Toward a Common Software Reference Architecture for CubeSats

Toward a Common Software Reference Architecture for CubeSats

Mr. L. M. Gagliardini[1], Dr. S. Corpino[1], Mr. A. G. Villa[2], Mr. R. Messineo[2], Dr. D. Fischer[3], Dr. M. Merri[3].

1) Politecnico d Torino, DIMEAS, Corso Duca degli Abruzzi, 24, Torino, 10129, Italy
2) Altec S.p.a., Torino, Italy
3) European Space Operations Centre (ESOC), Darmstadt, Germany

1 Abstract

Ever since the first CubeSat mission was launched, the concept and complexity of CubeSat Missions has evolved at a pace that current operational systems cannot match. In an increasingly dynamic space economy, where small businesses have become the norm, innovative solutions that abstract away complexity and increase autonomy are fundamental to reduce operational costs. It is within this frame that the current paper is presented. To address the need for a standardized software architecture of NewSpace companies, we first assess the European small satellite market needs through a survey with key players in the space sector. From this survey, we derive the high-level requirements, functionalities, and interfaces of a software architecture for CubeSats, the preferred platform due to its lower cost when compared to traditional platforms.

NewSpace ventures, ranging from private launch companies, small satellite operators, and bus manufacturers, have become the main driver of the space economy and are reinventing the traditional space industry supply chain. Among those that were recently born, only a few have successfully established themselves in the market, relying on venture capital and proprietary software solutions. We might think of Spire, recently acquired by Starlink with its huge nanosatellite constellations in LEO, as one of the most representative cases.

There are many reasons that can be attributed to the lack of market success or the delays that some companies experience getting their product to the market, but we can point out to the lack of a standardized, open-source software architectural reference for mission operations as one of the main obstacles they experience. Without access to proprietary software, these companies are forced to develop in-house knowledge and build and design their own software stacks, a lengthy and costly process. Moreover, this ad-hoc development further disincentives the development of a common architecture; if a company allocates resources to gain a market edge, it is not likely it is interested in losing it by making its software available to competitors.

Within this context, we aim at covering this gap by proposing a software architecture capable of satisfying a variety of stakeholder needs. To do so, we first characterize European small-sat operator needs through a survey involving both industry and governmental bodies, such as space agencies. Then, we address the technical challenges of defining an architecture capable of conciliating and satisfying competing operator needs. In this case, we aim not to define the specific architectural elements, but to derive the requirements that should be considered when developing such a system.

2 Market Polling

We started our survey in mid-2019 by submitting a questionnaire to several private companies and public bodies in the CubeSat market ("the entities"). Of these, twenty-four expressed their perspective on the existing standards and best practices applied to CubeSats.

To guide our work and gauge their interest, our investigation was centred around two topics: **if** the European standards satisfy their needs and **how** proposed solutions could bridge that gap, discussing how a reference software architecture devoted exclusively to CubeSats should be formalized. Particularly, we were concerned with how this architecture differed from their current products and whether they would be prone to adopt it in the future.
After the response period was closed, we reviewed their answers and characterized distributions: entity size, mission needs, and unique entity characteristics.
The first section intended to point out which existing standards, references, and best practices (e.g., CCSDS, ECSS standards) freely available to the European market the polled entities are familiar with, and which are used by respondents in their CubeSat projects. These were selected to cover the message transport issues both on-board and on-ground, the Space-to-Ground interface, and the orchestration of on-board activities, so to cover most of the crucial areas to the CubeSat operations. We also asked the companies to point out other references they apply in case these were not reported in the questionnaire. The questionnaire itself took into consideration the following standards, reported in table below *[5], [6], [7] , [8], [9], [10]:*

- CCSDS Space Packet Protocol
- CCSDS Mission Operations Services
- CCSDS Spacecraft Onboard Interface Services
- CCSDS Space Link Extension
- ECSS Packet Utilisation Standard
- SAVOIR-FAIR OBSW REFA

Beside the CCSDS and ECSS standards, the companies mentioned other references applied in their CubeSat projects, reported in *Table 1*.

ISO Standards [11]	Compass Stack
OMG Standards [12]	Other ECSS Standards (tailored and non-)
MISRA Coding Standards	Non-disclosed in-house developed reference
CubeSat Space Protocol (CSP) [13]	UNISEC Global

Table 1: Other adopted standards, references, best practices

According to our survey, 67% of the entities adopt the CCSDS Space Packet protocol (SPP) in their CubeSat project as a solution at the network layer, showing that it has become the de-facto standard even though most of them are not developing projects in collaboration with a public space agency. This provides a common baseline mission information exchange. At the same time, ECSS PUS and CCSDS MO services are adopted by respectively the 29% and the 42% of the entities as they provide a reference for managing data at application (operation) level. Of these entities, 43% and 50% respectively are contractors of the European Space Agency, which demands the use of these standards. Nevertheless, this suggests the CCSDS MO Services are more widely spread among CubeSat missions than ECSS PUS Services. Focusing on the second half of the questionnaire, we were interested in understanding how many companies are adopting 'a'/'their own' reference architecture (REFA) in their project development process and
to which extent. The majority (83%), of the polled companies
say they use a reference architecture developed in house or outsourced. Of those, 25% use an architecture that covers software interfaces while 35% apply a reference architecture for communication protocols. Indeed, they tend to take advantage of existing standards as a starting point for their in-house designs, despite not being mandatory (i.e., imposed by a contract).

The polled entities were then asked to deepen the software REFA topic about the possibility to adopt a common CubeSat reference architecture. We were interested in understanding how this choice would impact their business model, and what should the reference architecture encompass to add

value to them. We first evaluate at what level the reference should be designed (e.g., software/hardware interfaces/protocols), tackling technology gaps. The second inquiry is made to understand how the reference involving existing and innovative technologies should be proposed (e.g., as a paper-only design, in the form of Open-Source artefacts). What is immediately clear is that the request for an architecture covering on-board and on-ground functional components is, by far, the solution that best fits market needs. It is then followed by the request for a communication protocol level reference. The common opinion is that if a reference architecture were available, it would completely change their development process, allowing them to avoid incurring costly software development expenses. This leads us to the question: how far such a CubeSat REFA should go to provide a concrete benefit to your future business? According to the polling, 46% of the entities ask for an architecture involving the core functional components, to be presented as a non-mandatory reference. This aligns with what we reported before, showing companies and public entities taking advantage of existing technologies as a starting point for their internal design process, which in most cases results into an ad-hoc reference architecture. This further highlights the market need for a REFA and provides a line of research to further develop our study. Moreover, from the buyer's perspective, the presence of an open-source reference implementation would mean greater competitiveness and higher quality products, with vendors offering their own design, but still compliant with the reference architecture lest for implementation details.

The identification of the proper level of details in the architecture definition can be considered as the main driver of the design process. The aim at becoming a spread de-facto standard, implies the capability of reducing at most the unnecessary constraints coming from the design process while providing a solid baseline that serves as a common reference. The identification of such aspects whose definition is deemed relevant and those who can instead be accounted for tailoring processes is of paramount importance. The entire set on User Needs evolves around this hinge concept and underlines it in any treated aspect. It is worth to bear in mind that such needs do not constitute by any means a technical description of the architecture, and their lack of technicality would not allow this purpose. The statements reported in *Table 2* are only intended to resume the user needs as they resulted from the questionnaire. The translation to an applicable set of requirements is then described, in the present chapter, and the list of High-Level Requirements is reported in *Table 3*.

User Need ID	**User Need**
UN-001	The architecture shall take advantage of existing standards, references, and best practices.
UN-002	The architecture shall allow the usage/introduction of other standards.
UN-003	The architecture shall offer a baseline on top of which further solutions can be developed.
UN-004	The architecture shall avoid unnecessary constraints.
UN-005	The architecture shall be easily tailorable.
UN-006	The architecture shall balance the proper level of details

UN-007	The architecture shall allow vendors offering compliant and competing implementations of elements of the REFA.

Table 2: User Needs

3 Devised Architecture

In response to the market needs reported in *Table 2*, a list of high-level requirements was derived (*Table 3)*, describing the key drivers of a software reference architecture capable of best satisfying market needs. Having defined the requirements, we present the core components of REFA and a description of their interfaces and basic functionalities. Three different components are proposed and described: one On-Board component devoted to a specific on-board functionality; a Generic Component Model providing generic interfaces playing as a common software API for different on-board platforms and the functional components on-ground.

Req. ID	**Req. Title**	**High Level Requirement Text**	**Traceability**
SYS-0100	System's Domain	The architecture shall define functional on-board and on ground components	UN-001
SYS-0200	Implementation	The architecture shall not tie users to a specific technology/implementation	UN-002
SYS-0300	Component Functionalities	The architecture shall provide a framework of core functionalities	UN-003
SYS-0400	Adaptability	The architecture shall allow the user to replace the composing elements depending on the specific mission's needs	UN-004
SYS-0500	Inter Process Communication	The architecture shall not define the inter-process communication technology/implementation	UN-004
SYS-0600	Inter Component Communication	The functional components' interactions shall be expressed in terms of abstract interfaces	UN-005
SYS-0700	System Composability	The internal modification of any functional component shall be transparent to other components	UN-005
SYS-0800	System Compositionality	The composition of the core functional components shall ensure the overall system's functionality	UN-005
SYS-0900	Component Decomposition	Each functional component shall be internally decomposed into the different processes taking part to the activity	UN-006
SYS-1000	Architectural structure	Each system functionality shall be mapped to a different functional component	UN-006
SYS-1100	Market Model	The architecture shall not be mandatorily adopted.	UN-007
SYS-1200	Licencing	The architecture shall provide Open-Source Reference Implementations of the core elements of the REFA.	UN-007
SYS-1300	Documentation	The architecture shall be presented as a high-level descriptive standard with guidelines for its adoption and tailoring.	UN-007

Table 3: High-Level Requirements

The architecture shall represent a unified reference to which the companies can refer, as required by user needs. At first, we propose the architecture as a high-level design, i.e. System Requirements Document (SRS) and technical description of architectural components.

A brief list of existing technologies can be considered as a starting point for the definition of a software reference architecture targeting CubeSat-sized missions. We report some hereafter:

1) SAVOIR On-board Software Reference Architecture (OSRA)
2) The core Flight Software (cFS) developed by NASA
3) The CCSDS Mission Operations Services

The reported technologies cover quite well the higher levels of the architecture, from the application layer down to the transport layer. Among those, the cFS is probably the only providing a complete stack of functionalities. The cFS consists in an OS abstraction layer (OSAL) upon which the actual architecture is built. The OSAL provides a common reference layer for further developments. Right on top of this, a set of re-usable core components exist, providing basic functionalities, necessary for further, more complex components development and execution. Together with the core components, a set of re-usable applications built on top of the OSAL and taking advantage of the core components offer higher-level capabilities for orchestrating mission activities. It is possible to depict general pros and cons offered by the cFS. Among the pros we find the intuitive architectural structure, the segregation between functionalities, the fast deployment orientation thanks to self-standing library-oriented components and the strict reduction of functional overheads. On the cons we can observe that, beside the OSAL, little abstraction is provided with the core components and applications. This implies the need for mutual awareness when two applications are communicating, due to the lack of an internal service standardisation, intended in terms of operation's interaction patterns and data structures. This means that any operations provided by the core components requires the applications for an ad-hoc API integration.

From this point of view, the CCSDS Mission Operations services provide the abstraction needed for making the service user/provider transparent to each other. Each service provides a set of specific on-board functionalities, offered in terms of interaction patterns and data structures. The interaction patterns, namely Submit, Send, Request, Invoke and Publish Subscribe, are limited in number. Moreover, each service defines a data structure specific to the operation. In other terms, the interface is not imposed by the service provider but rather by the service itself. The pros of this concept are clear if we refer to a spread architecture where replaceable, plug-and-play components are subordinate w.r.t. the information framework itself. Among the cons we see the overhead introduced by the abstraction which guarantees such generalisation.

One more example is provided by the SAVOIR On-board Software Reference Architecture. OSRA provides an avionics-oriented component definition, expressed in terms of its interfaces. Each component interface is defined by set/get functions, component specific actions, event emitters and data emitters. Moreover, the functional properties of each component are segregated from the interface properties, by the introduction of containers. The containers provide the peer-to-peer components interaction capabilities for those functionalities offered by the component. Needless to say, the interface's interaction patterns and data structures are component specific.

A study for seizing the pros of several existing technologies was produced in 2015, with the objective of consolidating an architecture involving the CCSDS MO Services, the OSRA and the SOIS Services. The result of the study highlighted some commonalities between the different technologies and proposed a high-level description of a, so-called, consolidated architecture describing how the OSRA component interfaces could be mapped to existing MO Services and, below these, how the SOIS service could provide the component's access. Such consolidation has been done at application-level entities, by providing hypothesis of how components interfaces could be generalised into specific MO Services. Next step of the study was to harmonise such mapping. Nevertheless, an actual harmonisation would investigate the lower architectural levels of such architecture. Mapping a component to a service now implies to express such mapping in terms of data structures, data types, interaction patters and component's transactions.

The main body of this works defines therefore two aspects: as first, the provision of a methodology for harmonising the application-level components' interfaces, inherited by the OSRA, and the Service/Transport-level interfaces, as defined by MO. As second, the definition of application-level components' functionalities, providing a set of core components, whose definition takes advantage of the harmonisation methodology.

The harmonisation methodology focusses on the information exchange patterns, both on the service side and on the component side. While for the MO services a set of interaction patterns has been defined by the MAL[14], for the components, an entire interaction pattern set has to be defined, reflecting the interface properties of the OSRA components. The so-called *Transactions* define the input-output data type of a component interaction, depending on the different combination of input and output data (e.g. parameters, data sets, events, actions, etc). Any application-level component exposes interfaces in terms of such data types and, depending on the operation, a mapping to an interaction pattern is possible.

An adaptation component is responsible for making translating and segregating the Service-level concerns from the application-level environment and vice versa. By providing a strict definition of interfaces and rules for translation from one interaction pattern to a transaction, it is possible to automate the adaptation component generation.

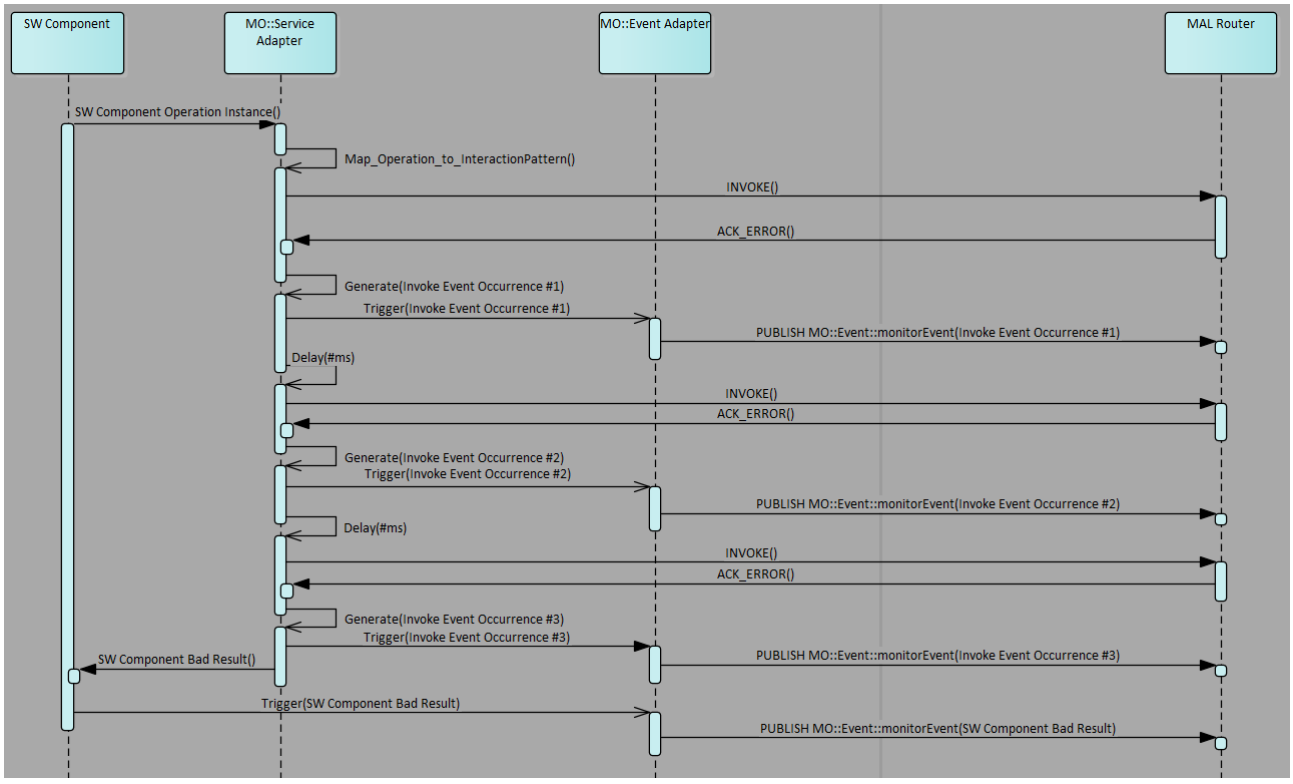

Figura 1: Component Action adaptation component handling errors coming from Invoke interaction pattern

The same translation is done for the data structures composing the interactions. It was chosen, for harmonisation purposes, to define component interface's data structure, taking the MO Services' data structure as a starting reference. MO Services provide a well-defined structure of data, derived from the MAL definition, for each Service's operation. Doing this requires a strict alignment with MO data structures definition in order to avoid custom-made harmonisation, therefore making the harmonisation process worthless.

4 Discussion

Addressing the need for an on-board software reference architecture oriented to the CubeSat market, implies considerations about the target entities of such harmonisation effort. A complete harmonisation process would tend to align all the architecture components to a same interface convention. Nevertheless, the aim for abstraction obstacles the process of minimising adaptation components among the architectural stacks. When talking about software architecture, every additional component/middleware and layer implies the need for maintenance activities. Long way has yet to come for a complete harmonisation in CubeSat software, which cannot avoid considering the overhead such harmonisation introduces.

5 Conclusions

The reported survey shows that a software reference architecture (REFA) devoted to CubeSats can be a game-changer for private and public companies in the small-sat market. Currently adopted solutions and new technologies in the CubeSat sector can be a baseline for an architecture that would be welcome by companies that are unable to implement it by themselves. Starting from this baseline, we propose a design that continues the harmonisation process as left over by the MOSS study. Abstraction implies a remarkable overhead in terms of system complexity, implementation cost and maintenance. If abstraction, as introduced by the MO, has to be imported also at component level, the discussion falls into the optimisation of the adaptation components/middleware. In conclusion, the good balancing between component abstraction and overhead reduction constitutes the game-changing parameter to be addressed by next development.

6 References

1. *OECD, Handbook on Measuring the Space Economy, 2012*
2. *D. Masters, T. Duly, S. Esterhuizen, V. G. Irisov, P. Jales, V. Nguyen, O. Nogués-Correig, L. Tan, T. Yuasa, M. Angling, Status and accomplishments of the Spire Earth observing nanosatellite constellation, Sensors, Systems, and Next-Generation Satellites XXIV, September 2020.*
3. *Crist R., CNET, https://www.cnet.com/home/internet/starlink-satellite-internet-explained, December 2021*
4. *Farrell, Luke J., "A Reference Architecture for Cubesat Development" (2020).*
5. Space Packet Protocol, CCSDS 133.0-B-2, Blue Book. Issue 2. June 2020.
6. Mission Operations Service Concept, CCSDS 520.0-G-3, Magenta Book, Issue 3, December 2010.
7. Spacecraft On-Board Interface Services, CCSDS 850.0-G-2, Green Book, Issue 2, December 2013.
8. Space Link Extension – Forward Space Packet Service Specification, CCSDS 911.3-B-3, Blue Book, Issue 3, August 2016.
9. Telemetry and Telecommand Packet Utilisation, ECSS-E-ST-70-41C, April 2016.
10. Savoir-Faire working group, “Savoir-Faire On-board software reference architecture”, TEC-SWE/09-289/AJ.
11. A. S. Portugal Dias, M. M. Ribeiro Magalhaes, ISO Standards and Audit, Organizational Auditing and Assurance in the Digital Age, January 2019.
12. M. E. Morales Trujillo, H. Oktaba, M. Piattini, The Making of an OMG Standard, Computer Standards and Interfaces, May 2015.
13. GOMspace, CubeSat Space Protocol (CSP), Network-Layer delivery protocol for CubeSats and embedded systems, June 2008
14. Mission Operations MAL Space Packet Transport Binding and Binary Encoding, CCSDS 524.1-B-1, Blue Book. Issue 1. August 2015
15. Boettiger, C. (2015). An introduction to Docker for reproducible research. ACM SIGOPS Operating Systems Review, 49(1), 71-79
16. D. Jaramillo, D. V. Nguyen and R. Smart, "Leveraging microservices architecture by using Docker technology," SoutheastCon 2016, 2016, pp. 1-5, doi: 10.1109/SECON.2016.7506647