

Model-Based CubeSat Flight-Software Architecture using a Docs-as-Code Approach

Sean Marquez* and Kevin Chiu†
Old Dominion University, Norfolk, VA, 23529

Sharan Asundi, Ph.D.‡
Old Dominion University, Norfolk, VA, 23529

This article presents a model-based systems engineering approach in conjunction with a docs-as-code approach for the SeaLion CubeSat mission. CubeSats are of increasing population and thus, there is a need to have easy and ready to use tools for new developers such as students to use. A model-based approach allows models to be the authoritative source of truth in place of documents and thus, allows for interconnectivity. The docs-as-code approach allows for local model building and allows code and documents to talk to one another. The Mach 30 Modeling Language was chosen as the basis for the SeaLion approach with the conjoined philosophies and its results are presented here with this very article being generated using Mach30 as well.

I. Nomenclature

<i>1U</i>	=	1-Unit
<i>3U</i>	=	3-Unit
<i>BOM</i>	=	Bill of Materials
<i>ConOps</i>	=	Concept of Operations
<i>DOF</i>	=	Distributed OSHW Framework
<i>M30ML</i>	=	Mach30 Modeling Language
<i>MBSE</i>	=	Model-Based Systems Engineering
<i>ODU</i>	=	Old Dominion University
<i>OML</i>	=	Ontological Modeling Language
<i>OSHW</i>	=	Open Source Hardware
<i>OWL2</i>	=	Web Ontology Language 2
<i>SWRL</i>	=	Semantic Web Rule Language
<i>USCGA</i>	=	United States Coast Guard Academy
<i>WFF</i>	=	Wallops Flight Facility

II. Introduction

THIS article presents the modeling language, tools, and technical approach used to facilitate the configuration management, design, specification, & implementation of the SeaLion mission architecture for the flight software using a model-based approach.

A model-based systems engineering (MBSE) approach to systems engineering is where models, as opposed to documents, serve as the authoritative source of truth for conducting systems engineering activities, such as the design, specification, analysis, verification & validation of a system [1]. The NASA handbook on systems engineering can be applied to CubeSat Mission design, in effort to facilitate a top-down design methodology from mission concept to specification of subsystem components, including flight software architecture [2].

*None

†None

‡Assistant Professor, Department of Mechanical & Aerospace Engineering, Old Dominion University

For the SeaLion Mission, the flight software team adopted an MBSE approach to flight software architecture using a document as code, also known as docs-as-code, approach [3]. This approach is where the same tools and methodologies for managing software are also used for configuration management of documentation. Applying both a MBSE and docs-as-code approach means that aspects of the mission architecture, such as stakeholder needs, user stories, and data structures pertaining to the CubeSat mission, are captured are within a model that is both human-readable and machine-queryable while minimizing configuration management overhead. A docs-as-code toolchain is also a file-based approach that allows for models to be persistent on local file systems without the additional use of cloud-based software or services.

III. SeaLion Mission

The SeaLion mission is a joint CubeSat mission between the Old Dominion University (ODU), Coast Guard Academy (USCGA), and the Air Force Institute of Technology (AFIT) to design and produce a 3-Unit (3U) CubeSat.

SeaLion consists of three payloads for on-orbit validation. ODU provided one payload while the USCGA and AFIT provided the other two payloads. SeaLion is planned to fly as a secondary payload on a Northrop Grumman Antares Rocket from Wallops Flight Facility (WFF), currently scheduled for March, 2023 [4].

A. CubeSats

The CubeSat, originating from California Polytechnic State University in 1999, are a standardized form of nanosatellites. Nanosatellites are satellites typically defined with a mass of less than 10 kg. CubeSats, also known as Cube Satellites, are defined by a standardized and modular architecture of 1-Unit (1U) cubes with dimensions of 10 cm X 10 cm X 10 cm with a maximum mass of 2.00 kg [5]. These units can be expanded upon such as, for example, a 3U CubeSat with standard dimensions of 10 cm X 10 cm X 34.05 cm and a maximum mass of 6.00 kg [5].

B. CubeSat Users

CubeSats were initially conceived as educational tools for space systems engineering [6]. Now, their roles have been expanded to not only just educational tools but for observation, technology demonstrations, and research, that were previous monopolized by much larger satellites, due to the low cost of production and launch of these satellites. As such, there has been increasing popularity for CubeSats as seen by the increasing number of launches in 1 since inception [7].

University groups especially are a large contributor in the overall number of launches of CubeSats yearly with university groups consistently maintaining plurality on total launches [7]. Thus, a need for readily available and easily learnable system engineering approaches and tools, for students especially, exists for those who are new to either CubeSat design and systems engineering.

IV. Goals

The goal of the SeaLion CubeSat flight software architecture was to capture the data structures and expected behaviors of the flight software, such that it can be unambiguously understood well enough to be implemented, as well as provide full traceability and rationale for architectural elements with minimal configuration management overhead [8]. The MBSE approach was of interest to the SeaLion CubeSat flight software team, as to yield the benefits of reducing ambiguity that usually comes with using informal language for specifying aspects of a system, as well as minimizing duplication of content that tends to accumulate in a document-based systems engineering approach.

The final approach had to achieve the following goals:

- Ensure templates only contain formatting data (this includes not storing boilerplate text in templates)
- Ensure models are the authoritative source of truth for all artifact content (e.g. artifact structure, meta-data, boilerplate, commentary, discussion, diagrams, tables, etc.)
- Models should persist on the local filesystem
- Documents should be in plaintext as to be compatible with modern distributed version control system (e.g., Git) and for ease of use.
- Documents should be able to sit alongside code and speak to one another.
- Documents should be model-based as to have a separation of concerns between content and formatting as well as be both human and machine-readable for querying and generating views.

Selection of a modeling language and modeling tool was taken into consideration in order to properly adopt a MBSE

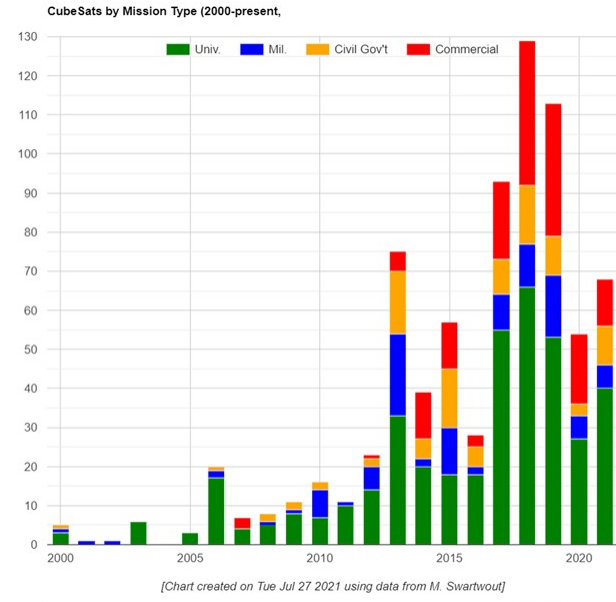


Fig. 1 Nanosatellite launch data provided by M. Swartwout [7].

approach. Other considerations include overhead incurred with training the team, as well as technical overhead with setting up and maintaining the modeling tools. The flight software team also decided to adopt a docs-as-code approach to further enhance a MBSE approach to achieve the listed goals.

A. Model-Based Systems Engineering

Traditional approaches use documents as their authoritative source of truth for conducting system engineering activities [1]. Information in a traditional systems engineering approach today is mostly captured informally, not authored based on a methodology, configuration managed in silo tools, adhocly and infrequently integrated, not easily traceable to its provenance, not properly configuration managed, not properly changed managed, and not effectively shared with stakeholders [9]. These documents often do not have a living relationship with other documents or to other corresponding elements; thus, changes to one document require manual changes to other documents [10].

In contrast, a model-based systems engineering approach supports capturing information in a highly structured modeling language, authored based on a methodology, configuration managed in a common tool, highly integrated, traceable to its provenance, and sharing with stakeholders. Models provide the following key advantages over document-based approaches [10]:

- Information is readily communicated and shared within the project.
- Changes are easily accommodated.
- Traceability is automated.

B. Docs As Code Approach

Docs-as-code refers to a philosophy that team members should be writing documentation with the same tools as code [3]. These tools may include version control (e.g., Git), issue trackers, code tools, etc.

This means following the same workflows as development teams, and being integrated in the product team. It enables a culture where writers and developers both feel ownership of documentation, and work together to make it as good as possible [3].

It was of interest to adopt a docs-as-code approach, as to yield the benefits of utilizing the same tools used to manage code, using version control tools (e.g., Git), for the configuration management of flight software architecture documentation, captured in a model-based approach [3]. It also allows for models to be persistent on a local file system without use of cloud-based services or software.

V. Mach30 Modelling Language

The flight software team conducted a trade study to downselect a suitable modeling language for the aforementioned goals. The languages considered with SysML V1, SysML V2, PlantUML, and the Mach 30 modelling language (M30ML). M30ML was chosen since it had extensible ontology language, supports both textual and graphical view generation, lightweight textual syntax, and relatively minimal overhead with modern doctools. The other candidates lacked in many regards compared to M30ML in these criteria and thus, M30ML was selected.

The SeaLion Architecture uses M30ML to specify references, stakeholder needs, user stories, and data structures for facilitating traceability of design decisions within the Mission Concept of Operations (ConOps) [11]. The SeaLion repository is also structured as a Distributed Open Source Hardware (OSHW) Framework (DOF)-component for defining the contents of the Mission ConOps as a collection of nested subcomponents, component interfaces, and component functions for generating bill of materials (BOMs), assembly instructions, and/or usage documentation.

This is all done in an effort to execute the project via a docs-as-code philosophy. A philosophy in which documentation is written with the same tools as code [3]. This allows for workflows between development teams to be more closely integrated.

A. Ontological Modeling Language

M30ML was developed using the Ontological Modeling Language (OML) as its basis. OML is a language that enables defining systems engineering vocabularies and using them to describe systems [12]. OML, inspired by Web Ontology Language 2 (OWL2) and the Semantic Web Rule Language (SWRL), is meant to be a more gentler and more disciplined method of aforementioned standard for use in systems engineering [12]. OML was created in part since OWL2 does not conform easily to individual modelling rules without tooling support; OML is a tool to improve speed of modeling and the quality of models while in a more concise and human-friendly high-level external representation [13].

B. Modeling Language, Tool, & Methodology

The M30ML is a YAML-based modeling language for defining a software architecture. Since YAML is a lightweight, highly-structured, human-readable, machine queryable, and line-oriented markup language, it was ideal for document generation use cases, as well as use with version control tools like Git. The following 2 showcases the simplicity of the YAML file.

```
1 id: 1
2 name: Ping Satellite
3 actor: Ground Station Operator
4 behavior: Ping satellite
5 rationale: Establish communication link with satellite
6 derivedFrom:
7 - "1-StakeholderNeeds/1.1-PrimaryMissionObjective-A1.yaml"
8 example: Ping the satellite in order to establish UHF communication link with Virginia ground station
```

Fig. 2 An example YAML file for a user story.

M30ML also had minimal technical overhead as it was compatible with modern doc tools such as asciidoctor & bibtex. M30ML also provided modeling elements familiar in agile software development, such as stakeholder needs, user stories, and data structures, with relationship elements for defining traceability between modeling elements [11].

C. File Structure and Generation

The M30ML file structure for architecture is simple and easy to use. Architecture is split into the four folders of references, stakeholder needs, user stories, and datastructures. Datastructures are derived from user stories which are subsequently derived from stakeholder needs with their respective references. The continued link between their respective YAML files allows for continuous path from which documents can be updated.

The following 3, PLACEHOLDER TABLE, and PLACEHOLDER TABLE are all automatically generated using the M30ML structure and injected directly into this article. Should any changes be made to the SeaLion architecture repository, the tables can be immediately regenerated for continuous updates.

D. Stakeholder Needs

The SeaLion project's methodology documentation uses M30ML based on YAML architecture modeling tools. The first step to build the architecture is to define the stakeholder needs. The two primary stakeholders for SeaLion are ODU

and CGA with their respective needs categorized on priority from primary to secondary to tertiary. These stakeholder needs are listed in the following Table PLACEHOLDER.

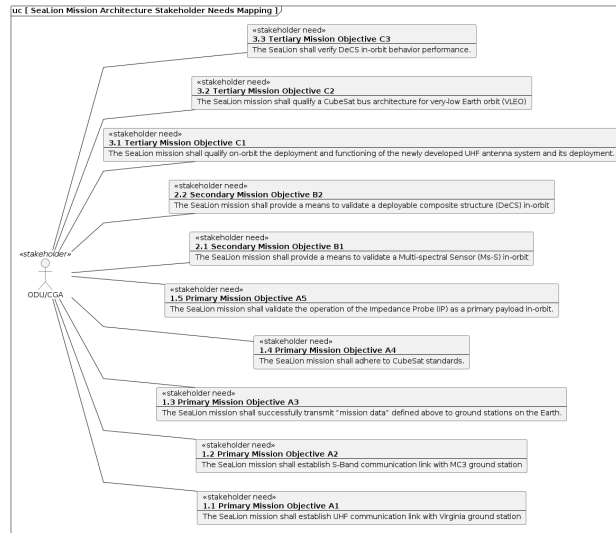


Fig. 3 a view describing mapping of stakeholders to stakeholder needs in UML

E. User Stories

The SeaLion Mission Architecture’s stakeholder needs are then used to identify a series of user stories which then lead to design decisions captured in data structure and activity definitions. These are created from the perspective of the ground station operator to define the tasks that need to be completed to satisfy the user stories. These user stories are listed in the following Table PLACEHOLDER.

PLACEHOLDER TABLE

F. Example Data Structure

An example data structure can be derived from user stories as presented in Table PLACEHOLDER.

PLACEHOLDER TABLE

VI. Conclusion

The approach and methodology presented in this article is done as an effort to reduce the painpoints associated with traditional systems engineering for the CubeSat developers. The ever growing number of CubeSat projects in existence demands a lightweight model-based approach that can be adopted for implementing flight software while minimizing configuration management overhead. Especially as many CubeSat developers will be new to the development environment. Docs-as-code combined with MBSE can be used as an approach to flight software for tight coupling between both software and documentation as well as providing a highly structured template to base future developments on for the CubeSat community [8]. Future actions include validating this approach as the flight software is created based on the model-based docs-as-code approach with the upcoming SeaLion launch as well as further expanding the potential users of M30ML.

References

- [1] Friedenthal, S., and Oster, C., *Architecting spacecraft with SysML*, 2017.
- [2] Asundi, S. A., and Fitz-Coy, N. G., “CubeSat mission design based on a systems engineering approach,” *2013 IEEE Aerospace Conference*, 2013, p. nil. <https://doi.org/10.1109/aero.2013.6496900>, URL <http://dx.doi.org/10.1109/AERO.2013.6496900>.
- [3] Holscher, E., “Docs as Code,” ??? URL <https://www.writethedocs.org/guide/docs-as-code/>.

- [4] Academy, O. D. U. . U. S. C. G., “Critical Design Review: Mission SeaLion - ODU/CGA 3U CubeSat,” , 2022.
- [5] *CubeSat Design Specification Rev. 14*, The CubeSat Program, Cal Poly SLO, ??? URL https://www.cubesat.org/s/CDS-REV14_1-2022-02-09.pdf.
- [6] Heidt, H., Puig-Suari, J., Moore, A. S., Nakasuka, S., and Twiggs, R. J., “CubeSat: A New Generation of Picosatellite for Education and Industry Low-Cost Space Experimentation,” 2000.
- [7] Swartwout, M., “CubeSat Database,” , ??? URL <https://sites.google.com/a/slu.edu/swartwout/cubesat-database>.
- [8] “SeaLion Mission Architecture,” ??? URL <https://github.com/ODU-CGA-CubeSat/sealion-mission-architecture>.
- [9] David Wagner, A. J. M. E. N. R. S. J., So Young Kim, “CAESAR Model-Based Approach to Harness Design,” *Proceedings of IEEE Aerospace Conference*, 2020.
- [10] Brown, B., “Model-based systems engineering: Revolution or evolution?” Tech. rep., 12 2011.
- [11] Simmons, J., “Mach30 Modeling Language,” ??? URL <https://github.com/Mach30/m30ml>.
- [12] Maged Elaasar, N. R., “Ontological Modeling Language: Origin and Rationale,” ??? URL <http://www.opencaesar.io/oml/>.
- [13] Jenkins, S., “Ontological Modeling Language 1.4,” ??? URL <http://www.opencaesar.io/imce/2021/06/19/OML-Origin-and-Rationale.html>.