Design of On-Board Software for an Experimental Satellite[*]

Alejandro Alonso Emilio Salazar Juan A. de la Puente

Grupo de Sistemas de Tiempo Real
y Arquitectura de Servicios Telemáticos,
Universidad Politécnica de Madrid (UPM)

Abstract

UPMSat-2 is an experimental micro-satellite mission which is being developed at the Technical University of Madrid (UPM) with the collaboration of several industrial companies and research institutions. This paper describes the design of the software architecture for the on-board computer of the satellite.

1 Introduction

UPMSat-2 is a project aimed at developing an experimental micro-satellite that can be used as a technology demonstrator for several research groups at UPM, the Technical University of Madrid. The project is led by IDR/UPM,[1]and is being carried out with the participation of STRAST/UPM,[2] TECNOBIT,[3] and some other University and industry groups that are active in the Spanish aerospace sector. The launch date is currently set to early 2015.

The STRAST/UPM group is in charge of developing all the software required for the UPMSat2 mission. A previous paper (de la Puente et al., 2012) described the main aspects of the hardware and software architecture of the on-board computer (OBC). This paper describes in detail the methodological approach taken for on-board software development and the software tools that are being used for it.

Section 2 summarises the main aspects of the OBC functionally and architecture. Section 3 describes the software methodology that has been used to develop the software. Section 4 shows the software architecture of the system, and section 5 describes the implementation of the ADCS component as an example. Finally, some conclusions and a summary of the remaining work to be done are recorded in section 6.

[*]This work has been partly funded by the Spanish Government, project no. TIN2011-28567-C03-01 (HI-PARTES).

[1]"Ignacio Da Riva" Institute, see `www.idr.upm.es`.

[2]Real-Time and Telematic Services Group, see `www.dit.upm.es/str`.

[3]`http://www.tecnobit.es/`.

2 On-board computer overview

2.1 On-board computer functions

The UPMSat-2 On-Board Computer (OBC) is an embedded computer system that is in charge of executing all the control and monitoring functions of the satellite, including:

- Attitude determination and control (ADCS). The attitude of the satellite is determined from magnetic field and solar sensor measurements, and is corrected by means of a set of magnetorquers.
- Telemetry and telecommand management (TMTC). Telemetry messages are sent to the ground station, and telecommands are received from it, by means of a radio unit operating in the UHF 400 MHz band.
- Platform monitoring and control. Important sensor data (voltages, temperatures, orbit and attitude parameters) are measured periodically and their values checked for safe operation. Monitoring data are periodically sent to the ground station by telemetry.
- Experiment management. The payload of the satellite consists of a set of experiments provided by several research groups and industrial companies. The experiments include testing new sensor and actuator designs, as well as collecting operational data and experimenting with new control algorithms.

2.2 Computer system architecture

The hardware architecture is based on a LEON3 computer[4] implemented on an FPGA, with a clock frequency of 100 MHz. The computer board includes 1 MB EEPROM, 4 MB SRAM, and a number of analog and digital input and output lines and serial ports.

Figure 1 shows a context diagram of the system.

3 Software development methodology

3.1 Model-driven software engineering

Following previous experience in the ASSERT,[5] CHESS,[6] and HI-PARTES,[7] projects, a model-driven engineering approach (Schmidt, 2006; Zamorano and de la Puente, 2010) has been chosen to develop the on-board software system. Accordingly, the system is first described as a high-level platform-independent model (PIM), which is later transformed to a platform-specific model (PSM), including all implementation details. The system is implemented by automatically generating code from the PSM.

[4]See `www.gaisler.com`.

[5]Automated proof-based System and Software Engineering for Real-Time systems, `www.assert-project.net`

[6]Composition with Guarantees for High-integrity Embedded Software Components Assembly, `www.chess.project.ning.com`.

[7]High-Integrity Partitioned Embedded Systems, `www.dit.upm.es/str/projects/hipartes`.

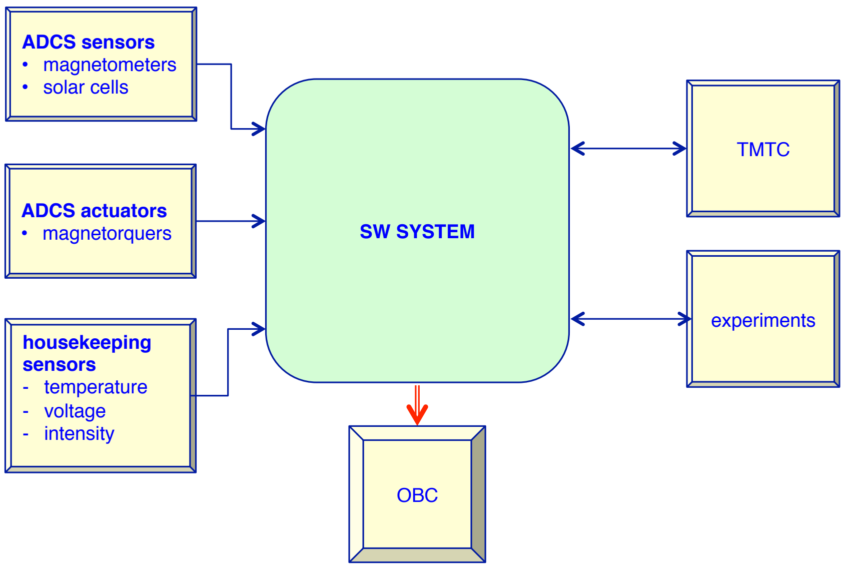

Figure 1: OBC context diagram.

3.2 UPMSat-2 software process

The main steps of the software process for the UPMSat-2 software are shown in figure 2. The process starts with the development of a platform-independent *system model*. This model includes all the aspects of the application software that do not depend on the details of the executing platform. The UML2 language (OMG 2011a) is used as a basic notation for describing the functional properties of the system. Functional models in other languages, such as Simulink®, can be integrated into this model as well (see e.g. Garrido et al., 2012). Real-time properties can be specified in the model using the UML MARTE profile (OMG 2011b). For instance, a periodic activity can be modelled as a class annotated with the stereotypes GQAM::WorkloadEvent and GRM:SchedulableResource to specify a periodic arrival pattern and a deadline for the activity.

From this model an *analysis model* is derived using an automatic transformation tool. The analysis model is based on the MAST modelling language and tools (González Harbour et al., 2001), which are used to carry out static timing analysis in order to validate the real-time behaviour of the model. To this purpose, some assumptions about the execution platform and execution time data are needed, which makes this model a platform-specific model (PSM).

At a lower level of abstraction, a *neutral model* is used in order to reduce the gap between higher-level models and source code. It is automatically generated from the system model, and also incorporates information coming from the analysis model. It is described in plain UML, and is not intended to be manipulated by the designer.

Finally, source code is generated from the neutral model using a specific tool. The tool produces Ravenscar-compliant concurrent Ada code (ARM05), which can be executed on the ORK+ real-time kernel (de la Puente et al., 2008) which

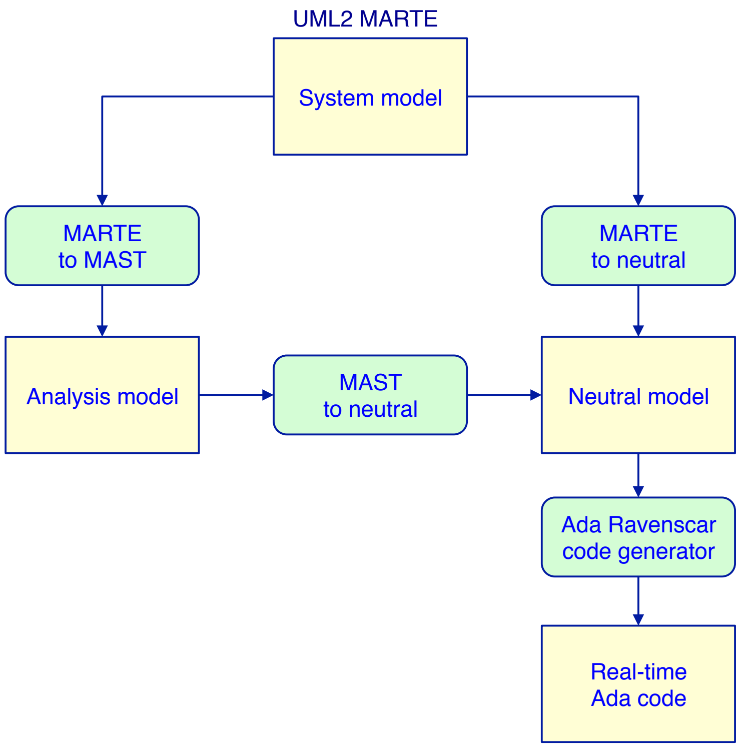

Figure 2: Model-based software process

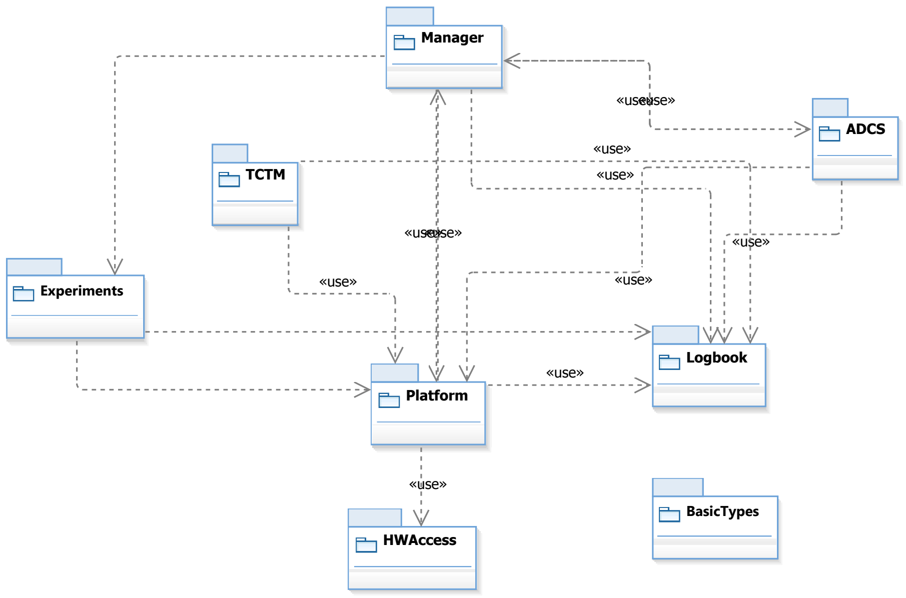

Figure 3: Software architecture

is part of the GNATforLEON toolchain.[8]

4 Software Architecture

The UPMSat2 software system model is an architectural model which is composed of a set of components. The connections between components are based on their *provided interfaces*, which include all the public operations exported by each component. Internally, a component is structured as a set of software objects that implement the operations in the interface and perform additional duties. The design complies with the Ravenscar computation model (Burns et al., 2004), and thus the basic level components are restricted to be cyclic, sporadic, and protected objects.

The overall architectural design of the UPMSat2 software system is shown in figure 3. It consists of the following components:

Manager: The satellite may be in a number of operational models that determine its expected behaviour. The transitions between models are driven by system events, such as reaching a low battery level, detection of errors, or receiving specific telecommands. Each software component has a specific behaviour for each of the modes. The Manager is in charge of setting the operational mode and notifying it to the rest of the software components, so that they can adapt their operation accordingly. This component is also responsible for handling errors.

[8]`www.adacore.com`.

Platform: This component is mainly in charge of general satellite housekeeping functions. It gets state values from the satellite by reading sensors, checks if the values are correct, and prepares housekeeping telemetry messages to be sent to the ground station. The platform component also provides a more abstract access to the sensor measurements.

The internal structure of the platform component relies on a cyclic activity that is executed according to the operation mode. Sensors are grouped according to the type of measurements, such as temperatures, voltages, magnetometer currents, and solar cells. Last measured data will be kept in memory for providing this information to other modules, such as ADCS or TC/TM.

Data is read by using the HWAccess components, which provides raw data. The platform is in charge of converting them into engineering units, and analysing their validity. This can be done by checking that they are within a valid range, or by analysing it with respect to previous samples.

HWAccess: This component encapsulates the access to the hardware sensors and actuators. In particular, it includes the required drivers. It provides a set of operations for reading and writing the required data.

Logbook: This component encapsulates access to a buffer storage area. It is used to store telemetry messages that are generated during system operation, but cannot be sent during the periods when there is no antenna coverage from the ground station.

ADCS: This component is in charge of determining and controlling the satellite attitude. The attitude of the satellite is calculated based on two types of sensors: magnetometers and solar cells. Magnetorques are used for correcting the satellite attitude. The ADCS internal structure and detailed design is presented in section 5.

TCTM: This component includes the operations for communicating with the ground station: receiving telecommands (TC), and sending telemetry (TM), with information of the satellite state and operations. Some examples of telecommands are configuration of satellite parameters, requests for changing the operational model, activation of experiments, etc.

Experiments: The UPMSat2 satellite will include a number of experiments in order to test the behaviour of selected electronic equipment in a space environment, or to check some new algorithms or measurement techniques. The experiments are activated one at a time, when the satellite is in a particular operational mode, when an appropriate command is received from the ground station. The data resulting from the experiment are sent to the ground station by means of telemetry messages.

5 Detailed design of the UPMSat2 ADCS

This section describes the detailed design of one of the UPMSat2 software components, the ADCS subsystem, as an example of how this step of the software development process is performed. Figure 4 shows the class diagram that makes up the ADCS internal structure. It consists of four elements.

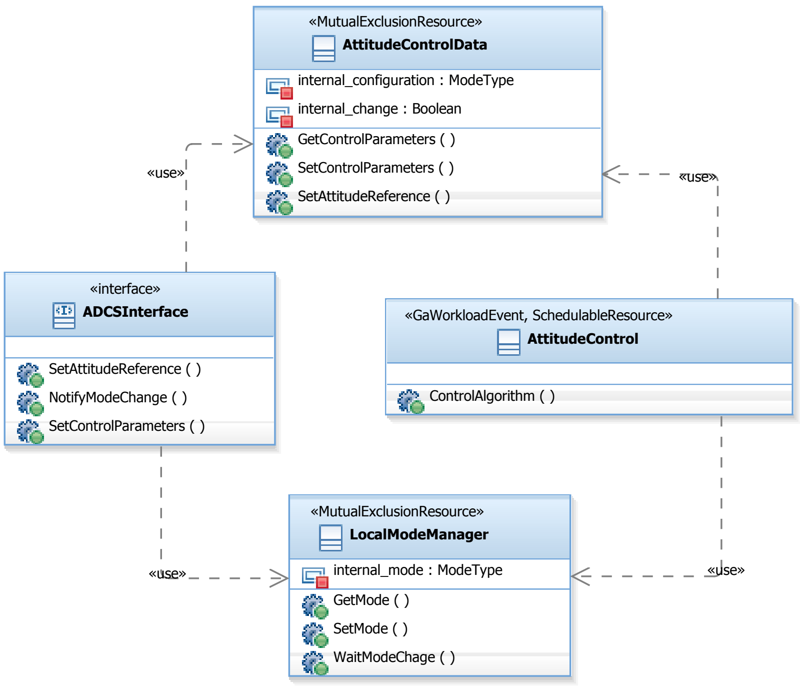

Figure 4: Class diagram of the ADCS component.

ADCSInterface: This interface defines the operations that the ADCS component will make public for the rest of the system, i.e. its provided interface. The operations specified here are implemented in other ADCS classes.

LocalModelManager: This class is used to manage the operational model at a local level. The global operational mode is set by the higher-level

Manager component, whereas the local manager provides access to the current mode and mode change requests for the rest of the ADCS subsystem. It also provides an operation that can be used by other elements in the ADCS subsystem to wait for a mode change to occur, if they are inactive in the current mode.

Since the local mode manager is used by other elements exhibiting a concurrent behaviour, its operations are defined to be executed in mutual exclusion. In terms of the Ravenscar computational model it is a *protected class*.

AttitudeControlData : This class handles control parameters for the ADCS subsystem. It provides operations for changing the control parameters and for setting the attitude reference values. This is also a *protected class*.

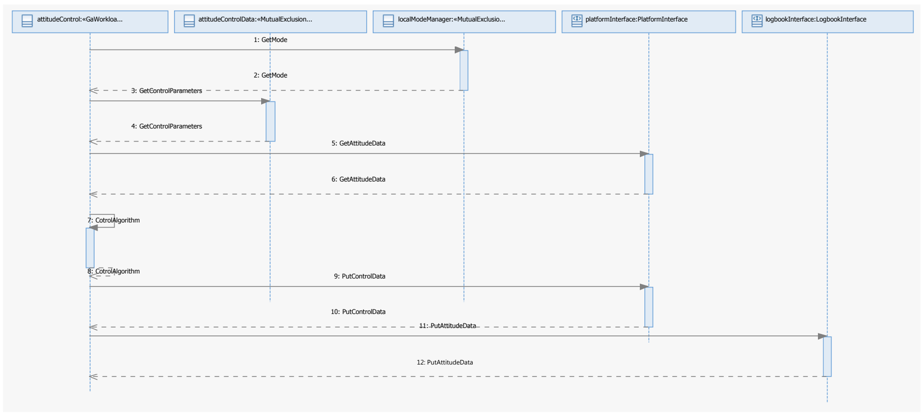

Figure 5: Sequence diagram of the ADCS operation.

AttitudeControl : This class does the real job of performing the satellite attitude control. It contains a thread that periodically gets the mode and current control parameters, applies the control algorithm, and actuates by means of the magnetorquers. The sequence diagram of this class is shown in figure 5. It is modelled as a schedulable resource with an event activation pattern, using UML MARTE notation. In terms of the Ravenscar model this class is a *periodic class*.

The next step is to derive a deployment model corresponding to this class diagram. This model uses objects that are instances of the previous classes, and defines parameters, such as priorities, periods, and deadlines. It is represented as a component UML model, as shown in figure 6. This representation is used for response time analysis and code generation purposes. Mutual exclusion objects generate protected objects, schedulable resources with periodic activation patterns are transformed into periodic tasks, and the full component will be decomposed into packages, where the operations in the component interface are included in its public specification.

The control algorithm has been design using a model in Simulink,[9] in order to keep up with the general practice of control engineers, as part of the platform-independent model. This high-level model has been integrated in the previous generated code by carrying the following steps:

- C sequential code for the control algorithm has been automatically generated from the dynamic model using the appropriate Simulink toolkit.
- Sequential code is embedded into the periodic task generated by the MDD toolset.
- The code is uploaded to the computer board for testing. The Simulink hardware-in-the loop facilities can be used to test the subsystem and validate its operation against a dynamic model of the satellite attitude.

[9]`www.mathworks.com/products/simulink`.

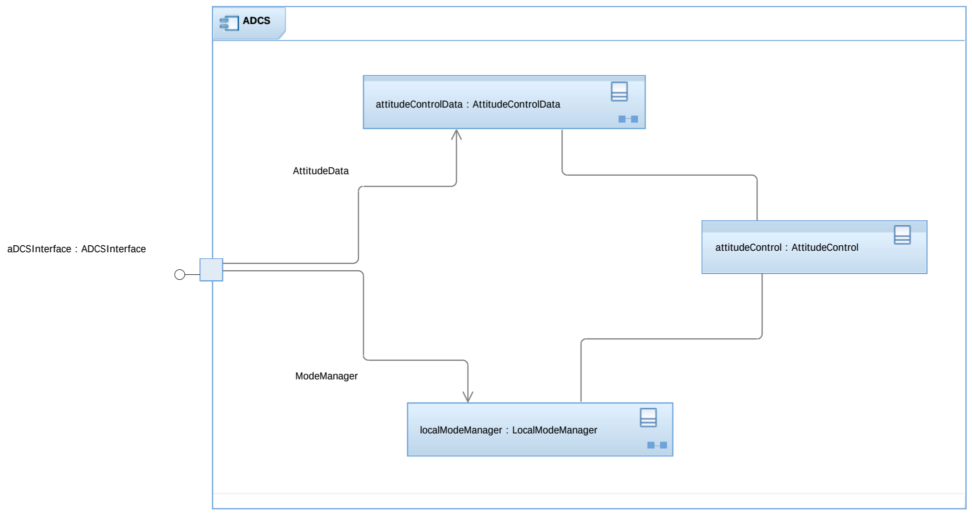
ADCS
attitudeControlData : AttitudeControlData
AttitudeData
attitudeControl : AttitudeControl
aDCSInterface : ADCSInterface
ModeManager
localModeManager : LocalModeManager

Figure 6: ADCS component in the deployment model

6 Conclusion

The UPMSat2 project is an excellent occasion for our group to apply recent research on real-time software engineering to a real space mission. Although the scope and size of an academic system is limited, it is still a real development which will give us the opportunity to show that the ORK+ technology and modern software engineering methods can be successfully applied in such a demanding kind of applications.

We expect to devote the next months until the launch date to completing the development of the software, and to carrying out all the necessary verification and validation activities that will ensure that the software is correct in its final form.

Acknowledgments

The authors wish to acknowledge the collaboration of the rest of the UPMSat2 team.

References

Alan Burns, Brian Dobbing, and Tullio Vardanega. Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV:1–74, June 2004. ISSN 1094-3641. doi: http://doi.acm.org/10.1145/997119.997120. URL `http://doi.acm.org/10.1145/997119.997120`.

Juan A. de la Puente, Juan Zamorano, José A. Pulido, and Santiago Urueña. The ASSERT Virtual Machine: A predictable platform for real-time systems.

In Myung Jin Chung and Pradeep Misra, editors, *Proceedings of the 17th IFAC World Congress*. IFAC-PapersOnLine, 2008.

Juan A. de la Puente, Juan Zamorano, Alejandro Alonso, and Daniel Brosnan. A real-time computer control platform for an experimental satellite. In *Jornadas de Tiempo Real — JTR-2012*, 2012. Available at `http://www.ctr.unican.es/jtr12/programme.html`.

Jorge Garrido, Daniel Brosnan, Juan A. de la Puente, Alejandro Alonso, and Juan Zamorano. Analysis of WCET in an experimental satellite software development. In Tullio Vardanega, editor, *12th International Workshop on Worst-Case Execution Time Analysis*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, pages 81–90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2012. ISBN 978-3-939897-41-5.

Michael González Harbour, J. Javier Gutiérrez, J. Carlos Palencia, and José María Drake. MAST modeling and analysis suite for real time applications. In *Proceedings of 13th Euromicro Conference on Real-Time Systems*, pages 125–134, Delft, The Netherlands, June 2001. IEEE Computer Society Press.

ARM05. *ISO/IEC 8652:1995(E)/TC1(2000)/AMD1(2007): Information Technology — Programming Languages — Ada.* ISO, 2007.

OMG 2011a. *OMG Unified Modeling Language (OMG UML), Infrastructure*, 2011. URL `http://www.omg.org/spec/UML/2.4.1/`. Version 2.4.1.

OMG 2011b. *UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded Systems*, 2011. URL `http://www.omg.org/spec/MARTE/`. Version 1.1.

Douglas C. Schmidt. Model-driven engineering. *IEEE Computer*, 39(2), 2006.

Juan Zamorano and Juan A. de la Puente. Design and implementation of real-time distributed systems with the ASSERT virtual machine. In *IEEE Conference on Emerging Technologies and Factory Automation — ETFA 2010)*, pages 1–7, sept. 2010.