On-board Software Reference Architecture (OSRA) Development Analysis

André Filipe Gonçalves Pereira
andre.g.pereira@ist.utl.pt

Instituto Superior Técnico, Universidade de Lisboa, Portugal

November 2016

Abstract

Spacecraft On-Board Software (OBSW) is continuously growing in size and complexity. The reduction in the development cost and schedule became thereby the leading concern within the space industry. Traditional software development approaches based on manual coding are no longer suitable to satisfy the actual needs. To tackle this problem a reference architecture was created in order to standardize the OBSW development process, together with various tools designed to implement complex systems by adopting a model based approach and to automatically generate executable application code from the implemented models. The recent VERICOCOS project, carried out by GMV, intends to create a complete toolchain including some of those tools and also to integrate state machines in the development process to perform the behavioural modelling of the systems. However there is a lot of room for the improvement and harmonization of such tools. This thesis analyses the full VERICOCOS toolchain, identifying errors and limitations, and presents various ways of improvement. In order to do that a complete mockup model based on the Mission Vehicle Manager (MVM) of the Intermediate Experimental Vehicle (IXV) is designed and implemented, both at avionics and software level. Additionally, application code is generated and the exercise to adapt it to a time and space partitioned execution platform is successfully carried out, taking a step further to the harmonization of the On-board Software Reference Architecture (OSRA) with the Integrated Modular Avionics (IMA) paradigm. The adapted code was ran in a simulator and a fully operational MVM system was achieved.
Keywords: Spacecraft On-Board Software, Reference Architecture, Model-Driven Engineering, VERICOCOS Project, Time and Space Partitioning

1. Introduction

Software (SW) stands as an efficient and very flexible way to implement functionalities on board a spacecraft. Thereby, the size and complexity of the spacecraft OBSW have been highly increasing in the past decades [1]. Improving efficiency of OBSW development methodologies while increasing functional complexity is a well known challenge and it is one of the keys of competitiveness of the space industry in the near future.

In fact, current practices of OBSW development can be made more efficient if, throughout the whole SW life cycle, generic functionality and SW architecture can be identified and reused. A feasible approach to handle this issue lies in the standardization of a reference architecture and the use of a component based SW engineering approach, which makes use of domain specific languages.

Being aware of the problem European Space Agency (ESA) has led various activities to improve platform SW development, being the standardization of the On-board Software Reference Architecture (OSRA) one of the ESA's core research activities, under the Component Oriented Development Techniques (COrDeT) studies [2]. As a result, a tool for the SW architecture design was created, commonly named OSRA Editor. Other studies were conducted to address the modelling of the avionics of the spacecraft. Namely the AAML ESA's study led by GMV [3], which intended to create a modelling language and respective editor, AAML Editor, to design the spacecraft avionics architecture. More recently, the VERICOCOS project [4] was initiated with the objective of fostering the use of state machines to represent SW behavior of ESA's operational projects using dedicated languages and tools. It aims at understanding how behaviour modelling languages can efficiently complement the OSRA and AAML design tools.

This thesis aims primarily at analysing the current VERICOCOS toolchain in order to identify pitfalls, ways of improvement and practicability, and secondarily at fostering the harmonization of OSRA with the Integrated Modular Avionics (IMA)

concept by adapting the code generated from the models to be deployed in a time and space partitioned platform. In order to perform such analysis, a mockup model of the Mission Vehicle Manager (MVM), one of the most relevant subsystems of IXV, was designed and implemented using both the OSRA Editor and the AAML Editor tools, exploring as many features and capabilities as possible. The OpenGeode tool was used to model the behaviour of the system, introducing the SW behavioural modelling through the use of state machine diagrams. Simultaneously the author was in charge of writing the VERICOCOS toolchain user manual [5], providing a set of instructions to operate with the various tools, create new projects from scratch, and to perform the behaviour modelling.

2. Background

This chapter provides background knowledge on the topics and concepts relevant to this study, being fundamental to a proper understanding of this work.

2.1. On-board Software Reference Architecture

According to the Software Engineering Handbook [6] a Reference Architecture (RA) is a single, agreed and common solution for the definition of the SW architecture of a set of SW systems whose domain of variation is identified. In practice, RA can be understood as a reusable system design pattern, which assigns the required functionalities of a complex system to a predefined set of composing components.

The RA is then used as a basis to develop standards for interface specification, enabling the development, by the industry, of building-blocks [2]. RAs facilitate standardization and interoperability of systems within a domain, since they can be design based on the same RA and therefore follow the same guidelines, standards and principles.

The success of a RA relies on four fundamental concepts:

- Separation of concerns: It is the development practice of breaking a computer program into distinct features that overlap in functionality as little as possible.
- Composability: It means the components keep their properties when assembled.
- Compositionality: It means the local properties of assembled components can provide a system global property.
- Correctness by construction: It is a SW development practice that fosters the early detection and removal of development errors in order to build safer, cheaper and more reliable SW. It a priori guarantees the semantic correctness of the user model by strong enforcement of metamodel constraints.

A typical example for a RA out of the space domain is the Automotive Open System Architecture (AUTOSAR), within the automotive industry.

The OSRA is then defined as a single, agreed, and common solution for the definition of the SW architecture of OBSW systems.

This architecture is sustained by fundamental principles that guide the SW design, namely the Component-Based Software Engineering (CBSE) and the Model-Driven Engineering as SW development methodologies that are covered in the following subsections, in addition to the principles of separation of concerns and correctness by construction already introduced.

2.2. Component-Based Software Engineering

In CBSE, the SW application is composed entirely of SW entities called components, which together with containers and connectors represent the SW entities of the component model.

In the context of OSRA, a component is a unit of reuse and it is assumed to have well defined interfaces, explicitly captured dependencies and also that it is purely functional, containing only sequential behaviour with no timing nor synchronisation. It provides a set of functional services exposed through an interface called provided interface and gathers required services through another interface called required interface.

Moreover, components are enveloped by containers, which are responsible for performing the Non-Funtional Properties (NFP) associated with tasking, synchronisation and timing, using the mechanisms offered by a given execution platform, which consists on the real-time operating system or kernel, the communication drivers and the board support package for a given HW platform [7].

The assembling of components into a system is performed connecting the required interface of one component to the provided interface of other component with a connector. This structure is depicted in figure 1.

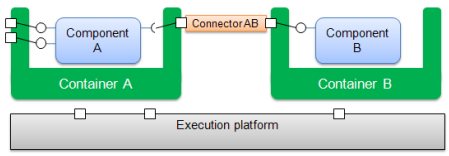

Figure 1: Component, container and connector scheme on top of the execution platform. [7]

By reusing building-blocks defined by a standard SW architecture, the SW development workload can be reduced while improving the functionality of the system and mitigating the risk of introducing errors in the code.

2.3. Model-Driven Engineering

MDE has emerged as a potential breakthrough in the development of complex systems, promising to reduce the cost of development and time-to-market mainly due to its automation capabilities, alleviating SW complexity by means of models. This allows the developer to abstract away the implementation details, which are not relevant to the problem domain, and provides an opportunity for an early verification of the system design [8].

According to Schmidt [9], a promising approach to deal with complexity of platforms is to develop MDE technologies that combine Domain-Specific Modelling Languages (DSML), which capture the specificities of a particular domain, formalizing its application structure, its behaviour and its requirements, and also transformation engines and generators, which aim to interpret the information contained in the model in order to produce various types of artifacts, such as more detailed models, source code, simulation inputs and even documentation.

Some details from this section were purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

2.4. Integrated Modular Avionics

The IMA concept emerged as an opposing concept to the federated architecture, and offered the possibility of integrating multiple functions into partitions of the same set of physical resources, allowing the aeronautical industry to manage the SW growth in functionality and in efficiency [10]. Partitioning keeps applications from inadvertently influencing each other by enforcing strict separation, segregating computing resources in space and time.

The architectural concept of IMA was transposed from the aeronautical to the space domain. This endeavour was taken by ESA and other agents of the European space industry. GMV had a major role on the outcome of some key activities, contributing in the definition of an IMA architecture tailored for space systems in the so called IMA-SP activity.

Each application can implement a system function by running several tasks managed by the Partition Operating System (POS), which is modified to operate along with the underlying hypervisor. In the context of IMA-SP, the selected POS was the space qualified version of the Real-Time Executive for Multiprocessor System (RTEMS). A graphical representation of an IMA-SP executive is depicted in Figure 2.

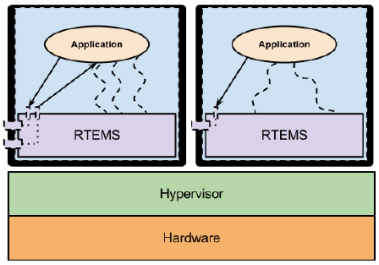

Figure 2: The IMA-SP executive composed by a hypervisor and two partitions, each one running its version of RTEMS [11]

The hypervisors are key players for the implementation of IMA-SP compliant architectures. GMV developed XKY hypervisor, also known as ARINC Interface in RTEMS (AIR) separation kernel, an ARINC 653 compliant time and space partitioned operating System that was originally based on RTEMS technology.

GMV also developed Hypervisor Emulator based AIR (HAIR), whose outcome is a performing and representative emulator for future CPUs (NGMP) implementing the Time and Space Partitioning (TSP) paradigm in multicore.

To support the definition and verification of ARINC 653 modules configuration GMV developed the Configuima, a graphical Eclipse based tool that allows the user to specify the most important aspects of an ARINC configuration, namely the number of partitions and their properties, the inter-partition communication, by means of partition ports and connections, and also the schedule policy. Afterwards the configuration can be exported to a .xml file. The current version of the tool can produce configuration files for GMV's AIR operating system, among others.

Some details from this section were purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

2.5. Intermediate Experimental Vehicle

The IXV is an ESA experimental suborbital reentry vehicle conceived as the step forward from the successful Atmospheric Re-entry Demonstrator (ARD) to demonstrate re-entry capabilities. The main goal was to design, develop and to perform an in-flight verification of an autonomous lifting and aerodynamically controlled re-entry system [12].

The IXV was successfully launched on 11th February, 2015 atop the Vega launcher as part of the VV04 mission being the first ever lifting body to perform full atmospheric re-entry from orbital speed.

Inside the IXV, the OBSW plays an important

role. It manages the system elements that are necessary to perform all mission modes, including launch, orbital, re-entry and descent-flight. It is needed to control the vehicle, perform experiments, monitor data, and provide data storage and telemetry.

One of the most remarkable subsystems of the IXV's OBSW is the Mission Vehicle Manager (MVM), which is the overall manager of the IXV and it is in charge of controlling and commanding the vehicle and its components executing all the functions required for the IXV to autonomously manage itself, including to perform the spacecraft modes transitions.

Some details from this section were purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

3. VERICOCOS Project

The VERICOCOS project aims at fostering the use of state machines to represent SW behaviour of ESA operational projects using dedicated languages and tools. It is desired a complete toolchain for the modelling of both SW and avionics of space projects, from the very basics of the architecture to the representation of the model behaviour with state machines, including the code and document generation capabilities, and also the possibility of performing analyses to check the model.

3.1. VERICOCOS Toolchain Architecture

The following tools were integrated in the VERICOCOS toolchain:

OpenGeode: This tool is able to create state machine diagrams using part of SDL-92 and SDL-2000 language definitions. OpenGeode has been selected as the tool for managing the state machines on all modelling levels, so the behaviour of the different models is represented as SDL state machines.

MSC Editor: This tool consists in a graphical editor of sequence diagrams following the Message Sequence Chart (MSC) specification. The MCS Editor has been selected as the tool for working with scenarios on the different modelling levels.

AAML Editor: The AAML Editor was selected as the tool for modelling the system and avionics levels of the spacecraft. The editor is based on a metamodel that supports the modelling process and that is used by the analyses in order to process the design and obtain the suitable data.

OSRAEditor: This tool is based on the Space Component Model (SCM), a domain specific metamodel that allows to describe the architecture of OBSW, the platform where the designed OBSW executes, and a set of non-functional characteristics. The OSRA Editor has been chosen to model the architectural design of the OBSW.

3.2. System and Avionics Modelling

The content from this section was purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

3.3. Software Modelling

The content from this section was purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

4. Implementation

The implementation of the avionics system of the mockup model started with the avionics functional definition since it represents the highest level of the system's definition.

Regarding the software modelling it shall start with the definition of the data types, exception types, interfaces, together with the respective operations and attributes, events, and data set specifications.

At this point the NFP of the system were specified, such as the NFP of each defined operation of a provided interface, or the sequence of such operations.

The next step was the definition of the HW in order to conduct the system deployment. The created HW diagram is similar to the one obtained in the avionics system implementation.

Finally all the instantiated components, devices and bindings were deployed onto the respective HW elements.

Some content from this section was purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

4.1. Behaviour Modelling

Once the OpenGeode is opened from the Component Implementation Diagram within the OSRA Editor a file containing the information about the data types and the interfaces is automatically loaded. This way the designer can directly use these elements to communicate and define new internal parameters.

In order to implement an operational MVM system the behaviour of each one of the ten components that compose the it was thoroughly implemented.

Together the ten diagrams represent the behaviour modelling of the complete system in terms of the OBSW. These diagrams were then used to generate the source code in C language.

The code generation engine transforms the state machine diagram in a program structured in a main function called "runTransition" that

essentially performs the transition between states. It takes as input the current state of the state machine, execute all steps described thenceforth and ends with the transmission of signals to other components that currently correspond to function calls of other components' implementations. These function calls will trigger the other components' "runTransition" according to the signal inputs and so on.

Some content from this section was purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

4.2. Code generation and Execution

To generate and execute the code two different approaches were identified. The first one is to run the Transformation Engines provided by the OSRA Editor tool. These engines would produce the code skeletons into which the C code generated from each component would be encapsulated to create the final application ready to be executed on a non TSP execution platform. The other approach, much more interesting but way more challenging and laborious, is to foster the integration of the generated C code in a TSP execution platform by adapting the code to a partitioned system following the IMA-SP paradigm and hence the ARINC 653 standard.

The second hypothesis presents a greater scientific relevance, taking a step forward towards the harmonization of OSRA with IMA-SP, thus being the one the author chose to investigate in order to obtain the final executables and run the application.

Henceforth the sequence of steps taken from the adaptation of the generated C code to the actual running of the application will be presented, together with the final obtained results.

Firstly Configuima was used to build an IMA configuration XML file compatible with ARINC 653 containing the information about the partitions, the inter-partition communication channels, namely ports and connections, and also the partitioning schedule. A one-to-one approach between the partitions and the component instances was adopted in order to benefit as much as possible from TSP. Furthermore, as simplifications or default approaches, the component generated code ran inside a single RTEMS task, and a basic schedule was defined giving an equal execution time window to all partitions since no computational model data was available.

From this point are described the manual steps taken to adapt the generated code from each one of the ten components to the TSP platform. These steps can be intended as the basis of a pseudo algorithm for a post processing tool development in future studies.

The steps performed were purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

Although the description is quite simple, this process was very time consuming and requested some engineering. Special care had been taken when handling the reception of queuing messages from different ports at same time. They were implemented sequentially with a non-blocking policy being polled in a fixed time interval. More complex but more efficient solutions are possible, as for example the creation of different tasks per queuing port.

The GMV's toolchains for the XKY hypervisor and for the HAIR emulator provided the full infrastructure to integrate the C files resultant from the previous adaptation. HAIR automatically generates the executable files from the existent information, one for each partition.

Finally the HAIR Emulator was ran. A screenshot of the resultant simulation is depicted in figure 3, where one can see the MVM application running in different partitions following the predefined schedule with the exchanged messages between partitions being displayed.

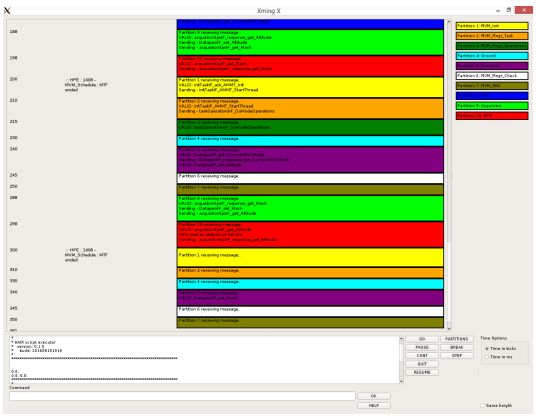

Figure 3: MVM application's simulation with ten partitions running in HAIR Emulator

The final result was a fully operational MVM application, representative of the IXV's MVM, corroborating the success of the implementation. If any timing constraint problem had occurred further work would have been required to optimize the partition schedule or set a specific functionality in an independent task to improve its availability.

5. Toolchain Analysis

This chapter derives from the experience I gained as a fresh developer of OBSW using the VERICOCOS toolchain, and also from the several conversations I had with more expert engineers including my

colleagues, my supervisor, and engineers from ESA during the meetings in which I have participated.

The list of identified errors and limitations while using the toolchain is very extensive and specific, thereby it is not included. Hereafter some suggestions to enhance the current toolchain will be presented and discussed.

The content from this section was purposefully removed to protect confidential data. In case you need further information please contact andp@gmv.com.

6. Conclusions

This thesis work was intended to take a step further to a faster and safer development of ever more complex OBSW within the space domain.

Firstly a literature review was conducted on the most important concepts behind the tools that support this work.

Secondly a detailed analysis of the IXV's OBSW was conducted in order to understand its behaviour. An MVM system based on the IXV's MVM was modelled using the VERICOCOS toolchain. During this process the full toolchain was analysed and the VERICOCOS User Manual was produced. Ultimately the code describing the behaviour of the system was automatically generated based on the implemented models. A detailed analysis has been made to the resultant code, and the exercise to adapt it to a time and space partitioned execution platform has been successfully achieved, taking a step further to the harmonization of OSRA with the IMA paradigm.

Resulting from the toolchain analysis various improvements were suggested.

The resultant executables were ran in the HAIR emulator, presenting a fully operational MVM system, which confirmed the success of the design and implementation of the system.

6.1. Future Work

It does exist in fact much more work that can be done in the future to improve the modelling tools for them to satisfy the actual and the short term needs on space OBSW development. Section 5 presents several suggestions to that end.

With more time to implement a model the author would suggest future researchers on this topic to model not only the OBSW of the MVM system but the whole IXV system's OBSW in order to explore non-explored functionalities and capabilities such as event handling or the integration of control algorithms for example.

Regarding specifically the harmonization of OSRA with the IMA concept some functionalities could be added in the future to the OSRA component model, namely the possibility to create partition ports and connections to create the communication channel between partitions for example. The OSRA computational model could be assessed in order to optimize the partition execution schedule in Configuima hence improving the performance of the MVM application.

Acknowledgements

The author would like to thank the Engineer Daniel Silveira for the opportunity to work on very interesting topics, strongly tied with industrial needs, at GMV, for his valuable help and guidance along the way, and also for the indispensable support with more technical issues. His immense knowledge and expertise were paramount to steer this work. The author would also like to express his gratitude to his supervisor, Professor Agostinho Fonseca, for his guidance and valuable help in the revision of this thesis. His suggestions and experience were fundamental to improve the quality of this dissertation.

References

[1] Daniel Dvorak. Final Report: NASA Study on Flight Software Complexity, May 2009.

[2] SAVOIR FAIRE working group. Space Onboard Software Reference Architecture. In *Proceedings of the Data Systems in Aerospace Conference* (DASIA 2010) , Budapest, Hungary, June 2010.

[3] Elena Alaña, Héctor Naranjo, Raúl Valencia, Alberto Medina, Christophe Honvault, Ana Rugina, Marco Panunzio, Brice Dellandrea, and Gerald Garcia. Avionics Architecture Modelling Language. In *Proceedings of the Data Systems in Aerospace Conference* (DASIA 2014), Warsaw, Poland, June 2014.

[4] Régis De Ferluc. *User Needs For Behavioural Modelling And Analysis Of The Tool Market.* Thales Alenia Space, September 2015. Reference: TN1.1 (VERICOCOS).

[5] André Pereira and Daniel Silveira. *VERICOCOS User Manual.* GMV, June 2016. Reference: UM1.1 (VERICOCOS).

[6] Space engineering: Software engineering handbook. Technical Report ECSS-E-HB-40A, European Cooperation for Space Standardization Secretariat, ESA-ESTEC, Noordwijk, The Netherlands, December 2013.

[7] Marco Panunzio and Tullio Vardanega. A component model for on-board software applications. In *2010 36th EUROMICRO Conference on Software Engineering and Advanced Applications.* Institute of Electrical & Electronics Engineers (IEEE), September 2010.

[8] Luigi Pomante, Sante Candia, and Emilio Incerto. A model-driven approach for the development of an IDE for spacecraft on-board software. In *2015 IEEE Aerospace Conference*. Institute of Electrical & Electronics Engineers (IEEE), March 2015.

[9] D.C. Schmidt. Guest Editor's Introduction: Model-Driven Engineering. *IEEE Computer Society*, 39(2):25–31, February 2006.

[10] James Windsor and Kjeld Hjortnaes. Time and space partitioning in spacecraft avionics. In *2009 Third IEEE International Conference on Space Mission Challenges for Information Technology*. Institute of Electrical & Electronics Engineers (IEEE), July 2009.

[11] Cláudio Silva. *Integrated Modular Avionics for Space Applications: Input/Output Module*. Master's thesis, Instituto Superior Técnico, October 2012.

[12] Massimo Succa, Ilario Boscolo, Alessandro Drocco, Giovanni Malucchi, and Stephane Dussy. IXV avionics architecture: Design, qualification and mission results. *Acta Astronautica*, January 2016.