Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Model-Driven Design of Real-Time Software for an Experimental Satellite [⋆]

Juan A. de la Puente [*] **Jorge Garrido** [*] **Juan Zamorano** [*] **Alejandro Alonso** [*]

[*] *Universidad Politécnica de Madrid (UPM), Spain*
E-mail: `str@dit.upm.es`

Abstract: Model-driven engineering has gained widespread interest as a means to raise the abstraction level in software development, thus lowering cost and increasing efficiency. In this paper, a case study on using this approach to design the real-time software for UPMSat-2, an experimental micro-satellite, is described. The functionality of the software includes attitude determination and control, on-board data handling, platform monitoring and control, and payload management. A mix of modelling and software engineering tools have been used, enabling automatic code generation of most of the application software. The lessons learned from such an approach are analysed in the paper, and the use of model-driven engineering in real-time control systems is discussed.

Keywords: Real-time systems, computer software, software engineering, satellite control.

1. INTRODUCTION

Model-driven engineering (MDE) is a software development approach that allows engineers to use high-level abstractions to define the components of a software system throughout the software development cycle (Schmidt, 2006). It is based on the extensive use of models to describe and analyse the system behaviour, thus supporting reasoning on the system properties at an abstract level. Models provide support for different types of problems: description of concepts, validation of these concepts based on checking and analysis techniques, and generation of code and other implementation components.

This approach has been familiar to control engineers for a long time. Control systems are customarily designed using abstract models, based on well-established mathematical foundations. Controller models are combined with plant models, and their configuration and parameters can be adjusted as necessary until a satisfactory behaviour is obtained. The resulting controller can then be implemented using a variety of technologies, including computer hardware and software (see e.g. Albertos and Mareels, 2010, for a comprehensive approach to control systems design).

The advent of modern simulation tools has provided further support to the use of models in control systems design. Complex systems that cannot be described with simple linear models can be simulated, and the effect of different control algorithms can be experimented on them in order to get the desired behaviour. Simulation models are also often used to validate the results of other control design methods. Implementation code for the control algorithms can be automatically generated from the models, thus simplifying the development of control systems.

The design of complex control systems usually encompasses much more than control algorithms. Other components, such as external interfaces, communications, data storage and processing, etc., contribute to a large extent to the complexity of the system. MDE aims at using a modelling approach to software development at large, including all kinds of components of real-time control systems. It is worth noting that not all of the above domains have as neat a mathematical foundation as automatic control, although rigorous notations are often available to build models.

We analyse in the following the use of an MDE approach for designing the on-board software system for UPMSat-2, 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. The functionality of the software includes attitude control, on-board data handling, platform monitoring and control, and payload management.The use of different languages to build models of various parts of the software at different abstraction levels is illustrated, and the tools used to support the development are described, with special focus on the attitude control system. The lessons learned from the work are discussed, and the approach is compared to other kinds of development processes.

The rest of the paper is organised as follows. The characteristics of the UPMSat-2 satellite and the general structure of the control software are described in section 2. High-level modelling activities are described in section 3. Detailed design models and code implementation are described in section 4. Model validation is discussed in section 5. Finally, section 6 contains the conclusions of the study and lessons learned.

[⋆] This work was supported in part by the Spanish National R&D&I Plan, project HI-PARTES (TIN2011-28567-C03-01).

2. THE UPMSAT-2 SYSTEM

2.1 Overview

UPMSat-2 is a micro-satellite with a mass of approximately 50 kg and a cubic geometric envelope of $0.5 \times 0.5 \times 0.6$ m. The satellite will be set to a polar sun-synchronous orbit at 600 km altitude with a period of about 97 min. The launch is planned in 2015, and the expected life of the satellite is two years.

The aim of the mission is to act as a technical demonstrator and an educational platform for University researchers and associated companies. The payload consists of a number of experiments proposed by research groups and industrial companies, including different kinds of sensors and actuators to be tested in a space environment.

Figure 1 shows the general structure of the spacecraft. Power is generated from solar panels covering five sides of the satellite. Communications with a ground station are carried out by means of two radio links using the UHF 400 MHz band, with a maximum data transfer rate of 9600 b/s.

There is an on-board computer (OBC) that carries out all the data handling, supervision and control functions of the spacecraft platform and the experiments. The computer is based on a LEON3 processor (Gaisler Research, 2012) implemented on a radiation-hardened FPGA with also including 4 MB SRAM, 1 MB EEPROM, timers, analog inputs and digital I/O, making up a system-on-chip configuration (SoC).

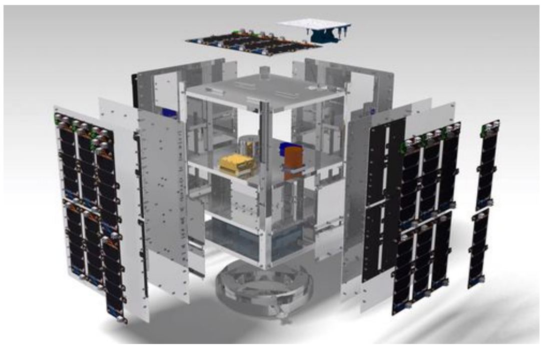

Fig. 1. Structure of the UPMSat-2 satellite.

2.2 On-board software functions

The main functions to be carried out by the on-board computer software are:

- Attitude determination and control (ADC). The attitude of the satellite is determined from the data provided by magnetic sensors, and controlled by means of magnetic actuators. Other configurations are included as experiments.
- Platform monitoring (*housekeeping*), by periodically measuring temperatures and voltages from platform sensors. Deviations from nominal ranges may cause switching to an error mode and sending error messages to the ground station.
- On-board data handling (OBDH), including the execution of telecommands (TC) received from the ground station and the generation of telemetry (TM) messages to be sent to the ground station.
- Experiment management. The payload of the satellite consists of a number of experiments with different instruments an devices.

Figure 2 shows a context diagram of the OBC and its connections to the satellite platform.

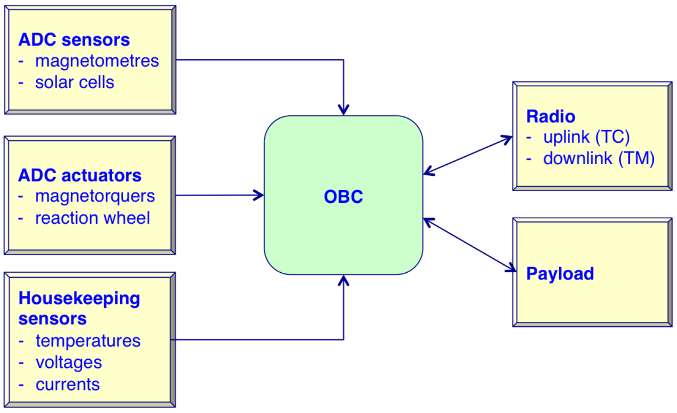

Fig. 2. Context diagram of the OBC subsystem.

2.3 Attitude determination and control

The attitude of the satellite is the orientation of the satellite, defined as the relationship between a body reference frame and an orbit reference frame. The attitude can be represented as the rotation matrix or —more conveniently— by the quaternion defining the transformation between both reference frames (de Ruiter et al., 2012).

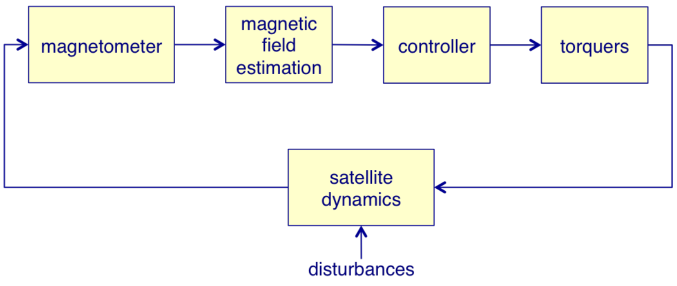

Fig. 3. Attitude control system.

In the UPMSat2 mission, the attitude control system has to keep the Z_B axis normal to the orbit, so that the radio antenna is always visible from the Earth. The body of the satellite must rotate slowly around this axis, in order to provide spin stabilization (Sidi, 1997). A simple control algorithm has been designed to this purpose, based on estimates of the Earth magnetic field vector and its derivative (Farrahi et al., 2013). The control algorithm computes the actuation signals to be fed to three magnetorquers in order to make the required corrections in the orientation of the satellite (figure 3).

3. SOFTWARE DESIGN

3.1 Model-based development process

The overall software architecture of the satellite was designed from the context diagram shown in figure 2. The on-board software has been developed using a model-based process consisting of a series of models that are progressively refined until the implementation code can be generated (figure 4).

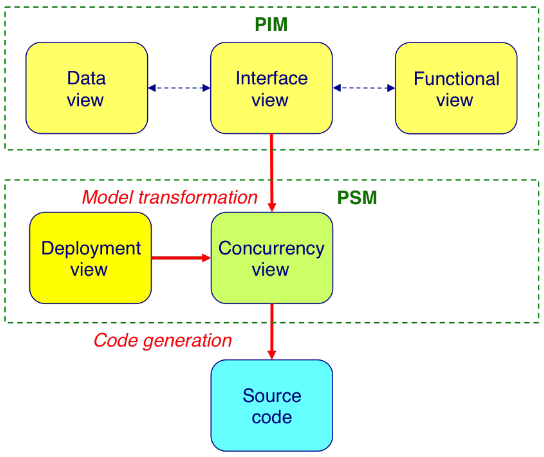

Fig. 4. Model-based software process.

As a first step, a *platform-independent model* (PIM) is built in order to represent the intended behaviour of the system without taking into account the platform-specific details, such as the computer hardware, specific devices, or operating system characteristics. The PIM includes several model views:

- *Data view.* The data types that are used in other model views are defined here, using an implementation-independent notation such as ASN.1 (ITU, 2008).
- *Functional view.* This view describes the functional behaviour of the system components, with no reference to real-time or concurrency aspects. Simulink (Mathworks, 2013) and SDL (ITU, 2011) have been used to this purpose.
- *Interface view.* This model view describes the architecture of the software system in terms of its constituent parts and the relationships between them. AADL (Feiler, 2012) is the language of choice for expressing the architecture and gluing together the system components.

The PIM can be used to reason about properties of the system at an abstract level. From it, a *platform-specific model* (PSM) can be derived, taking into account the characteristics of the computer platform to be used. This model consists of two main views:

- *Deployment view.* This view is used to model the platform itself, including processors, memory, interconnection buses and networks, and operating systems. The software components making up the interface view are mapped to platform components in the deployment view.
- *Concurrency view.* This view models the concurrency and real-time aspects of the system. Its elements are concurrent tasks with real-time attributes, shared data objects and other kinds of software components that can be implemented in terms of operating system services. This view can be generated from the interface view and the deployment view provided the former includes the real-time requirements to be guaranteed at the interface level.

AADL has been used for both PSM views, as it supports modelling of low-level hardware and software elements in addition to abstract model views.

The concurrency view, together with PIM views, contain enough information to automatically generate the implementation code for the software system.

We have used the TASTE[1] toolset (Perrotin et al., 2012) to support the above development process. The toolset includes graphic and textual editors for AADL and ASN.1, and supports automatic building of the concurrency view and Ada code generation.

3.2 High-level models of the UPMSat-2 software

The views in the UPMSat-2 PIM have been built using a variety of modelling languages. Some examples of model elements are given in the following paragraphs.

Interface view. Figure 5 shows the interface view, with all the components making up the software architecture of the OBC represented as subsystems in the graphic AADL notation, and the interfaces between them represented as data flows. The software architecture reflects the structure of the main functions listed in section 2.1, with an additional component, the *orchestrator*, which coordinates the operating modes of the satellite.

The different subsystems are modelled in the respective functional views, and the data types used in the interface are modelled in the data view.

Functional view. In order to model the functional behaviour of the different components, different notations have been used. In particular, SDL has been used for the state-driven components, such as the payload manager and the TMTC[2] controller, and Simulink models have been used for continuous time functions, such as the attitude control. As an example, we provide more detail on the attitude determination and control subsystem.

Figure 6 shows the top-level Simulink model that has been used to adjust the attitude control law. The model includes the different torques that affect the attitude of the satellite, including the magnetic field of the Earth, the controlling torque, and some disturbances. The torques are input to a model of the rotational dynamics of the spacecraft, from which the attitude (both in quaternion and rotation matrix forms) and the angular speed vector are output.

The block labelled *controller* represents the control algorithm. Several kinds of control laws and different parameter values have been used in order to select the best control performance with the available sensors and actuators, which are also modelled as part of this block. See Farrahi et al. (2013) for a detailed explanation.

Data view. All the data types used in the interface view have been modelled using ASN.1. Listing 1 shows

[1] `taste.tuxfamily.org`
[2] Telemetry and telecommand.

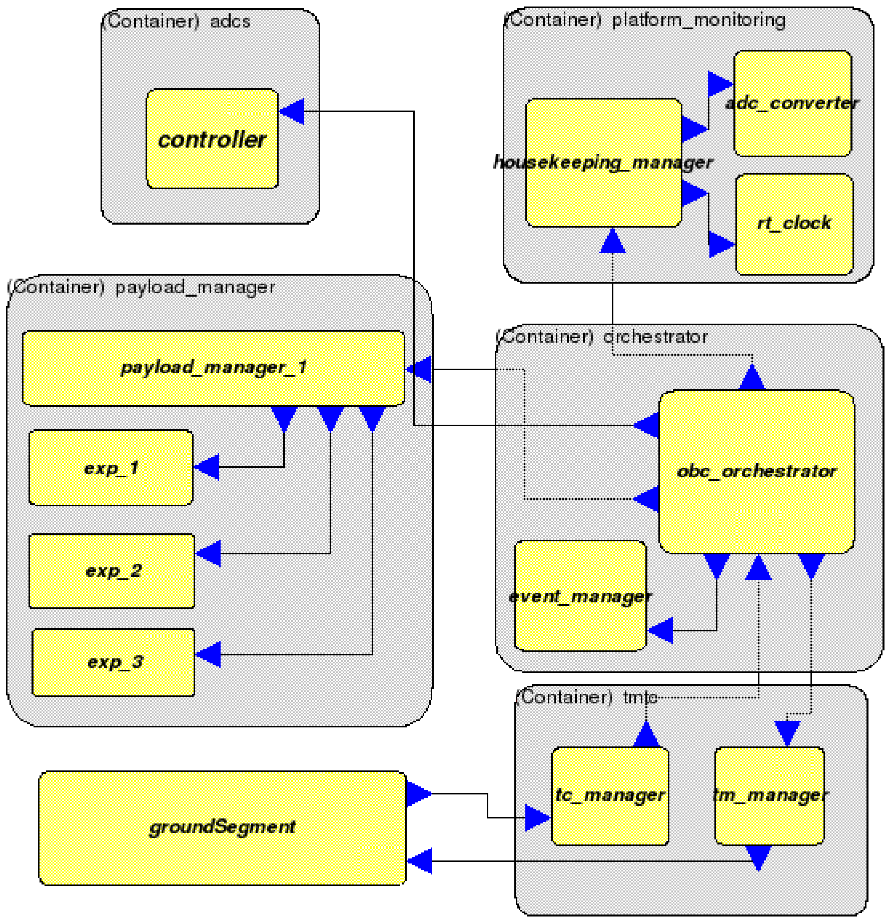

Fig. 5. Interface view of the UPMSat-2 software.

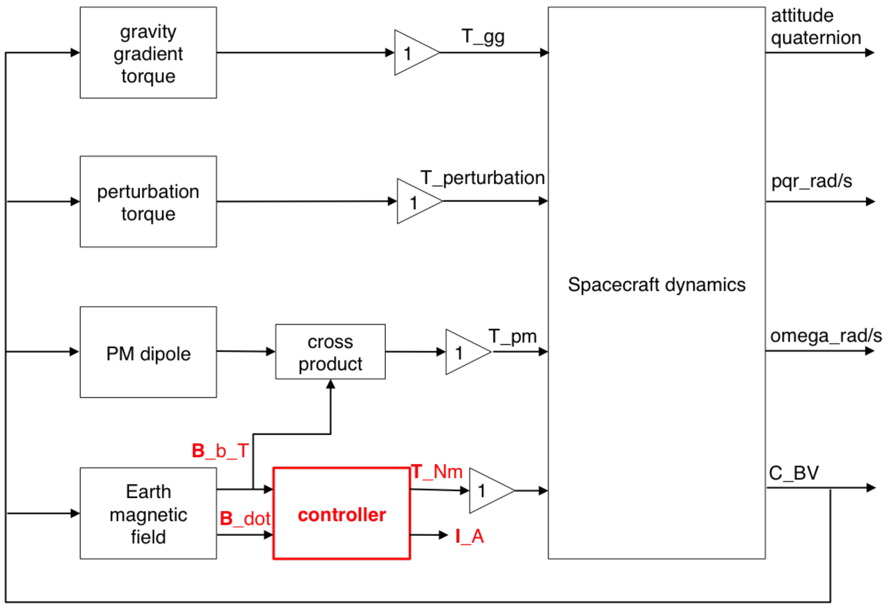

Fig. 6. Functional model of the attitude dynamics in Simulink.

an example of the definition of the types used by the ADC system.

```
MagnetometerAxisType ::= SEQUENCE {
        x MagnetometerType,
        y MagnetometerType,
        z MagnetometerType
}

MagnetometerType ::= REAL (0.0 .. 5.0)

MagnetorquerAxisType ::= SEQUENCE {
        x Boolean,
        y Boolean,
        z Boolean
}
-- etc
```

Listing 1. Data model of the ADC system types in ASN.1.

This kind of definition can be completed with encoding rules, in order to precisely define the protocol that is being used for telecommands. The ACN[3] notation (Mamais et al., 2012) has been used to this purpose. Listing 2 shows an example of ACN encoding for magnetometer values. Notice that the specification of the encoding, although low-level in nature, does not make any assumptions on the hardware platform that will be used in the implementation.

```
...
MagnetometerType[size 32, encoding IEEE754-1985-32]
...
```

Listing 2. Example of magnetometer values encoding rule.

4. DETAILED DESIGN AND IMPLEMENTATION

4.1 Platform-specific modelling

The *deployment view* models the implementation platform using the AADL platform components (Feiler, 2012). Its main purpose is to allocate the architectural components identified in the interface model to platform components. Since the UPMSat-2 OBC only has a processor board, this view is easily generated, as shown in figure 7.

The *concurrency view* is generated from the interface and deployment views. It models the concurrent and real-time behaviour of the system as a set of threads and data objects. Listing 3 shows an extract of the concurrency view in AADL, including the definition of the attitude controller as a periodic thread.

4.2 Code generation

The implementation code has been automatically generated from the model views. The implementation language is Ada 20005 (ISO, 2007), with the Ravenscar profile restrictions for tasking (Burns et al., 2004). The application code runs on a dedicated runtime, based on the PolyORB-HI middleware (Hugues et al., 2006) and the GNAT/ORK+ kernel for bare LEON computers (de la Puente et al., 2008).

[3] ASN.1 Control Notation

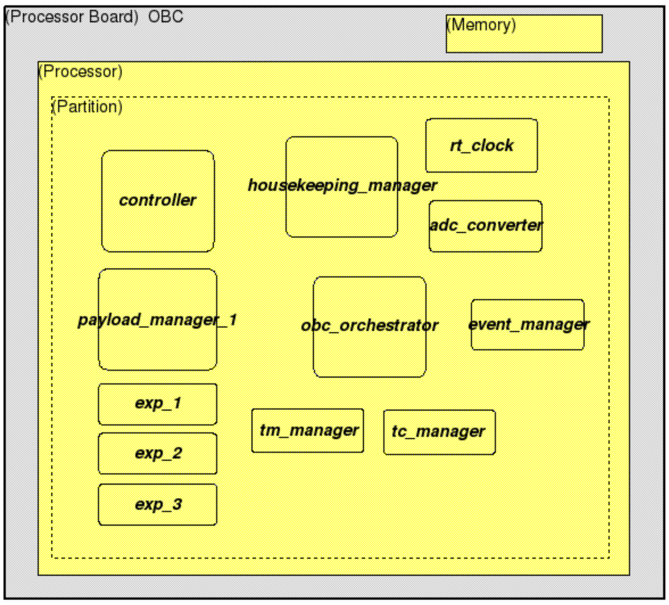

Fig. 7. Deployment view of UPMSat-2.

```
PROCESS IMPLEMENTATION ConcurrencyView_Async.others
SUBCOMPONENTS
   Controller_Cyclic_FV  : THREAD Controller_Cyclic_FV.others;
   ...
  -- etc
END ConcurrencyView_Async.others;

-----------------------
 Controller_Cyclic_FV
-----------------------

THREAD Controller_Cyclic_FV
FEATURES
  Magnetometer : REQUIRES DATA ACCESS Magnetometer;
  Magnetorquer : REQUIRES DATA ACCESS Magnetorquer;
END Controller_Cyclic_FV;

THREAD IMPLEMENTATION Controller_Cyclic_FV.others
CALLS {
  getMagnetometerValues :
     SUBPROGRAM Magnetometer.getMagnetometerValues;
  setMagnetorquers :
     SUBPROGRAM Magnetorquer.setMagnetorquers;
};
PROPERTIES
   Initialize_Entrypoint  => "Controller.executeControl";
  Dispatch_Protocol => Periodic;
  Period    => 1000 ms;
  Deadline => 100 ms;
END Controller_Cyclic_FV.others;
 ...
-- etc
```

Listing 3. Extract of concurrency view in AADL.

Skeletons of real-time tasks and shared data objects are generated from the concurrency view using the TASTE code generator. The functional code comes from the functional view models using the appropriate code generation tools. For example, the functional code of the attitude controller has been generated with the Simulink code generator. Listing 4 shows an extract of the Ada code generated for the attitude controller, and the import of the functional code generated from Simulink.

```
package body Controller is

  -----------------------------
  -- Required interface "getMagnetometerValues"
  -----------------------------
  procedure getMagnetometerValues(magnetometeraxis:
    access asn1sccMagnetometerAxisType);
  pragma import(C, getMagnetometerValues,
    "Controller_RI_getMagnetometerValues");

  ------------------------------
  -- Required interface "setMagnetorquers"
  ------------------------------
  procedure setMagnetorquers(magnetoqueraxis:
    access asn1sccMagnetorquerAxisType);
  pragma import(C, setMagnetorquers,
    "Controller_RI_setMagnetorquers");

  -----------------------------
  -- Import Simulink-generated control function
  -----------------------------
    procedure Control;
    pragma Import (C, Control, "Controller");

  -----------------------------
  -- main function
  -----------------------------
  procedure executeControl is

  Magnetometers : aliased asn1sccMagnetometerAxisType;
  Magnetorquers : aliased asn1sccMagnetorquerAxisType;

  pragma Export (C, Magnetometers, "control_U");
  pragma Export (C, Magnetorquers, "control_Y");

  begin
    getMagnetometerValues(Magnetometers);
    Control;
    setMagnetorquers(Magnetorquers)
  end executeControl;

end Controller;
```

Listing 4. Controller code skeleton with control function in Ada.

The functional skeleton code is automatically inserted into a run-time container that includes the required concurrent and communication entities in terms of Ada tasks.

5. VALIDATION APPROACH

Software validation is carried out at different levels of the model-based development process. Models are validated through simulation, when possible, or by inspection when not. For example, the design of the attitude controller has been validated, and its parameters adjusted, using the Simulink model in figure 6 above.

Alternatively, some properties of the system are validated using static analysis tools. In particular, the timing behaviour of the system is analysed using MAST, a real-time analysis tool (González Harbour et al., 2001), and RapiTime (Bernat et al., 2005), an execution-time analysis tool. Both tools have been integrated with the TASTE concurrency view Pérez et al. (2008) and together provide accurate estimates of worst-case response times of every real-time task, which can be used to validate real-time requirements.

The implementation code is validated by testing on an engineering version of the computer board, built with commercial hardware that is not protected against radiation

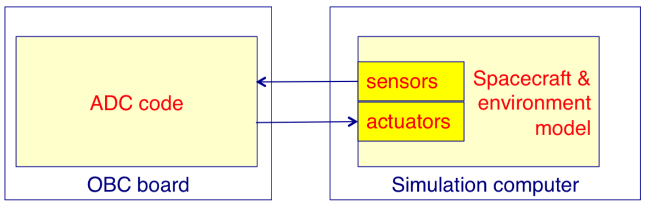

Fig. 8. ADCS test configuration.

in order to reduce costs. Embedded systems should be validated with respect to their real physical environment. However, the environment of space systems is not available for testing, and therefore it has to be simulated. We have used a dynamic hardware-in-the loop (HIL) facility to this purpose. Figure 8 shows the software validation structure for the ADCS software. The spacecraft dynamics and its environment are simulated on the simulation computer, while the control software runs on the engineering board. The configuration has also been used to estimate execution-time data that are fed back to the response-time analysis tools (Garrido et al., 2012).

6. CONCLUSION

The UPMSat-2 software, in particular the attitude control system, is an excellent testbed for Model-Driven Engineering of real-time control software. It is complex enough to raise non-trivial problems in software systems design, and on the other hand its size is not too large, enabling the development process to be manageable for a research activity.

MDE has allowed us to describe the software architecture and the main design details at an abstract level, as promised. The possibility to integrate validation tools with software modelling tools has proven to be especially interesting to check that the real-time behaviour of the design is adequate to the control requirements.

Model-based testing, as implemented in the HIL software validation facility, has also proved to be a valuable strategy, saving us significant time and facilitating fixing problems in the software at an early stage of development.

Plans for the near future include testing with real sensors and actuators, as well as radio communication links instead of the serial line connections between the satellite board and the software validation facility that are being currently used. The development of the on-board software system and the software validation facility is scheduled to be completed by mid 2014. The ground station software system is planned to be developed based on the latter.

ACKNOWLEDGEMENTS

The UPMSat2 project is led by the *Ignacio da Riva* Institute (IDR) of the Technical University of Madrid.[4] We would like to acknowledge the valuable collaboration with the IDR team, and especially Assal Farrahi, Javier Cubas and Ángel Sanz.

The OBC hardware is being built by TECNOBIT with the collaboration of the UPM group.

[4] `www.idr.upm.es`

REFERENCES

Albertos, P. and Mareels, I. (2010). *Feedback and Control for Everyone*. Springer.

Bernat, G., Burns, A., and Newby, M. (2005). Probabilistic timing analysis: An approach using copulas. *J. Embedded Comput.*, 1(2), 179–194.

Burns, A., Dobbing, B., and Vardanega, T. (2004). Guide for the use of the Ada Ravenscar profile in high integrity systems. *Ada Letters*, XXIV, 1–74. doi:http://doi.acm.org/10.1145/997119.997120.

de la Puente, J.A., Zamorano, J., Pulido, J.A., and Urueña, S. (2008). The ASSERT Virtual Machine: A predictable platform for real-time systems. In M.J. Chung and P. Misra (eds.), *Proceedings of the 17th IFAC World Congress*. IFAC-PapersOnLine.

de Ruiter, A., Damaren, C., and Forbes, J. (2012). *Spacecraft Dynamics and Control: An Introduction*. Wiley.

Farrahi, A., Cubas, J., and Sanz, A. (2013). Design of the attitude control subsystem of the UPMSat-2 satellite. Technical report, Instituto de Microgravedad Ignacio da Riva, UPM.

Feiler, P. (2012). *Architecture Analysis & Design Language — SAE AADL AS5506B*. SAE.

Gaisler Research (2012). *LEON3 - High-performance SPARC V8 32-bit Processor. GRLIB IP Core User's Manual*.

Garrido, J., Brosnan, D., de la Puente, J.A., Alonso, A., and Zamorano, J. (2012). Analysis of WCET in an experimental satellite software development. In T. Vardanega (ed.), *12th International Workshop on Worst-Case Execution Time Analysis*, volume 23 of *OpenAccess Series in Informatics (OASIcs)*, 81–90. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

González Harbour, M., Gutiérrez, J.J., Palencia, J.C., and Drake, J.M. (2001). MAST modeling and analysis suite for real time applications. In *Proceedings of 13th Euromicro Conference on Real-Time Systems*, 125–134. IEEE Computer Society Press, Delft, The Netherlands.

Hugues, J., Zalila, B., and Pautet, L. (2006). Middleware and tool suite for high integrity systems. In *Proceedings of RTSS-WiP'06*, 1–4. IEEE, Rio de Janeiro, Brazil.

ISO (2007). *Ada 2005 Annotated Reference Manual. ISO/IEC 8652:1995(E) with Technical Corrigendum 1 and Amendment 1.*

ITU (2008). *Abstract Syntax Notation One (ASN.1)*. Recommendations ITU-T X.680–683.

ITU (2011). *Specification and Design Language – Overview of SDL-2010*. Recommendation ITU-T Z.100.

Mamais, G., Tsiodras, T., Lesens, D., and Perrotin, M. (2012). An ASN.1 compiler for embedded/space systems. In *Embedded Real Time Software and Systems — ERTS 2012*.

Mathworks (2013). *Simulink*. URL `www.mathworks.com/products/simulink`.

Perrotin, M., Delange, J., Schiele, A., and Tsiodras, T. (2012). TASTE: A real-time software engineering toolchain overview, status, and future. In I. Ober and I. Ober (eds.), *SDL 2011: Integrating System and Software Modeling*, volume 7083 of *Lecture Notes in Computer Science*. Springer.

Pérez, H., Gutiérrez, J.J., Asensio, E., Zamorano, J., and de la Puente, J.A. (2008). Validating the end-to-end flow model to develop high-integrity distributed real-time systems. In F. Kordon and T. Vardanega (eds.), *Reliable Software Technologies — Ada-Europe 2011*, volume 5026 of *LNCS*. Springer.

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

Sidi, M.J. (1997). *Spacecraft Dynamics and Control: A Practical Engineering Approach*. Cambridge University Press.