Using Internet-based Technologies in a University Satellite Project

Juan A. de la Puente Jorge Garrido Emilio Salazar Juan Zamorano Alejandro Alonso

Abstract: UPMSat2 is an experimental micro-satellite mission that is being developed at UPM. The mission is intended to be used as a technological demonstrator as well as an educational platform. This paper focuses on the on-board computer system and the control functions implemented on it from an educational perspective. The project is used both in graduate courses and graduation projects as a basis for examples and a source of topics for laboratory work. Students work in designing and developing software for various satellite subsystems, and they use a set of web and internet-based technologies to access project documents and develop software in a collaborative work setup. The paper describes the use of internet tools by student teams and makes an assessment of the contribution of the tools to their work.

Keywords: Internet-Based Control Education; Web-Based Educational Environments; Problem-based Learning.

1. INTRODUCTION

UPMSat2 is an experimental satellite mission intended to be used as a technical demonstrator and an educational platform. It is being developed at UPM, the Technical University of Madrid, by a team including several research groups and industrial companies from Spain and other European countries. The project is led by IDR, [1] a research institute linked to the School of Aerospace Engineering. IDR is responsible for the overall design of the satellite, as well as the mechanical, thermal, power, and attitude control subsystems. The STRAST group, [2] to which the authors of this paper belong, is in charge of all on-board and ground software development (de la Puente et al., 2014b). The on-board computer (OBC) hardware is being built by TECNOBIT, [3] with the collaboration of the STRAST group.

The satellite has an external envelope of $0.5 \times 0.5 \times 0.6\,\mathrm{m}$, and a mass of approximately 50 kg, thus being in the micro-satellite range (figure 1). It is planned to be set on a sun-synchronous polar orbit (Fortescue et al., 2011) with an altitude of 600 km and a period of 97 min. Energy is provided by solar cell panels. Ground communications are based on a radio equipment operating in the VHF 400 MHz band. The expected launch date is in the second quarter of 2016.

The attitude control system is based on a simple control method based on magnetic field sensors and actuators (Cubas et al., 2015). Using a solar sensor and a

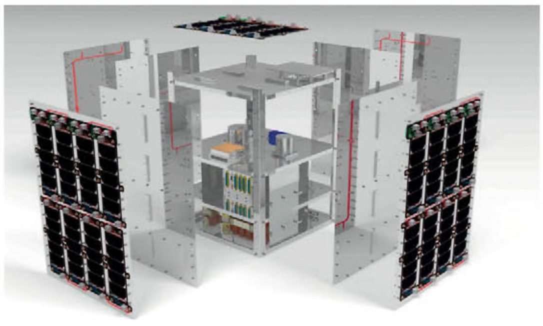

Fig. 1. UPMSat2 satellite platform

reaction wheel for attitude control is part of an experiment that will be carried out during the mission.

The main features of the on-board software system are described in section 2. The uses of the project in education, and the needs arising from a distributed educational framework, are discussed in section 3. The internet-based tools that have been used to fulfil such needs are presented in section 4. Finally, conclusions and hints for future work are included in section 5.

2. UPMSAT2 ON-BOARD SOFTWARE

2.1 Software architecture

The on-board software system performs control, communications, and monitoring functions that can be grouped as follows:

- Platform monitoring and control (housekeeping). Basic satellite data, such as voltages and temperatures at different points, are periodically sampled and checked in order to assess the status of the satellite.

- On-board data handling (OBDH). Telecommands (TC) received from the ground stations are decoded and executed, and telemetry (TM) messages with housekeeping and other data are sent to ground as required.
- Attitude determination and control (ADCS). The control algorithm is executed periodically, taking as inputs magnetometer readings, and generating magnetorquer commands in order to keep the attitude of the satellite aligned with the specified reference.
- Experiment management. The satellite includes some experiments, e.g. alternative attitude control methods and testing special devices, which are executed when commanded from ground.

According to the functional requirements, the software system has been decomposed into the four subsystems depicted in figure 2.

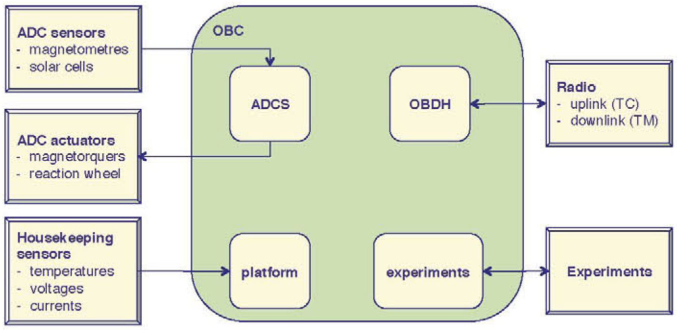

Fig. 2. Functional decomposition of the satellite software

2.2 Software design

A model-based engineering process (de la Puente et al., 2014a) has been used to design the satellite software. Figure 3 shows the main phases of the process, where a platform-independent model (PIM) is first built without taking into account the limitations of the hardware and operating system platform. A platform-specific model (PSM) is then derived using a description of the platform and the deployment of software subsystems onto it. This is a detailed design model from which source code can be automatically generated or manually produced, depending on the development requirements. Figure 4 shows the high-level design organization of the on-board software

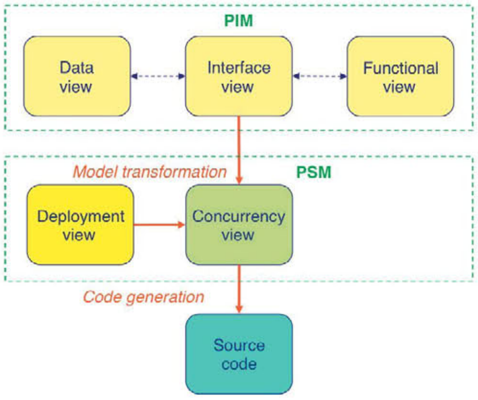

Fig. 3. Model-driven software process

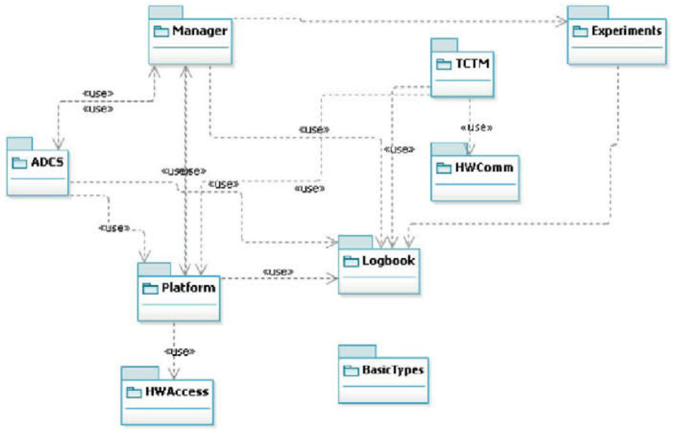

Fig. 4. High-level design structure

2.3 Source code development

The execution platform is a LEON3-based computer board (Gaisler, 2012) using a radiation-hardened FPGA. The Ravenscar profile of the Ada programming language (Taft et al., 2006) has been chosen for the implementation, due to it support for developing high-integrity real-time software and its wide use in space systems (Garrido et al., 2015). The software runs on the bare hardware, using the Open Ravenscar Real-Time Kernel (ORK) (Zamorano and Ruiz, 2002) which is part of the GNAT for LEON compilation system (Ruiz, 2005).

3. EDUCATIONAL ACTIVITIES

3.1 Overview

The UPMSat2 project has been used as a basis for educational activities at UPM. The students participating in them follow different curricula, oriented towards Telecommunications Engineering, Aerospace Engineering, and Computer Science, and work in two different campuses distant 18 km from each other. They can also work at home or other locations outside the campus. This kind of organization obviously requires support for distributed collaborative work, in order to enable the students and teachers to share project assets and participate in discussions irrespective or their physical location.

The activities in which the students can take part are introduced in the following paragraphs.

3.2 Regular undergraduate and graduate courses

Different parts of the project are used as examples in undergraduate and graduate courses at the Informatics School of UPM. For example, low-level code, including some VHDL code for the FPGA system, is used in a course on Industrial Informatics. Other software modules are used in master courses on Embedded Systems and on Real-Time Systems.

The main use of the project in regular courses has been in the UPM Master program in Space Technologies, specifically in the Software Engineering and the Real Time courses. This program has now been superseded by a new Master on Space Systems, in which all parts of the satellite, including mechanical, thermal, and electrical subsystems, are being used as basis for project based learning. The software subsystem, in particular, is used in two courses on Data Handling and Satellite Communications. The program is taught at the Aerospace Engineering School.

Students in these courses use the basic software system as a basis for alternative variants and sample code development. The ECSS E40, E-80, and Q-80 standards[4] are used to illustrate the software development process and the verification and validation requirements.

3.3 Special seminars

A seminar on the UPMSat2 software system is offered on a yearly basis as an elective in the Telecommunications Engineering undergraduate curriculum. The ECSS software engineering process and the basic software technologies used in the UPMSat2 are exposed to the students who participate in the course in order to improve their skills in embedded software engineering. The seminar is also used to select volunteers to do their graduation project with the real-time systems group.

3.4 Graduation projects

All undergraduate programs at UPM require a graduation project with an estimated workload of 300 to 360 hours (12 ECTS credits). A number of students have done their graduation projects with the STRAST group, working on the development of prototype modules of the UPMSat2 software system. The students work under the supervision of a member of the group, and must produce a project report as well as a working software product, complete with tests and documentation. Some students have also worked on master-level projects, which are typically longer (up to 30 ECTS credits) thus allowing for more complete developments.

4. INTERNET-BASED TOOLS

4.1 Requirements

The various activities described in the previous section are carried out by the students at different locations: classroom, laboratory, library, or even home. While a few activities may require the student's presence in a laboratory (e.g. testing with real hardware), most software development activities can be performed on personal computers, on a distributed basis. Therefore, in order to work efficiently in a distributed organization, an appropriate internet-based collaborative environment has to be set up.

The main needs that such an environment must fulfil are:

(1) Communications: in addition to basic email, teleconference facilities enabling remote meetings are needed.
(2) Document management: basic documentation, such as ECSS standards, and project-wide documents must be accessible to all the participants in the project in such a way that updates are immediately accessible. The documents must be under version control so that the users can track the changes and retrieve older versions whenever needed.
(3) Modelling tools. Using a model-driven engineering approach requires system modelling software and tools to be available to all the participants in the project. Sharing models should be possible through a central server, and the same requirements on updates and control version as for document management should be fulfilled.
(4) Code. Source code must available to all the participants in the project. Advanced version control mechanisms, including branching support for experimental developments and tagging of relevant versions, are also needed. The compilation chain tools must be shared by all the team members, and updating to new versions must be done in controlled ways in order to ensure consistency among software modules.

[4] European Cooperation for Space Standardization, `www.ecss.nl`.

There is no single tool that fulfils all the above needs, and therefore the collaborative environment had to be built using an assortment of internet-based tools. An additional requirement is that the tools should be open source or free software whenever possible, or at least be available at a low cost for universities.

The next section describes the architecture of the collaborative work environment and the choice of tools that are being used in the project.

4.2 Architecture and tools

Figure 5 shows the overall architecture of the UPMSat2 internet environment. It is a classical two-level architecture, with a public area that can ben accessed by anybody and a private area that is only accessible to the team members.

The public area is rooted at the STRAST UPMSat2 portal,[5] which is also linked to the project pages at IDR.[6] Its contents include general information about the project and the student-oriented activities, a list of publications and public documents, and a Twitter widget where the last tweets from the project account can be seen.

The private area is isolated from the internet by a firewall, and can only be accessed from the departamental network. Team members can get access from any other network by using a virtual private network (VPN) access. It contains collaborative work tools that cover the requirements stated in the previous section. The tools have been selected for functionality, availability, ease of use, and cost. A description of the main tools that are being used by the project team follows.

Communication tools. The Scopia videoconferencing system[7] is being used across all parts of the UPMSat2 project, with a central server run by IDR. The Scopia desktop client is freely available for the most common operating systems, and provides all the required functionality for holding project-wide remote meetings. Alternatively, Skype[8] has been used by smaller groups for short meetings because of its widespread availability and ease of use.

[5] `web.dit.upm.es/str/upmsat2`
[6] `www.idr.upm.es/tec_espacial/upmsat2-eng/01_UPMSAT2.html`
[7] `www.avaya.com`
[8] `www.skype.com`

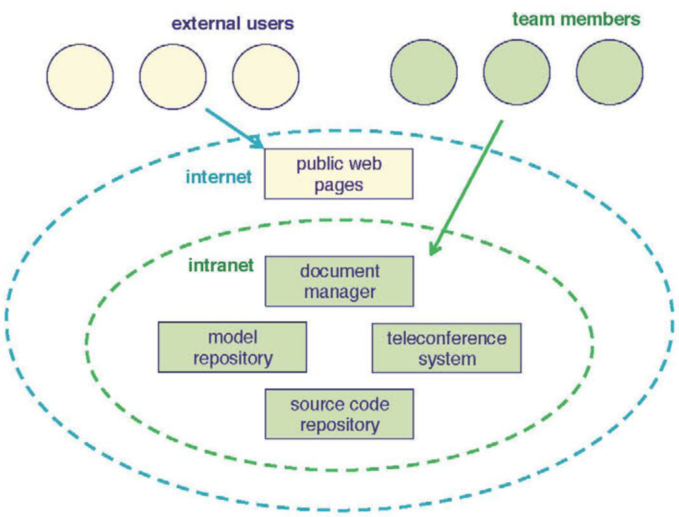

Fig. 5. UPMSat2 Internet environment

Document management. Although some public documents are stored on a plain web server in the public area, a more sophisticated kind of document manager is needed to keep track of changes and provide more elaborate access control. Simple solutions such as those provided by GoogleDrive[9] do not provide enough facilities for sharing and access control as needed by the project. Alfresco Community Edition[10] has been chosen instead for its availability as free software and its flexibility. Project documents are stored in a central server run by STRAST, and can be accessed by means of a web server.

Modelling tools. Two kinds of modelling tools are used in the project:

- *Dynamic modelling tools.* Simulink[11] is used to model the spacecrafts dynamics and to tune the attitude control parameters. Code for the attitude control algorithm is automatically generated from the Simulink model using the Simulink code generator.
- *Architectural modelling tools.* Software models developed in the model-driven software process (de la Puente et al., 2014a) are mostly coded using the AADL (Feiler, 2012) and ASN.1 (ITU, 2008) languages. The models are created and edited with the TASTE toolset[12] (Perrotin et al., 2012), which also uses other kinds of files, all in source text format.

The modelling tools have to be installed on the students' individual computers, but the models are shared in a central repository. In order to keep track of the changes and make the models, the same source code control tools as for the implementation code (see below) are used, thus collapsing the model and code repositories in figure 5 above into one single repository.

TASTE uses only textual languages for its model files, but Simulink model files are in binary formats. Therefore, in order to avoid possible conflicts when handling the models, the Simulink file types must be registered as binary files.

Source code control. Source code development tools include the GNAT compilation chain and other tools included in the GNAT Programming Studio[13] graphic environment. The tools are installed on the students' computers, but all the source code is managed through a central repository under version control.

The source code control tool that has been selected for the project is Subversion.[14] This system enables a team of software developers to work in a distributed environment. Source files are stored in a central repository which can be accessed through a variety of internet protocols, including `svn`, `ssh`, `https`, and `webdav`. The system has all needed features for collaborative block, including file locking, conflict detection, branching and merging, and flexible authorization schemes.

4.3 Laboratory facilities

Most student activities involve some amount of laboratory work. Most of it is related to the development of high-level software, and does not require any special facilities beyond ordinary PCs running GNU/Linux, with the GNAT compilation chain and its graphic environment as above described. The native x86/linux compilation chain is installed on the students and general programming lab computers for this purpose.

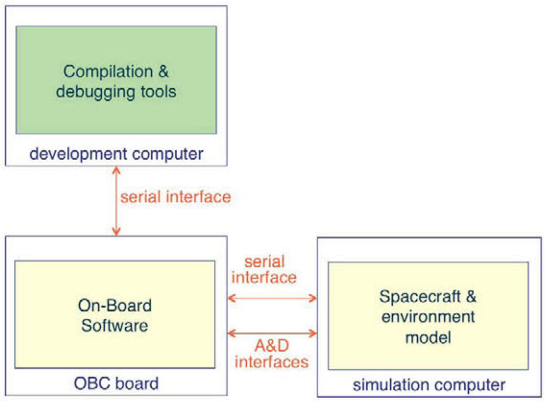

Fig. 6. Laboratory setup.

However, some activities, especially for students working on graduation projects or master theses, require access to the OBC hardware, including input/output devices and some sensors and actuators. In order to support this kind of activities, a specific laboratory has been set up including the following facilities:

- Development computer: a GNU/Linux workstation with a cross-compilation chain generating Leon3 executable code, including remote debugging software.
- An engineering model of the OBC based on an FPGA development board with a LEON processor, memory and other device cores.
- A simulation computer running MATLAB/Simulink with specific I/O boards: serial ports, analog outputs, and digital input and outputs.
- Electronic instruments such as oscilloscopes and multimeters.

[9] `www.google.com/drive/`
[10] `www.alfresco.com`
[11] `www.mathworks.com/products/simulink`
[12] `taste.tuxfamily.org`
[13] `www.adacore.com`
[14] `subversion.apache.org`

The simulation computer runs simulation models of the spacecraft dynamics and the orbital conditions in order to test the ADC software. It also runs lower-level simulation software for the radio equipment and other satellite hardware.

The development computer is connected to the internet and can be accessed via remote login. The students can develop software modules on their computers and then upload them to the laboratory for cross-compilation and loading on the OBC board. In this way, realistic hardware-in-the-loop testing can be carried out for most of the satellite software modules.

4.4 Usage

The above described internet tools have been extensively used throughout the project time line. Students have used the document repository to have easy access to specification and design documents, and the model tools to generate skeleton source code and explore changes in the models.

The source code control system has been the most useful tool for the students' work. They have been able to start new developments as code branches, without affecting the mainstream code. The student code can later be merged into the mainstream, if the teachers find it correct and complete, including a full set of unit tests, which are also stored in the repository. If several students work on the same code module, the system detects possible conflicts and helps resolving them. In this way they are encouraged to experiment and look at the code written by others, which contributes to enhancing their ability to write correct code and increasing their productivity.

5. CONCLUSIONS AND FUTURE WORK

Using internet-based tools in the UPMSat2 project has proved to be both a flexible and powerful approach to collaborative work, not only for professors and researchers, but also for students. Feedback from students who have worked on the project has been very positive, and they have stressed the fact that they have been able to get a taste of real work in industry even in an academic environment. The use of the system architecture, models, and implementation code in classroom examples has also be very fruitful, and has given the students attending the courses a more realistic view of the problems that can be found in an industrial environment.

The UPMSat2 software is now in its production phase, but we are still planning to use it as a basis for experimental developments and teaching activities.

ACKNOWLEDGEMENTS

We would like to acknowledge the work of the UPM students who have participated on the project: Pablo del Hoyo, Gonzalo Pérez-Tomé, Víctor Tomás, Verónica Gómez, Diana Marín, and Beatriz Lacruz.

We would also like to acknowledge the valuable collaboration with the IDR team.

REFERENCES

Cubas, J., Farrahi, A., and Pindado, S. (2015). Magnetic attitude control for satellites in polar or sun-synchronous orbits. *Journal of Guidance, Control, and Dynamics*, 1–12. doi:10.2514/1.G000751.

de la Puente, J.A., Garrido, J., Zamorano, J., and Alonso, A. (2014a). Model-driven design of real-time software for an experimental satellite. In E. Boje and X. Xia (eds.), *Proceedings of the 19th IFAC World Congress*, 1592–1598. IFAC-PapersOnLine.

de la Puente, J.A., Zamorano, J., Alonso, A., Garrido, J., Salazar, E., and de Miguel, M.A. (2014b). Experience in spacecraft on-board software development. *Ada User Journal*, 35(1), 55–60.

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

Fortescue, P., Swinerd, G., and Stark, J. (2011). *Spacecraft Systems Engineering*. Wiley, 4 edition.

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

Garrido, J., Zamorano, J., de la Puente, J.A., Alonso, A., and Salazar, E. (2015). Ada, the programming language of choice for the UPMSat-2 satellite. In *Data Systems in Aerospace — DASIA 2015*. Eurospace.

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

Perrotin, M., Delange, J., Schiele, A., and Tsiodras, T. (2012). TASTE: A real-time software engineering tool-chain 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.

Ruiz, J.F. (2005). GNAT Pro for on-board mission-critical space applications. In T. Vardanega and A. Wellings (eds.), *Reliable Software Technologies — Ada-Europe 2005*, volume 3555 of *LNCS*. Springer-Verlag.

Taft, S.T., Duff, R.A., Brukardt, R.L., Plöedereder, E., and Leroy, P. (eds.) (2006). *Ada 2005 Reference Manual. Language and Standard Libraries. International Standard ISO/IEC 8652:1995/Amd 1:2007.* Number 4348 in Lecture Notes in Computer Science. Springer-Verlag.

Zamorano, J. and Ruiz, J.F. (2002). GNAT/ORK: An open cross-development environment for embedded Ravenscar-Ada software. In E.F. Camacho, L. Basañez, and J.A. de la Puente (eds.), *Proceedings of the 15th IFAC World Congress*. IFAC-PapersOnLine.