ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INFORMÁTICOS

UNIVERSIDAD POLITÉCNICA DE MADRID

TRABAJO FINAL DE GRADO

MEMORIA DE SEGUIMIENTO

Integración y validación del software del computador de a bordo del UPMSat-2

Autor
DAVID HERRERO SÁNCHEZ

Supervisor
JUAN ZAMORANO FLORES

27 de Abril de 2018

1. Trabajo realizado

Durante esta primera etapa del desarrollo de la asignatura he cumplido con los siguientes objetivos especificados en el plan de trabajo:

- Estudiar y comprender los distintos subsistemas del OBC.
- Estudiar el diseño y organización del software del OBC.
- Aprendizaje de las herramientas con las que se trabajará.
- Estudio, revisión y comprensión de la SRS.
- Depuración de errores encontrados.
- Integrar todo el software del OBC, pero solo conseguir que funcione el software con todos los módulos que lo componen ejecutando a la vez, todavía falta realizar las pruebas de verificación y validación.
- Implementación de nuevos requisitos que puedan surgir o que hayan sido modificados. (Todavía pueden surgir más requisitos)

También se ha desarrollado el sistema que permite interpretar baterías de pruebas escritas en un fichero y la aplicación gráfica para el diseño de dichas pruebas.

2. Modificaciones en el plan de trabajo

La lista de objetivos a lograr es la siguiente:

- Diseño de pruebas de sistema.
- Validación y verificación de los requisitos de la SRS.
- Depuración de nuevos errores encontrados.
- Integrar todo el software del OBC, realizando las pruebas de verificación y validación.
- Reunirse con los *clientes*.
- Implementación de nuevos requisitos que puedan surgir o que hayan sido modificados.

Los motivos principales de haber pospuesto estos objetivos han sido la falta de tiempo por las tareas de otras asignaturas así como imprevistos que han surgido a nivel personal.

Las reuniones con los clientes se han pospuesto por motivos fuera de mi alcance.

3. Lista de tareas

La lista de tareas por realizar son las siguientes:

- Integrar el software del OBC. (Al completo).
- Probar que el software integrado cumple con la SRS.
  - Creación de ficheros de pruebas con pruebas de caja negra.
  - Creación de ficheros de pruebas con pruebas de caja blanca.
  - Observación de la respuesta del OBC a los TCs recibidos.
  - Registro de bugs y comportamientos anómalos con respecto a lo especificado.
- Corrección de errores.
- Reunión con los clientes.
  - Mostrarles los avances conseguidos.

- Preguntarles sobre cómo actuar ante casos no previstos ni indicados en la SRS.
- Actualizar los resquisitos existentes y añadir nuevos requisitos.

- Llevar a cabo los cambios necesarios para cumplir con los cambios en la SRS.
- Escritura de la memoria del TFG.
- Preparación de la defensa del TFG.

4. Diagrama de Gantt

En la figura 1 se encuentra el diagrama de Gantt correspondiente a las tareas planificadas para la segunda etapa del TFG.

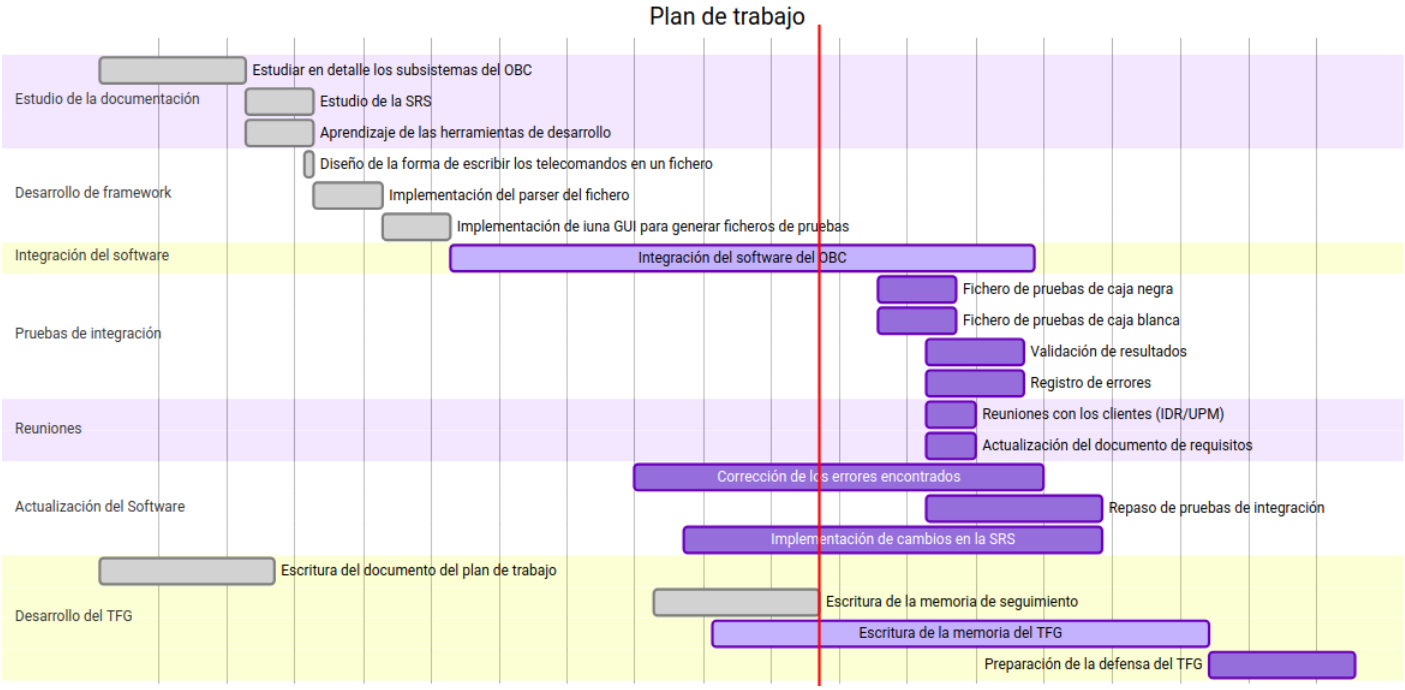

Figura 1: Diagrama de gantt

5. Borrador de la memoria final

En las siguientes páginas se adjunta el borrador de la memoria final del TFG.

CAMPUS
DE EXCELENCIA
INTERNACIONAL

Graduado en Ingeniería Informática

Universidad Politécnica de Madrid

Escuela Técnica Superior de
Ingenieros Informáticos

TRABAJO FIN DE GRADO

Integración y validación del software del computador de a bordo del satélite UPMSat-2.

Autor: David Herrero Sánchez

Director: Juan Zamorano Flores

MADRID, ABRIL DE 2018

ÍNDICE GENERAL

1. Introducción **1**
1.1. Equipamiento 1
1.2. Objetivos 2
1.3. Estructura del documento 2

2. Estado del Arte **3**

3. Bibliografía **5**

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Resumen

Aquí el texto del abstract. Será completado una vez el resto de capítulos estén completos.
Palabras clave: palabra 1, palabra 2, palabra 3. . .

Abstract

Here goes the abstract text. It will be filled once the rest of chapters are finished.

Keywords: keyword 1, keyword 2, keyword 3…

1 INTRODUCCIÓN

El satélite UPMSat-2 forma parte del proyecto UPMSat-2 UNION liderado por el Institudo de Microgravedad Ignacio Da Rivas (IDR) junto con la Universidad Politécnica de Madrid (UPM).

Como denota el nombre del satélite, este proyecto nace como sucesión al ya lanzado exitosamente UPM-SAT1, y está siendo desarrollado además del IDR por el grupo de investigación de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos (STRAST[1]).

Como detalles técnicos del satélite cabe destacar la información proporcionada en la página web del proyecto[2]:

- Peso: 50 kg.
- Medidas: 0.5m x 0.5m x 0.6m (ancho, largo, alto).
- Volumen útil: 0.4m x 0.4m x 0.25m.
- Carga útil: 15 kg.
- Potencia: 15 W.
- Órbita polar: 600 km de altitud.
- Vida útil: 2 años.

1.1 Equipamiento

Como equipamiento hardware el UPMSat-2 dispone de los siguientes componentes:

- Seis paneles solares (uno por cada eje X, Y, Z).
- Seis sensores de temperaturas.
- Una batería disponer de corriente en las zonas con ausencia de luz solar.
- Tres magnetómetros para medir el campo magnético de la Tierra.

- Tres magnetopares (uno por eje) para generar un campo magnético que permita mantener la orientación (también llamada actitud) del satélite en la posición adecuada.
- Una tarjeta de radio para las comunicaciones con la estación de Tierra.
- Un volante o rueda de inercia[1] para experimentación.
- Tres microswitches térmicos (microthermal switches) también para experimentación.
- Un computador de a bordo (OBC[2]) encargado de controlar y coordinar el funcionamiento mediante software de todos los dispositivos anteriores.

1.2 Objetivos

Este Trabajo de Fin de Grado (TFG) tiene como objetivo integrar el software (SW) de vuelo del OBC.

Los objetivos concretos son los siguientes:

- Estudio y comprensión del SW de los distintos subsistemas ya desarrollados.
- Análisis en profundidad del documento de requisitos software (SRS [3]).
- Desarrollo de un programa que permita ejecutar una batería de pruebas.
- Desarrollo de un programa que permita diseñar las baterías de pruebas de forma gráfica.
- Validación y verificación de los requisitos SW.
- Arreglar los fallos SW detectados en la validación.

1.3 Estructura del documento

Este documento está estructurado de la siguiente forma: en el capítulo 2 se describe a grandes rasgos cuales son las restricciones bajo las que se ha desarrollado el SW del OBC, sus necesidades y cómo son las herramientas de desarrollo de las que se dispone.

A COMPLETAR CON EL RESTO DE CAPÍTULOS.

[1] Un volante de inercia se basa en el principio de conservación del momento angular para controlar la actitud de satélites. Su funcionamiento básico consiste en un motor que gira y genera un momento angular que es transmitido al satélite en el eje correspondiente. De esta forma pueden emplearse un conjunto de ruedas de inercia en lugar de usar magnetopares.

[2] On Board Computer

[3] Software Requirements Specification

2 ESTADO DEL ARTE

El OBC del UPMSat-2 es un sistema empotrado (o embebido), ya que está diseñado para cumplir un conjunto de necesidades específicas, y también un sistema de tiempo real por la necesidad de que responda a los estímulos del entorno físico dentro de un intervalo de tiempo determinado.

A todo esto hay que sumarle que el satélite en sí es un sistema de alta integridad dado que una vez en órbita no pueden hacerse reparaciones, por lo que el diseño del software debe haberse realizado muy meticulosamente para actuar correctamente ante cualquier problema que surja.

Por estos motivos, es necesario emplear herramientas que permitan cumplir con estas necesidades de la manera más simple posible.

Un lenguaje de programación que facilita estas tareas es Ada, ya que incorpora de forma innata todos los mecanismos necesarios para el control de tiempo y la interacción entre tareas o hilos de ejecución. Además cabe destacar que el lenguaje Ada es uno de los lenguajes oficiales de la Agencia Espacial Europea (ESA[3]) para este tipo de sistemas.

El software del OBC del UPMSat-2 por tanto ha sido desarrollado en el lenguaje Ada usando un compilador cruzado[1] para procesadores de la arquitectura SPARC, en concreto para el procesador LEON3.

Como cada uno de los subsistemas que componen el OBC tiene sus propias tareas para llevar a cabo su funcionalidad, es necesario controlar de alguna manera sus prioridades y la compartición de recursos entre ellas.

Un ejemplo de la importancia de ajustar correctamente las prioridades de las tareas es el siguiente: Tenemos dos tareas: una para el control de la actitud del satélite (tarea crítica) y otra para el control de un experimento (tarea prescindible). Si la tarea de control de actitud necesita comprobar el estado del satélite cada medio segundo y resulta que ambas tareas tienen la misma prioridad, puede ocurrir que la tarea del experimento esté ocupando la CPU e impida cumplir con los requerimientos temporales de la primera, dando posibilidad al descontrol del satélite y la pérdida de las comunicaciones.

[1] Compilador que genera un código ejecutable para otra arquitectura distinta.

Una forma eficiente de ser capaces de controlar todo este tipo de detalles es seguir un conjunto de pautas de programación que faciliten el análisis de tiempos, ramas de ejecución y el control de los distintos recursos del sistema.

Para ello, se ha seguido para el desarrollo del software el perfil de Ravenscar [4], que es un subconjunto de las características de Ada que restringe ciertos aspectos importantes como los siguientes:

- Una tarea con prioridad N no puede acceder a recursos compartidos (conocidos como *objetos protegidos o protected objects*) de prioridad inferior. De esta forma se evita que tareas de menor prioridad *adelanten* a esta tarea.
- Si una tarea con prioridad N accede a un objeto protegido de prioridad M con $M \geq N$, esta tarea adquiere la prioridad M hasta que haya terminado de usar dicho objeto.
- Una tarea no puede llamar a otra directamente. La única forma de comunicación entre tareas se debe hacer de forma indirecta a través de los objetos protegidos.
- Un objeto protegido solo puede tener una *entry* [2]. De este modo se evita que haya más de una tarea esperando en un recurso compartido.
- Solo puede haber una tarea esperando en la *entry* de un objeto protegido, es decir, solo hay capacidad en la cola de entrada para una única tarea.
- Todos los tiempos de espera (sentencias *delay*) deben ser relativos. La razón es la siguiente: si una tarea tiene más adelante de su hilo de ejecución un delay relativo de 30 segundos con respecto al tiempo actual y no puede ejecutar durante 25 segundos al haber sido desalojada de la CPU, al volver a su ejecución y llegar a dicho delay no se detendrá durante esos 30 segundos, sino solo durante 5 para cumplir con los tiempos establecidos.

En el caso de que no se cumpla alguna de las normas anteriores se producirá una excepción que provocará el fin de la ejecución del software.

Gracias al perfil de Ravenscar se consigue controlar de forma mucho más eficiente la concurrencia, evitando entre otras cosas la aparición de *deadlocks*[3].

[2] Procedimiento en el que una tarea va a quedarse esperando hasta que se dé una condición establecida en el objeto protegido.

[3] Situación en la que una tarea se queda esperando indefinidamente para continuar su ejecución.

3 BIBLIOGRAFÍA

[1] G. de investigación de Sistemas de Tiempo Real y Arquitectura de Servicios Telemáticos. (2018). Página web oficial de STRAST, dirección: http://web.dit.upm.es/~str/.

[2] I. de Microgravedad Ignacio Da Rivas. (2018). Página web oficial del proyecto upmsat-2 union, dirección: http://www.idr.upm.es/tec_espacial/upmsat2/01_UPMSAT2.html.

[3] E. S. Agency. (2018). Lenguajes de programación avalados por la ESA, dirección: http://www.esa.int/TEC/Software_engineering_and_standardisation/TECRFBUXBQE_0.html.

[4] A. Burns, «The ravenscar profile», 1999. dirección: http://www.dit.upm.es//~str/proyectos/ork/documents/RP_spec.pdf.