pesar de ello, el modelado de datos realizado en las fases iniciales del diseño permite que ese desarrollo independiente sea en todo momento coherente con el presentado en esta memoria. Esta cualidad es otra de las ventajas del desarrollo basado en modelos, y de los modelos independientes de plataforma.

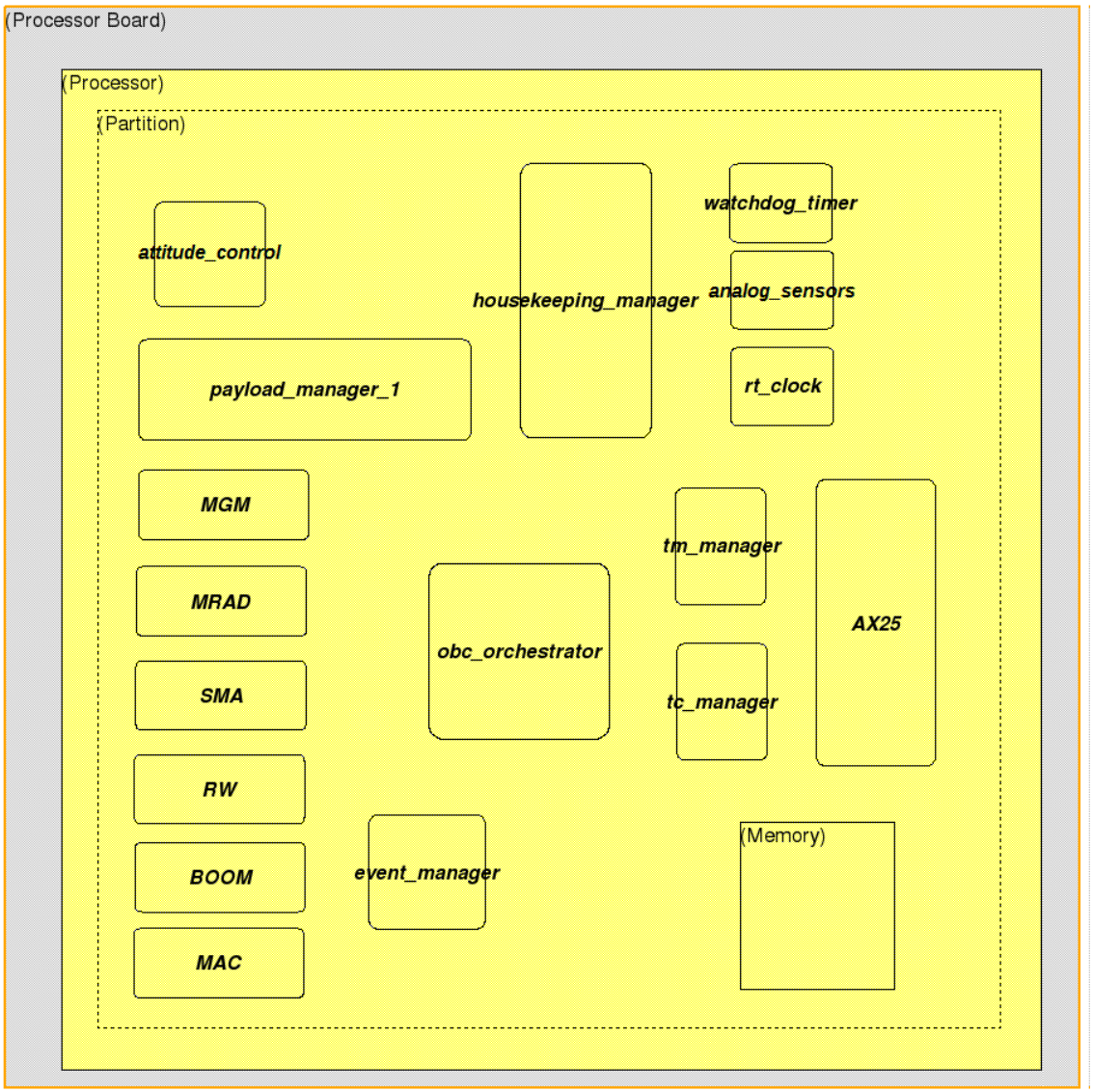

Figura 19 – Modelo de despliegue de las funciones en la plataforma de ejecución.
Al tratarse de un sistema monoprocesador con una única todas las funciones quedan asignadas a la misma partición.

7 Implementación

Una vez se han declarado las funciones en la fase correspondiente, el entorno permite la generación tanto de esqueletos de código que implementen la información contenida en los modelos de interfaz y de datos, como los mandatos para compilar el código generado para obtener un ejecutable por partición. Por tanto, antes de ejecutar los scripts de compilación, lógicamente, debe haberse llevado a cabo la fase de modelado de despliegue. No es necesario, sin embargo, a la hora de generar los esqueletos, ya que la cadena de compilación está diseñada de manera que la asignación a particiones se haga en tiempo de compilación.

La implementación del software del satélite se llevará a cabo en el lenguaje de programación Ada. Este lenguaje, como se adelantó, es un lenguaje profusamente empleado en la programación de sistemas de tiempo real, dada las facilidades que ofrece la especificación del perfil de Ravenscar. Además permite la implementación de núcleos ligeros, como ORK, sobre el cual ejecutará el código de aplicación, por lo que parece intuitivo el hacer uso de este lenguaje.

Para demostrar la capacidad de la herramienta para generar un sistema ejecutable se ha realizado la implementación de un subconjunto de los servicios del satélite.

7.1 Evaluación de la generación de ejecutables en TASTE

Como se adelantó, TASTE ofrece diferentes facilidades para la implementación final del sistema. Además de diferentes lenguajes de programación disponibles, TASTE ofrece una cadena de herramientas enfocada a preservar las características del sistema a lo largo de todas las fases del desarrollo del mismo.

Una de ellas, de gran utilidad, es la generación automática de código. A partir de la información provista en los diferentes modelos, TASTE genera los esqueletos de código para cada una de las funciones descritas en el *Interface View*, encargándose de proveer los mecanismos necesarios para la intercomunicación de las diferentes funciones, independientemente del lenguaje en el que vayan a ser codificadas.

Además, TASTE se encarga de generar para cada lenguaje las estructuras de datos correspondientes a cada una de las modeladas en el *Data View*, de manera que pueden ser fácilmente utilizadas por el desarrollador. Al igual que con las interfaces, TASTE se encarga de ocultar la complejidad que pueda existir a la hora de hacer uso de los mismos, incluso pasando parámetros a procedimientos implementados en otros lenguajes de programación.

Por tanto, se ha hecho uso de la funcionalidad de generación de esqueletos para comenzar con la implementación, generando TASTE un directorio por cada función declara en el Interface View, conteniendo tanto los ficheros de código fuente a

completar por el desarrollador como los mencionados *wrappers* para la intercomunicación entre funciones y tipos de datos.

Figura 20 – Contenido del directorio principal del proyecto tras la generación automática de código.

Además, como muestra la Figura 20, se genera un archivo con los mandatos para la compilación del sistema completo. Esta compilación es relativamente compleja, al incluir diferentes lenguajes para un mismo ejecutable, además de requerirse diversas comprobaciones sobre el comportamiento de código, como las mencionadas del perfil de *Ravenscar*.

Así, se ha realizado la implementación de un pequeño subconjunto del procesado de telecomando y envío de telemetría. Concretamente, ante la recepción de un telecomando de petición de datos de *housekeeping* de temperatura, el sistema simplemente toma el valor del reloj de misión y devuelve "a tierra" un mensaje de telemetría de temperatura, con la estructura e identificadores adecuados para este tipo de mensajes, pero incluyendo únicamente datos en el campo *time* (el resto con valor 0). Para poder realizar esta sencilla implementación y demostrar la capacidad del sistema se ha hecho uso, como se mostrará en la siguiente sección, de una interesante funcionalidad de TASTE: la generación de interfaces gráficas para el uso de interfaces provistas y requeridas de forma sencilla.

8 Validación

Se ha realizado un proceso de validación de la implementación realizada consistente en la simulación de un sistema de telecomunicaciones para hacer llegar telecomandos al satélite y analizar su comportamiento.

Para ello se ha hecho uso de una de las funcionalidades del entorno TASTE: generar interfaces gráficas a partir de funciones declaradas en el modelo de interfaces que permiten la edición manual de los parámetros de las interfaces requeridas y muestra los valores recibidos por las interfaces provistas.

De este modo, como se puede ver en la Figura 21, se ha realizado el envío de un telecomando del servicio Housekeeping Log Request, definido en la sección 4.3.2, obteniéndose la respuesta adelantada en el capítulo anterior, un mensaje de telemetría del servicio Housekeeping Parameter Report incluyendo el valor del reloj del sistema.

Mediante este método, se puede realizar una comprobación y validación de la implementación de forma bastante sencilla, generando conjuntos de test adecuados. En el caso concreto del proyecto, esta herramienta será de gran utilidad en el trabajo futuro de la implementación completa del sistema.

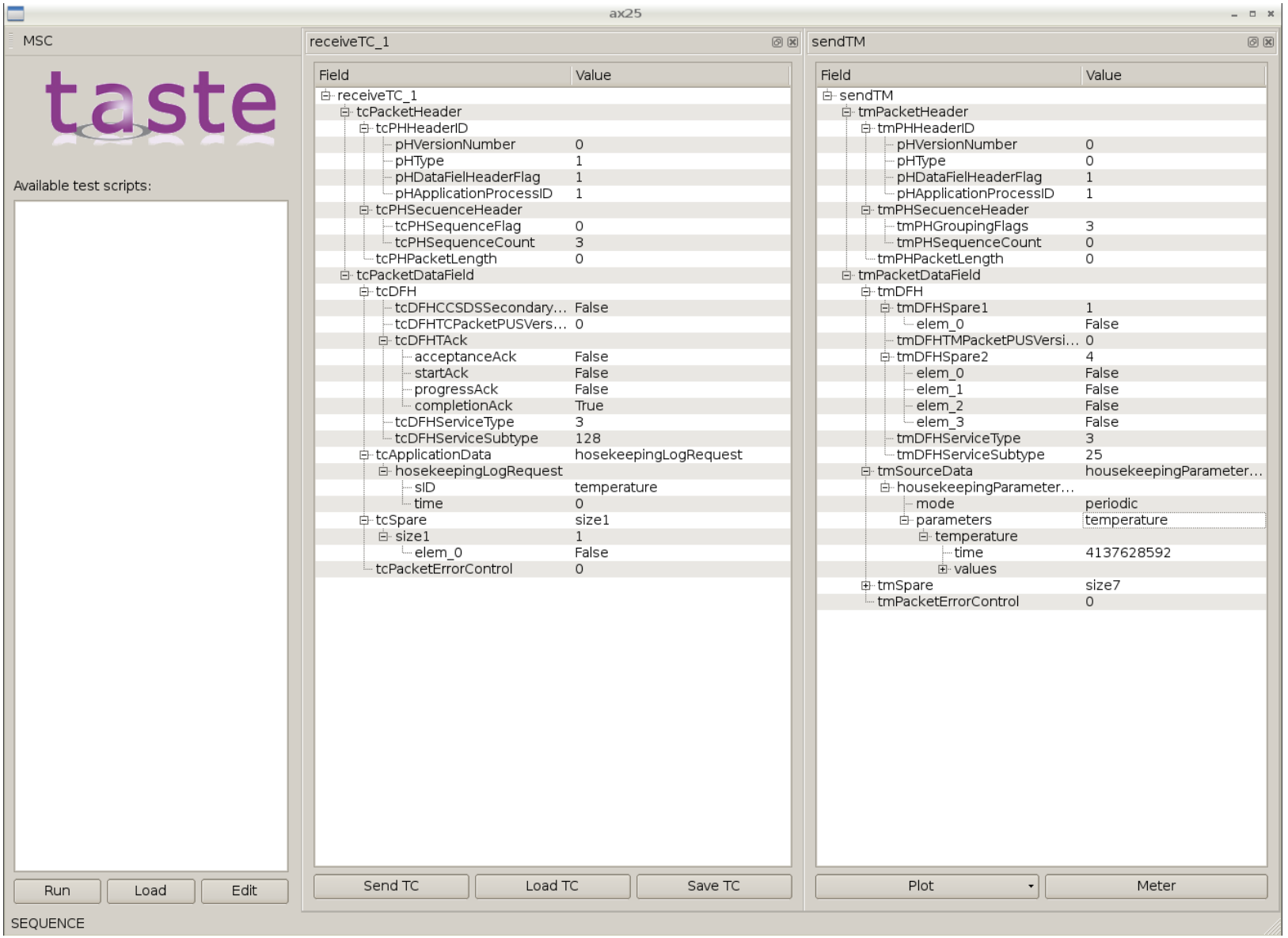

Figura 21 – Herramienta de validación del sistema.

9 Conclusiones

El proyecto UPMSat-2 está suponiendo una gran oportunidad para los diferentes grupos de investigación de la Universidad Politénica de Madrid. En el ámbito del mismo, se están desarrollando múltiples experimentos y poniendo en práctica metodologías y herramientas punteras en el sector aeroespacial. Es el caso de la metodología ASSERT y su conjunto de herramientas TASTE, desarrollado por la Agencia Espacia Europea en diversos proyectos con participación del grupo STRAST y del autor. La satisfactoria consecución del presente trabajo supone una demostración de la utilidad y viabilidad de ambas.

Así mismo, el desarrollo de este trabajo ha supuesto un importante avance en el ámbito del proyecto UPMSat-2. A lo largo del mismo se ha realizado una detallada especificación tanto de la arquitectura software del sistema de a bordo como del sistema de telecomunicaciones y sus servicios.

En concreto, cabe destacar las aportaciones más relevantes del mismo:

- El documento de especificación de requisitos se ha analizado en profundidad- Como consecuencia, se han detectado y corregido diversas inconsistencias o aspectos no especificados.
- Se ha analizado las características del radioenlace del satélite UPMSat-2, estudiado diferentes protocolos aplicables al sistema y seleccionado los más adecuados al proyecto.
- Se ha definido un completo conjunto de servicios que permitirá la operación del satélite de acuerdo a la especificación de requisitos y los estándares pertinentes.
- Se han definido los tipos, estructuras y codificación de los datos del sistema. De este modo se podrá realizar una implementación del software de tierra compatible. Además, este modelado ha permitido realizar diversas consideraciones sobre la telemetría requerida por cada experimento, así como de la memoria requerida por el computador de a bordo.
- Se ha elaborado un diseño de la arquitectura software del satélite, así como de su despliegue en la plataforma de ejecución.
- Se ha realizado la implementación de un subconjunto del sistema, comprobando la utilidad de las herramientas de generación automática de código y compilación de TASTE.
- Se ha verificado la implementación realizada con las herramientas provistas por TASTE. De este modo se ha completado el proceso de desarrollo del sistema tal y como es concebido en TASTE.

Durante el desarrollo del trabajo presentado se ha tenido que hacer frente a un cierto número de problemas y dificultades. Entre ellas cabe destacar la complejidad

que supone el desarrollo de un proyecto multidisciplinar, con un alto número de grupos de investigación y empresas involucradas. Mediante la adopción de metodologías de desarrollo de extendido uso en la industria aeroespacial se ha podido superar esta dificultad, permitiendo la evolución de la especificación del sistema.

Igualmente, las especiales características del satélite han supuesto un reto a la hora de seleccionar los protocolos adecuados para el sistema de comunicaciones del satélite. El relativamente bajo ancho de banda disponible, así como la baja frecuencia de paso y tiempo de cobertura del satélite han supuesto un reto de diseño tanto a nivel de protocolos como de la definición de servicios. En lo referido a estos últimos, nuevamente la colaboración entre grupos ha permitido superar la falta de especificación de algunas áreas del sistema. Esto, unido a un profundo estudio de los estándares relacionados, ha permitido la elaboración de un detallado conjunto de servicios para la operación del satélite.

Para completar satisfactoriamente el trabajo se ha requerido realizar un profundo estudio de los conceptos y técnicas MDE, y más concretamente de la metodología ASSERT. Fruto de ello es la arquitectura software y los diferentes modelos presentados a lo largo de la memoria.

Por su parte, el conjunto de herramientas TASTE, si bien de gran utilidad para este tipo de proyectos, presenta algunas dificultades fruto de su falta de madurez, así como debidos a su continuo desarrollo. Como consecuencia, se han notificado diversas erratas en las herramientas e identificado algunas funcionalidades incompletas o no correctamente documentadas.

A pesar de estas dificultades se ha podido completar el trabajo recogido en esta memoria, cumpliendo ampliamente los objetivos marcados al comienzo del mismo. A continuación se relacionan algunas líneas de trabajo futuro relacionadas con el presente Trabajo Fin de Máster.

9.1 Trabajo futuro

Como se ha mencionado a lo largo de la memoria, en el apartado de implementación solo se ha desarrollado un subconjunto de la misma, a fin de demostrar la validez de las herramientas. Esta implementación ha permitido obtener una valiosa experiencia en el desarrollo de sistemas con TASTE. Una línea de trabajo futuro en el ámbito del proyecto es la finalización de dicha implementación, que por la relativa complejidad de la misma, así como de la carga de trabajo requerida bien podría suponer un Trabajo en sí mismo.

Igualmente, como se ha mencionado, el autor forma parte de un nuevo proyecto financiado por la ESA para la mejora y refinamiento del ciclo de vida propuesto por ASSERT/TASTE. De este modo, y gracias a la experiencia adquirida en el desarrollo del presente trabajo, el autor profundizará y colaborará en la definición de una

mejorada metodología para el desarrollo de sistemas empotrados de tiempo real basado en modelos.

Bibliografía

[1] Jorge Garrido, Daniel Brosnan, Juan A. de la Puente, Alejandro Alonso, Juan Zamorano. "Analysis of WCET in an experimental satellite software development". 12th International Workshop on Worst-Case Execution Time Analysis – WCET'2012. OpenAccess Series in Informatics (OASIcs), July 2012.ISBN 978-3-939897-41-5.

[2] D. C. Schmidt. "Model-driven engineering." IEEE Computer, 39(2), 2006.

[3] Rivas, Mario Aldea, and Michael González Harbour. "MaRTE OS: An Ada kernel for real-time embedded applications." Reliable SoftwareTechnologies—Ada-Europe 2001. Springer Berlin Heidelberg, 2001. 305-316.

[4] Juan A. de la Puente, Juan Zamorano, José A. Pulido, Santiago Urueña. The ASSERT Virtual Machine: A Predictable Platform for Real-Time Systems. In Myung Jin Chung, Pradeep Misra (eds.), Proceedings of the 17th IFAC World Congress. IFAC-PapersOnLine, 2008 ISBN 978-3-902661-00-5

[5] Perrotin, Maxime, et al. "TASTE: a real-time software engineering tool-chain overview, status, and future." SDL 2011: Integrating System and Software Modeling. Springer Berlin Heidelberg, 2012. 26-37.

[6] Burns, Alan, Brian Dobbing, and Tullio Vardanega. "Guide for the use of the Ada Ravenscar Profile in high integrity systems." ACM SIGAda Ada Letters 24.2 (2004): 1-74.

[7] Mezzetti, Enrico, Marco Panunzio, and Tullio Vardanega. "Preservation of timing properties with the ada ravenscar profile." Reliable Software Technology, Ada-Europe 2010. Springer Berlin Heidelberg, 2010. 153-166.

[8] Dubuisson, Olivier. "ASN. 1: communication between heterogeneous systems." Morgan Kaufmann Pub, 2001.

[9] ISO. "Ada Reference Manual ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007)," 2007. Available at http://www.adaic.com/standards/ada05.html.

[10] Juan, A., José F. Ruiz, and Juan Zamorano. "An open Ravenscar real-time kernel for GNAT." In Hubert B. Keller and Erhard Plödereder (eds.)Reliable Software Technologies Ada-Europe 2000. Springer Berlin Heidelberg, 2000. 5-15.

[11] Sanz, Ángel, and José Meseguer. "El satélite español UPM-Sat 1." Mundo Científico 169 (1996): 560-567

[12] Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation

[13] Gaisler Research. "LEON3 - High-performance SPARC V8 32-bit Processor." GRLIB IP Core User's Manual, 2012.

[14] SPARC International. "The SPARC architecture manual." SPARC International, Upper Saddle River, NJ, USA.: Version 8, 1992.

[15] Ellsberger, Jan, Dieter Hogrefe, and Amardeo Sarma. "SDL: formal object-oriented language for communicating systems." Prentice Hall, 1997.

[16] European Cooperation for Space Standardization. "ECSS-E-70-41A Space engineering — Ground systems and operations – Telemetry and telecommand packet utilization", January 2013. Available from ESA.

[17] ITU-T X.680 Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)

[18] ITU-T X.691 ASN.1 encoding rules: Specification of Packed Encoding Rules (PER)

[19] AX.25 Amateur Packet-Radio Link-Layer Protocol Version 2.2, 1998

[20] UPMSat-2 Especificación del sistema de software - Software System Specification, versión 1.8. Documento interno.