www.adeepakpublishing.com

Kelly, S. et al. (2021): JoSS, Vol. 10, No. 3, pp. 1097–1108
(Peer-reviewed article available at www.jossonline.com)

www. JoSSonline.com

Using a CubeSat Reference Architecture for Accelerated Model Development and Analysis

Major Sean Kelly, Dr. David Jacques, Dr. Brad Ayres, Dr. Richard Cobb, Dr. Thomas Ford

Air Force Institute of Technology
Wright-Patterson AFB, OH US

Abstract

The domain of space systems engineering is currently in transition towards increased use of Model-Based Systems Engineering (MBSE) tools. This study examines the potential for reference architectures to assist with this transition by providing a starting point for engineering teams to build from, facilitating rapid design, prototyping, and requirement verification and validation. Specifically, this paper describes a current effort to create a CubeSat reference architecture for use in a university setting, aiming to shorten the development time and improve model and design quality for teams going through an accelerated design timeline. The present status of this CubeSat reference architecture is described, with two features are highlighted in greater detail: built-in analysis using parametric diagrams and stakeholder document generation. Future improvements for this reference architecture are also discussed.

1. Introduction

The CubeSat class of nanosatellites has lowered the barrier of entry to space and has rapidly gained popularity over recent years. The lower development cost, small form factor, and use of commercial off-the-shelf (COTS) components (Karvinen et al., 2015) make the CubeSat form factor an ideal platform for university teams, where budget and development time are extremely limited. In fact, many academic institutions have embraced this field for research, and have developed their own space programs (Pradhan and Cho, 2020). To successfully design a CubeSat system in a rapid cycle conducive to academic timelines, a reference architecture geared towards university CubeSat development would be helpful. A reference architecture would further speed up the development process by providing a template, capturing previous work and lessons learned from subject matter experts, and providing the framework to focus on the design, rather than the intricacies of modeling software. A reference architecture can also add functionality that student teams could use and improve over time, such as pre-built analysis functions and a library of components to

Corresponding Author: Major Sean Kelly - sean.kelly.33@spaceforce.mil
Publication History: Submitted – 02/15/21; Revision Accepted – 09/16/21; Published – 10/29/21

choose from. This paper will discuss the need for a CubeSat reference architecture and explore features of one developed at the Air Force Institute of Technology (AFIT) for their space program.

2. Model-Based Systems Engineering

The International Council on Systems Engineering (INCOSE) defines Systems Engineering as *"[a]n interdisciplinary approach and means to enable the realization of successful systems"* (Walden et al., 2015). The system, comprised of a collection of hardware, software, people, facilities, and procedures, begins as a theoretical concept in the eyes of users or stakeholders, and from that idea needs are defined, a system is developed and used operationally, and finally retired or disposed of (Buede and Miller, 2016). Systems Engineering is all about addressing this complete life cycle, and there are many strategies and techniques to accomplish this. The Department of Defense and NASA have traditionally used a linear, document-based approach, but they are currently transitioning to a Model-Based Systems Engineering (MBSE) approach.

Documents are the primary artifacts available to stakeholders (Delligatti, 2014) in the traditional approach, including requirement and traceability matrices, interface documents, concept of operation documents, and other unique documents in a wide variety of formats. As systems become more complex, the traditional document-based approach becomes challenging to maintain. Each document is manually generated, so file management and version control are problematic. For example, it is difficult to know for sure if a file is current or if it has been subsequently updated but is located on some other file system or storage drive. Furthermore, any changes in one document, drawing, etc., must be made in any other document that contains items affected by the change, or risk multiple versions of the same document being presented. This system is prone to errors, inconsistencies, and difficulties maintaining an accurate representation of the entire system. MBSE can help mitigate these concerns by consolidating the source of truth to one file. In MBSE, a system model represents the system and any information traditionally needed for documents can be found within this model. The model becomes the source of truth instead of the documentation. When in doubt, the model always has the most current information, making it easier to stay consistent. If the modeler updates a component or interface in one area, it will be updated throughout the system as appropriate. Acquisition program reviews may still require paper documents, but the necessary information for those can still be found within the system model. Note that teams must still take care to maintain version control for their system model so multiple versions are not being used, and cloud-based modeling makes this task much easier.

MBSE requires a modeling language, a modeling method, and a modeling tool (Delligatti, 2014). The CubeSat reference architecture described here was developed using the Systems Modeling Language and NoMagic's Cameo Systems Modeler (CSM) tool.

SysML is a standard modeling language that added systems engineering functionality to the Unified Modeling Language (UML) that has been used extensively in Software Engineering for decades (Delligatti, 2014). SysML provides a language, or the definitions and notations for nine different diagram types to describe a complex system, many of which will be used in this reference architecture. Model elements are expressed graphically through those diagrams, with SysML defining what those model elements are and how they are expressed. For example, a Block Definition Diagram (bdd) expresses system structure, and an Activity Diagram can show specific system behaviors. Within blocks, further detail can be expressed on an Internal Block Diagram (ibd). The modeling tool implements the SysML language, which allows for custom extensions if needed.

The modeling method is the specific methodology used to ensure that important design tasks have been accomplished, and provides the general guidance, processes, or steps for the system design. This paper will focus on the Object-Oriented Systems Engineering Method (OOSEM), but there are other popular methods, such as the Weilkiens System Modeling (SYSMOD) method (Weilkiens, 2016) and the IBM Telelogic Harmony-SE method (Hoffman, 2020).

OOSEM uses SysML in a top-down, model-based approach that leverages object-oriented concepts with

traditional systems engineering methods to architect more flexible and extensible systems and that can evolve with technology and changing requirements (Estefan, 2008). OOSEM was developed in part by Lockheed Martin Corporation as a method to capture and analyze requirements of complex systems, integrate with object-oriented software and hardware, and support system-level reuse and design evolution (Walden et al., 2015).

OOSEM includes the following steps in an iterative fashion (Object Management Group, 2011), all of which are incorporated into the reference architecture.

1. *Analyze Stakeholder Needs*: Capture the "as-is" system and mission enterprise and identify gaps or issues. The "as-is" depiction helps develop the "to-be" system, and the gaps or issues can help drive mission requirements for the new system. OOSEM frequently uses measures of effectiveness for the primary mission objectives identified in this step.
2. Define System Requirements: Once the "as-is" system is defined and produces Mission Requirements, the system is modeled as a "black box" in a Mission Enterprise model. For example, instead of going deep into subsystem-level detail on a CubeSat, the entire CubeSat will be a "black box" that interacts with ground stations, other satellites, and the environment. This "black box" model allows for system-level activity diagrams and use cases to show how the "to-be" system will support the mission enterprise. This step helps derive system-level functional, performance, and interface requirements.
3. *Define Logical Architecture*: A "logical" architecture is created that captures key functions in logical blocks, allowing for specific components to be chosen later in place of the logical depiction.
4. *Synthesize Candidate Allocated Architectures*: From the logical architecture, potential physical instantiations are created, using value properties and selected components. Each component at this stage is then traced to system requirements in table or matrix form.
5. *Optimize and Evaluate Alternatives*: Trade studies or other analysis is conducted at this step among the candidate architectures. Parametric diagrams within the model or integrating other tools can simulate system performance with the chosen components so alternative solutions can be compared.
6. *Validate and Verify System*: Once a candidate architecture has been chosen from the alternatives, the system needs to be validated and verified to ensure the requirements are being met and that stakeholder needs are satisfied. This step uses inspection, demonstration, analysis, and test activities to validate and verify the system.

Finally, the modeling tool is how the language, developer, and method work together. The modeling tool is a critical piece of software that maintains an underlying model of the system that can be used to display many different viewpoints or diagrams, depending on what is needed. The system model in a modeling tool is comprised of model elements and relationships between those elements, and from those, diagrams can be generated and displayed. When the source element or relationship is modified or deleted, that change gets carried out throughout the entire model, in all diagrams those elements or relationships appeared. The authors of this paper used the CSM tool from No Magic Inc., but other tools are available on the market to accomplish the same goals with different user interfaces and feature sets.

3. Reference Architectures

Complex systems require a well-thought-out architecture early on in the design process. The Department of Defense (DoD) recognized this issue for their complex systems, so they published the Department of Defense Architecture Framework (DoDAF) to establish “Enterprise-level Architectures” and “Solution Architectures” throughout the department. DoDAF defined an architecture as a “fundamental organization of a system embodied in its components, their relationships to each other and to its environment, and the principles governing its design and evolution over time (OASD/NII, 2010a).” This framework works well for

major Defense Acquisition Programs, but small university teams that turn over every academic cycle are not able to take full advantage of this framework. Reference architectures can help alleviate that problem by consolidating subject matter expertise and previous relevant architectures into digestible models that system designers can benefit from when creating a solution architecture (Cloutier et al., 2010). The DoD saw the benefits of reference architectures and put out a reference architecture description in 2010, describing them as "*an authoritative source of information about a specific subject area that guides and constrains the instantiations of multiple architectures and solutions*" (OASD/NII, 2010b). A reference architecture should be an "elaboration of company (enterprise) or consortium mission, vision, and strategy, facilitating a shared understanding about the current architecture and the vision on the future direction" (Cloutier et al., 2010). Furthermore, it should be continuously developed and improved over time as more teams use the architecture.

Finally, reference architectures should have at least the following elements (Cloutier et al., 2010):

1. *Strategic Purpose*: Goals, objectives, and a specific purpose or problem to be addressed;
2. *Principles*: High-level foundational statements of rules, culture, and values that drive technical positions and patterns;
3. *Technical Positions*: Technical guidance and standards that must be followed by solution architectures (maybe data vocabulary/ data model);
4. *Patterns (Templates)*: Generalized representations (e.g., Viewpoints, Views, Diagrams, Products, Artifacts) showing relationships between elements specified in the Technical Position; and
5. *Vocabulary*: acronyms, terms, definitions.

Reference architectures are being used in many industries, and at least one has been developed for CubeSats already. As part of the International Council on Systems Engineering (INCOSE) Space Systems Working Group (SSWG), Kaslow, Ayres et al. (2017) drafted a CubeSat Reference Model (CRM) to help promote and institutionalize the practice of MBSE for CubeSat development. Their CRM provides a reusable logical architecture for a generic CubeSat and provides a model to create a physical architecture from (Kaslow et al., 2017). The SSWG's CRM did not meet some specific needs for students, however. For example, the CRM is not designed to generate traditional documents for system level reviews. There is no easy way to generate a Concept of Operations document or Operational Requirements Document, for example, and that is a desire for a CubeSat reference architecture. Second, the CRM does not appear to have a component library or a generic, intuitive system that can be easily adapted by students new to MBSE. Finally, the CRM does not appear to have sufficiently detailed value properties for the system to be useful for detailed mission analysis using MATLAB and STK. Students in university courses must design down to a greater level of detail with many value properties for each subsystem in order to perform the required analysis and calculations. The CRM was quite useful though, in examining what subject matter experts deem important for a CubeSat model, and for their various subsystem internal block diagrams.

In summary, reference architectures can help systems engineers by providing a template, developed from years of experience, to aid in the systems engineering process. From the literature, it is clear that a reference architecture would be particularly useful for teams designing a CubeSat in a university setting; this paper will address that need.

4. Rapid Design Environment

Between the compressed schedule, the distraction of other courses and projects, and the lack of modeling experience for most students, designing a satellite in a short timeframe is a challenge. Using AFIT as an example, the space vehicle design sequence lasts just nine months. Students start with a Mission Capabilities Document (MCD), outlining the stakeholders' required capabilities and design constraints, and from there, they're expected to derive mission, system, and subsystem-level requirements, design the physical architecture, simulate that design, and ultimately test physical hardware to verify the requirements. Throughout the process, they build a system model

from scratch and use the model to create traditional stakeholder documents, such as a Concept of Operations, Space Vehicle Requirements Document, etc. They must also demonstrate traceability throughout the model, from that original MCD through the tiers of requirements and to the physical components themselves.

Clearly, developing brand new components within this short time period is not feasible, so COTS components or components developed by the university are used to accomplish their objectives. A component library within the reference architecture would aid this process, enabling teams to reuse or improve previous model elements if applicable. Students could copy and paste existing component blocks and simulate their system using those blocks to quickly assess mission feasibility with the chosen parts. Additionally, the primary mission stakeholders usually prefer traditional documents instead of a complex system model, so the model should aid in that process. In the past, students would copy and paste diagram images or transcribe requirement text into other tools, but this reference architecture will automate this process to rapidly generate deliverable documentation while avoiding the version control issues discussed previously.

In the university setting, the students generally come from a wide range of experience levels, with some having industry experience and others who have zero experience with satellites or with MBSE. Furthermore, students generally need to collaborate remotely due to their schedule demands. To address this, a cloud-based collaborative environment would be useful, and including examples and guidance will aid the less experienced team members. Template tables and diagrams should be provided, so students can focus more on the design choices instead of the details of model organization, structure, stereotypes, etc. In the end, a reference architecture should support rapid design, simulation, prototyping, and testing of a system by members of all experience levels.

5. Developing a CubeSat Reference Architecture

AFIT currently provides a template model to get students started with Stakeholder Analysis and developing a Concept of Operations, but support stops when students move on to future courses that build upon that foundation. Teams quickly diverge from using the model after the focus shifts from MBSE to design presentations and reports, so the need for a reference architecture to assist students becomes apparent.

The primary goal of this reference architecture is to encourage the use of MBSE throughout the entire design sequence, all the way through the testing of hardware, while incorporating faculty input to meet the needs of three courses. In this course series, students use the textbook "Space Mission Engineering: The New SMAD" (Wertz, 2011), so that was used as the primary source for equations, subsystem details, and mission activity descriptions.

The reference architecture opens with an overview diagram to show the organization of the model, as depicted in Figure 1. This figure shows the top-level package structure, including hyperlinks to additional organizational pages for each model section. This allows for intuitive navigation, instead of always digging into the directory structure (called the "containment tree" in CSM) to search for sections. The first package contains guidance for students, with a how-to guide and example diagrams for those that are new to MBSE.

The Component Library is a new feature inspired by an AFIT-developed Small Unmanned Aircraft System (SUAS) reference architecture (Jacques and Cox, 2019). This feature is still a work in progress, but the goal is to have a library of components for each subsystem that can be reused in new models for rapid prototyping. For example, if an engineer wants to quickly test how different antenna options affect the radio frequency link analysis, they can use the available antenna options in the component library, each having value properties that affect the calculations in the Analysis section of the model. Each subsystem has starter components contained within the respective packages, and the intent is for this library to be updated as new CubeSat designs are created using the reference architecture. After a team creates a working CubeSat model, those vetted components can then be imported to the Component Library for future reuse.

The third package (Generic CubeSat Model) is the core of the reference architecture. The "Generic

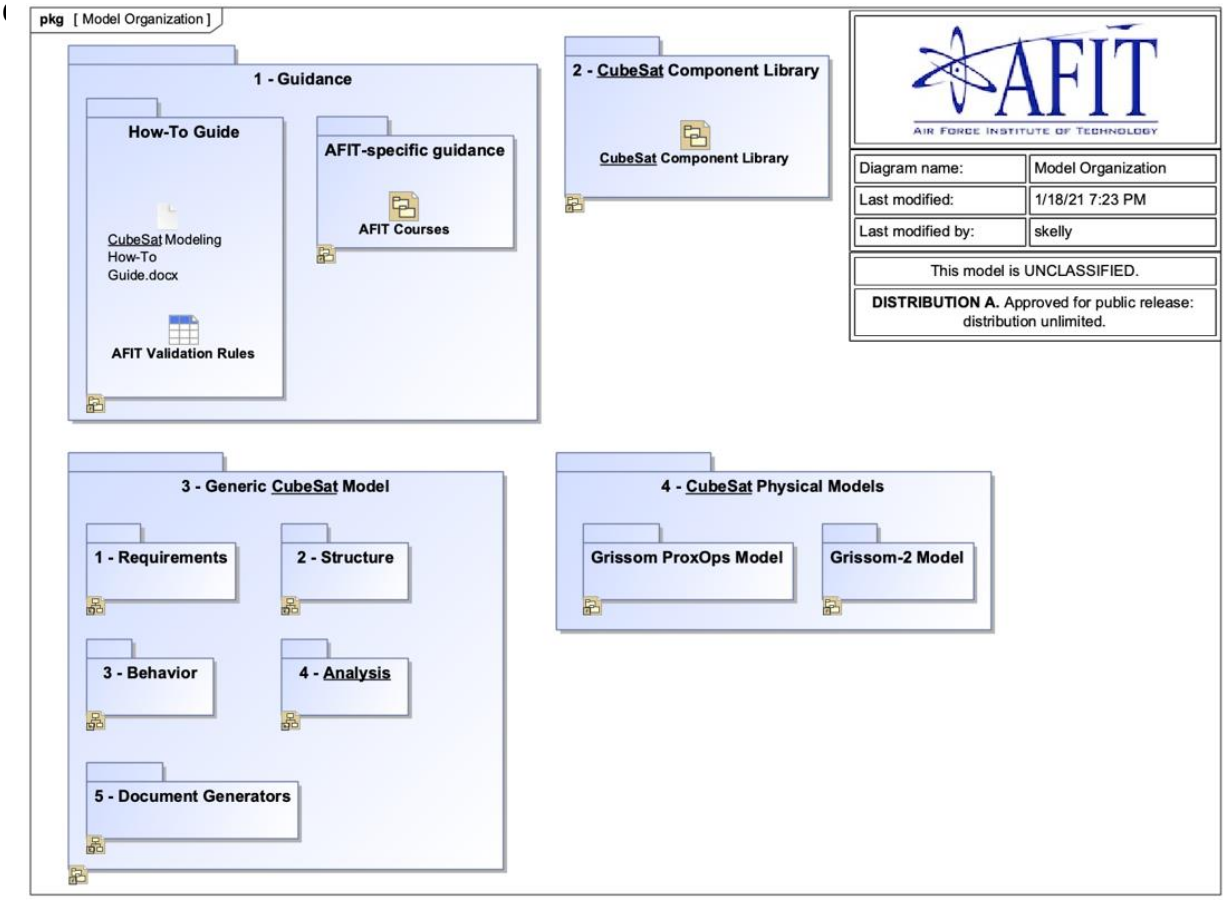

Figure 1. Top-level model organization.

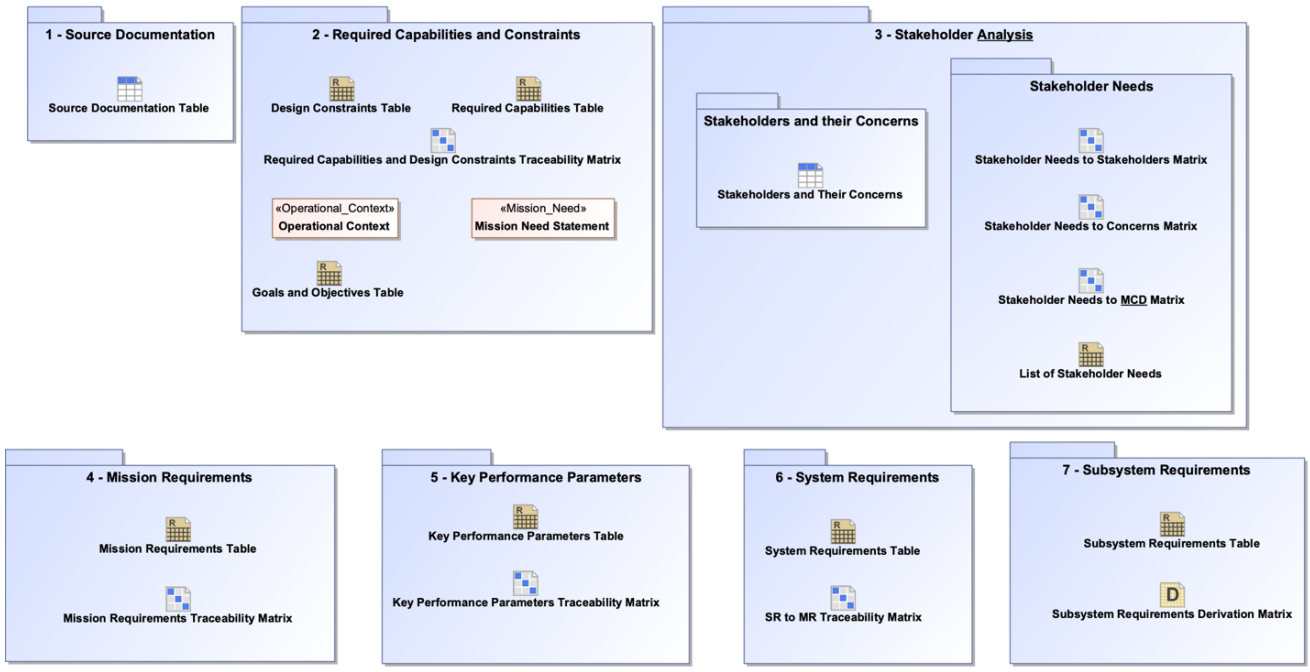

Figure 2. Requirements organization.

start. It has a pre-built, generic CubeSat model with diagrams, tables, and matrices provided with template data that is meant to be replaced by the design teams. It also contains the Document Generator tools that will be discussed later. Each package within the "Generic CubeSat Model" is hyperlinked to an informative diagram linking to all of the included tools and instructions for how to navigate them. An example diagram is shown in Figure 2, with links to all relevant requirement-related diagrams to fill out.

The CubeSat Physical Models package contains the various physical instantiations of the reference architecture. This could contain past projects to reference if needed, but for the purposes of the reference architecture development, it was used as the testbed to validate the model. This is also where teams will place their starting template to build from, keeping the generic CubeSat model for future use.

6. Use of CubeSat Reference Architecture for Requirements Verification and Validation

One of the key functions of this CubeSat reference architecture is the verification and validation section. Figure 3 shows the analysis portion of the reference architecture, which helps guide teams through the requirement verification and validation processes. Be-

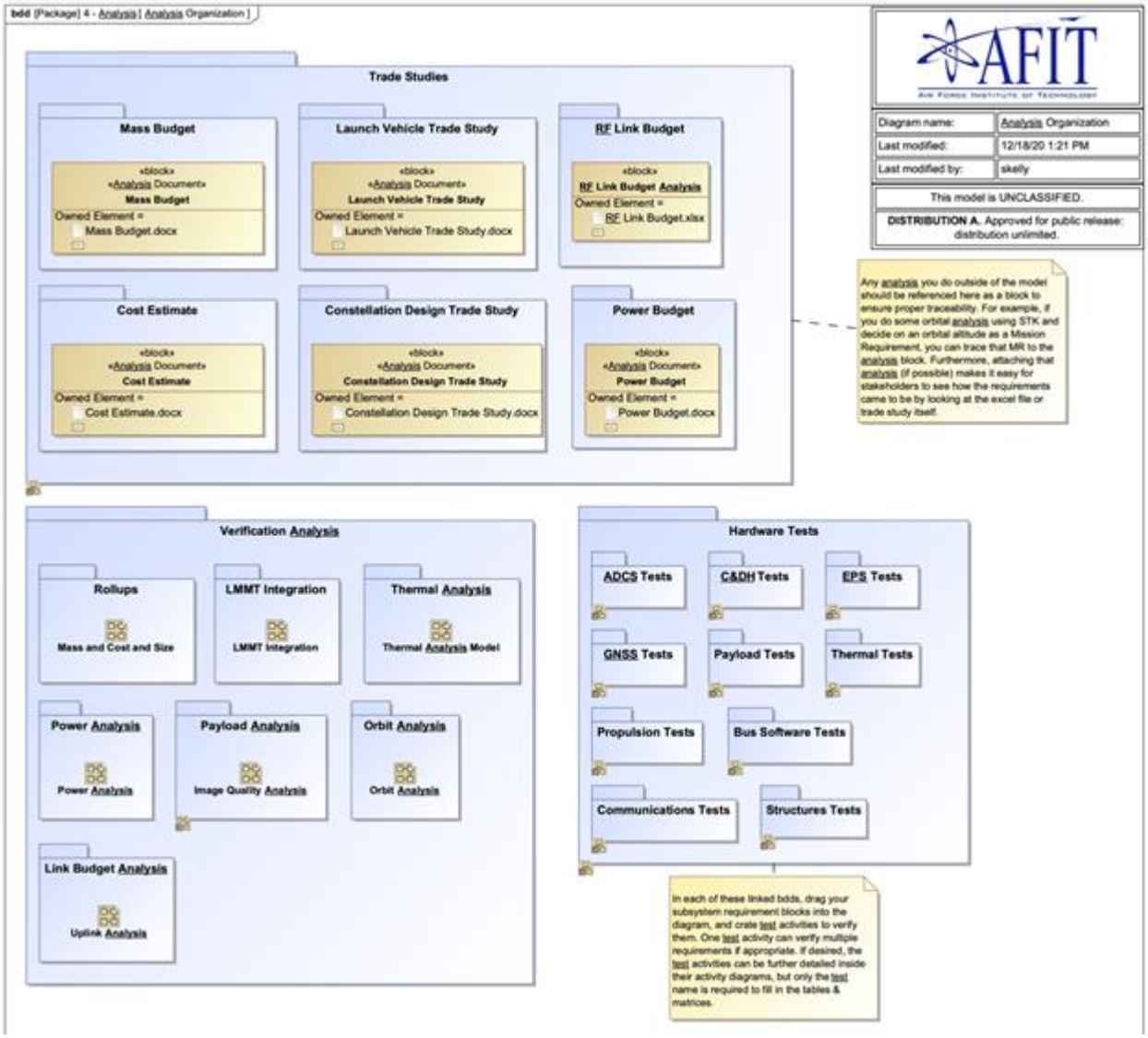

Figure 3. Analysis organization.

cause the CubeSat component blocks include predefined value properties, parametric diagrams could be created to perform a variety of calculations based off these values. An example of the included parametric analysis is associated with the thermal properties of the system, shown in Figure 4. This thermal analysis parametric diagram includes a constraint block with MATLAB code. This MATLAB code uses value properties from the "Thermal Subsystem" block (radiating area, emissivity, absorptivity, specific heat capacity, etc.), some parameters from the "Orbit" block (altitude, period, etc.), and any other value properties needed to perform the calculations. The MATLAB script then displays graphs that change automatically if the user swaps out a new thermal subsystem block, changes the altitude, or otherwise modifies the value properties. The constraint blocks can integrate engineering analysis into SysML modeling, and several analysis patterns were included in the reference architecture to assist future teams performing rapid analysis while keeping all work inside the system model.

The objective for the Analysis section of the reference architecture is to keep as much analysis contained within the model as possible, using the actual value properties to perform the calculations. Instead of moving values to other tools, the analysis calculations are kept within the model. Additional functionality can be added as well, depending on the requirements. For example, Figure 5 shows how a requirement for a Near Infrared (NIR) Ground Sample Distance (GSD) of less than 4 m could be tested using the same methodology. In this parametric diagram, the NIR GSD is calculated based off the imager's value properties and the CubeSat's altitude. This result is compared to the requirement and will automatically flag the result as green or red, depending on if it meets the requirement's constraint block or not, as shown in Figure 6. In this example, an engineer could tweak the design variables

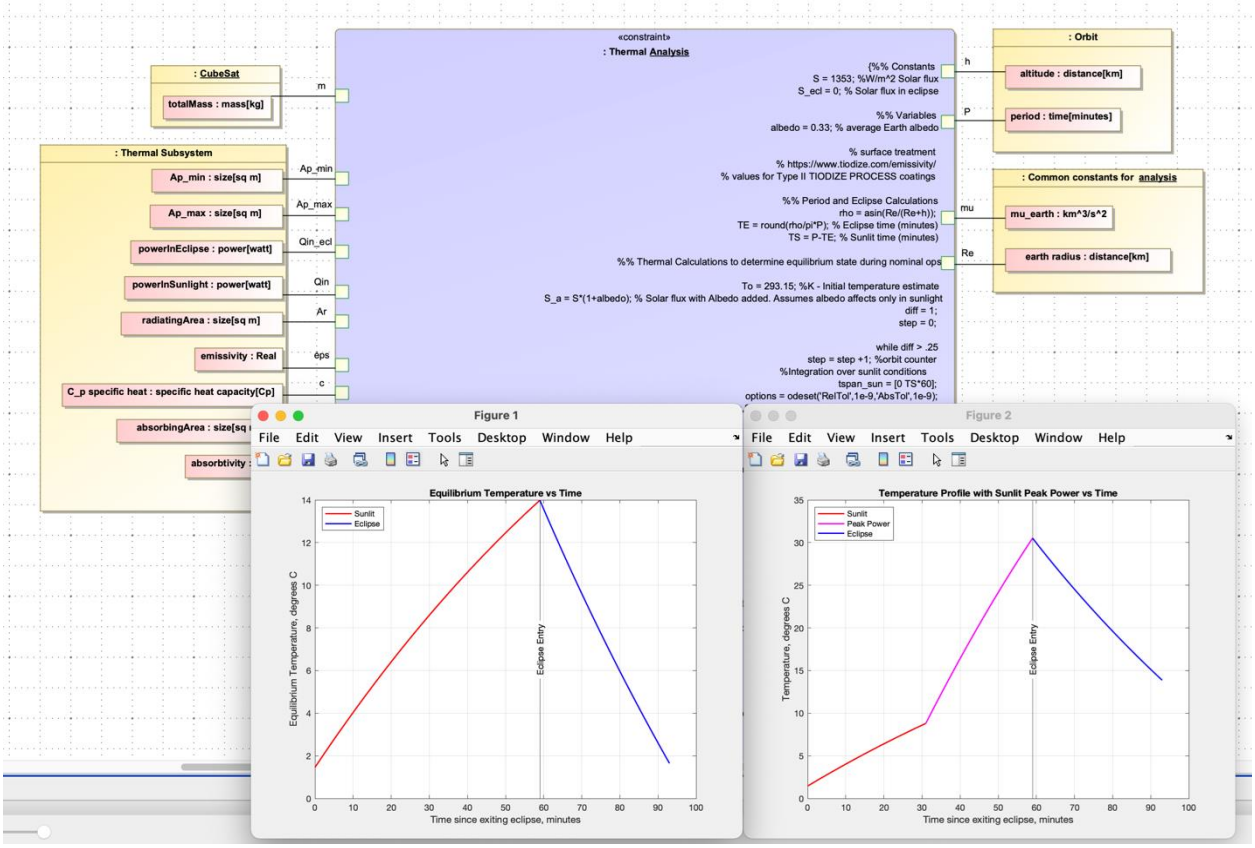

Figure 4. Thermal analysis.

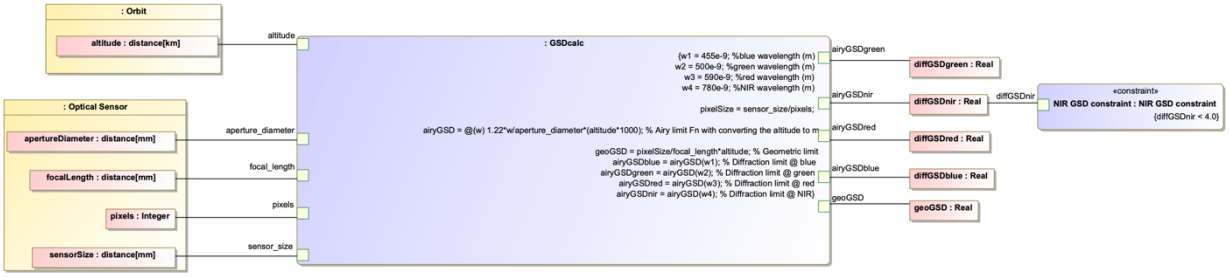

Figure 5. Image quality parametric diagram.

Name	Value
Image Quality Analysis	Image Quality Analysis@62a1c657
diffGSDblue : Real	1.8503
diffGSDgreen : Real	2.0333
diffGSDnir : Real	3.1720
diffGSDred : Real	2.3993
geoGSD : Real	0.0018
: Optical Sensor	Optical Sensor@18638d7d
: Orbit	Orbit@15816cce
: GSDcalc {w1 = 455e-9; %blue wavelength (m)w2...	GSDcalc@19a1763b
NIR GSD constraint : NIR GSD constraint {diffGSDnir...	NIR GSD constraint@58591fd7
diffGSDnir	3.1720

Figure 6. Image quality results.

and instantly see how the GSD is affected, which would be very useful in the early stages of design.

In addition to the parametric diagrams, teams will need to perform hardware tests in the lab, and the reference architecture accounts for that, as well. There is a package for defining hardware tests for each subsystem with features to help keep everything organized. Figure 7 shows the testing diagram for the Electrical Power Subsystem (EPS). The user can access the applicable requirements in the linked subsystem requirement table, create test activities to verify requirements, and include descriptions of each test in the included test description table. The linked subsystem requirement tables are automatically generated and also include color coding to highlight testing status. As tests are completed, the user can choose a verification status from a dropdown menu (such as Requirement Verified, Test Not Completed, Testing in Progress, etc.) to keep track of the test campaign in one standardized location. Each subsystem has a placeholder for test data and results within the test tables, so as teams conduct real-world testing, that data can be accessed from the model as well. In the future, more work can be done in this area to create a centralized test data repository and more detailed test activity diagram examples.

7. Use of CubeSat Reference Architecture for Generating Traditional Documentation

This reference architecture includes a polished document generator to create traditional documents for stakeholders. CSM can use Apache's Velocity Template Language (VTL) to export model elements into external tools such as Microsoft Word and Microsoft PowerPoint, so by including Microsoft Word file templates with tailored VTL code, documents can be automatically generated as needed as the model is updated. Even narrative text, such as introduction paragraphs or other lead-in text, can be included within the model as notes. The reference architecture includes

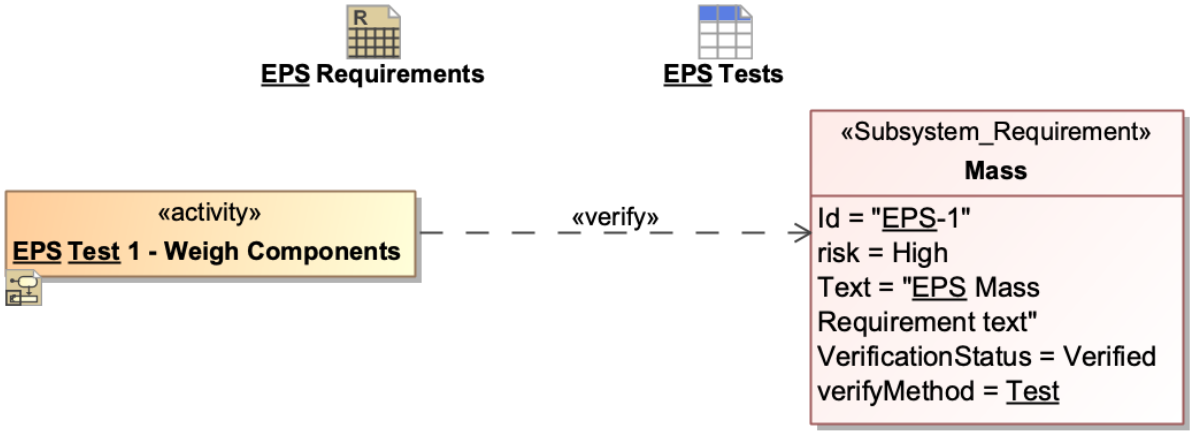

Figure 7. EPS tests.

document generators for a Stakeholder Analysis Report (SAR), Concept of Operations (CONOPS), Mission Requirements Document (MRD), Operational Requirements Document (ORD), Space Vehicle Requirements Document (SRD), Mission Capabilities Document (MCD), and a document to help start Test Plans. The templates were based off of AFIT course requirements, but they may be tailored if different sections are required, or if the order of sections needs to be modified. Additionally, the reference architecture includes a template for a master document, which exports all model elements that the team generated in an intuitive and visual format. This is useful as it contains all code that a user may wish to use if they want to create a new document template that was not provided. The generic model document can be used as a template to build custom documents from. The generic template is commented so users can know how it works and what to copy for a new document, as this template language is not well explained in the software user's manual.

The goal of these document generators is to encourage the use of the model throughout the design and build process. Historically, teams would copy and paste model elements into reports and transcribe requirement text into Microsoft Word or PowerPoint tables for reviews or presentations. This led to version control issues, such as requirement text being updated in the PowerPoint table but not in the underlying model. By keeping everything entirely within the model, these document generators take most of the manual work out of the process. Teams can focus on ensuring the model is accurate instead of needing to cross-check every diagram each time a change is made.

8. Mission Modeling on the AFIT CubeSat Bus

AFIT's CubeSat reference architecture is not only designed for classroom projects. AFIT has their own space program, and this reference architecture is intended to assist with its mission modeling. AFIT uses a 6U-sized bus called the "Grissom Bus" for its upcoming missions, and this reference architecture has that bus included in the component library along with other payloads and subsystem components that AFIT has in stock. As future Grissom-based CubeSats are designed at AFIT, this reference architecture can be used to rapidly prototype design configurations using the available components and the built-in analysis tools. Two upcoming missions, Grissom-1 and Grissom-P, have already been modeled using this reference architecture as test cases. Their requirements and structure were created, and stakeholder documentation was generated to ensure the process and tools worked for other models. Additional research is being done using this baseline reference architecture to easily simulate multiple AFIT payloads using the Grissom bus, and it all relies on the built-in structure and value properties associated with components. The goal is to be able to quickly test multiple design configurations using the same bus. For example, two payloads on the

same bus may have conflicting pointing requirements or a combined power draw that exceeds the power budget requirement, and the simulations should highlight these conflicts. As missions such as the Grissom-based missions are fully modeled in this architecture, the analysis tools can be used to model various mission phases or activities. For example, the relevant orbital parameters can be used to automatically create an STK scenario, displaying a visual simulation of a ground station contact using parametric diagrams in the reference architecture.

Grissom technicians at AFIT were consulted during the development of this reference architecture to determine what value properties should be included, and those mission modeling tools are actively being developed using this reference architecture as a baseline.

9. Future Work and Conclusion

To date, the CubeSat reference architecture effort has been to establish a working template to test with the next cohort of Space Systems Engineering students at AFIT. As the course sequence curriculum changes and as more people look at and use this reference architecture, improvements can and should be made for the benefit of future teams. The reference architecture, as is, has been tested by prior students, but every team has different styles and preferences, and the reference architecture can reflect those differences.

The Component Library will be expanded as the reference architecture is used by design teams. After several iterations, there will be several different types of payloads to choose from, multiple propulsion systems, chassis sizes, etc. The Component Library also contains other non-physical blocks for reuse as well, such as constraint blocks for analysis, object flows, value types, and custom stereotypes.

The Analysis section includes several working examples that work with the generic component blocks, but as future teams add working MATLAB code or STK configurations to their analysis, those can be saved in the Component Library as well. Future teams can copy any relevant constraint blocks to use in their own parametric diagrams, and over time, a wealth of working analysis can be saved and continuously improved upon.

Finally, the verification functionality built into the model primarily addresses technical performance, not programmatic requirements. Further work should be done to add functionality to validate top level mission requirements such as schedule or regulatory requirements.

In summary, this first attempt at a CubeSat reference architecture is designed to be improved over time. Currently, it will guide teams and constrain them in a way that makes the modeling effort easier so they can focus on the design details and technical analysis. Students new to MBSE should be able to use this model, and those more familiar with the tool can add new features for future teams to take advantage of. It will also be the platform upon which future mission modeling tools are based to integrate STK or similar tools for more in-depth analysis. The goal here is to keep system data within the model, and this reference architecture will help encourage teams to do that.

Acknowledgments

The author would like to thank the Center for Space Research and Assurance (CSRA) at AFIT for their guidance throughout this research.

References

Buede, D. and Miller, W. (2016): The Engineering Design of Systems: Models and Methods (3rd ed.). Hoboken, NJ: John Wiley and Sons.

Cloutier, R., Muller, G., Verma, D., Nilchiani, R., Hole, E., and Bone, M. (2010): The Concept of Reference Architectures. Systems Engineering. Vol 13: 14-27. doi 10.1002/sys.20129.

Delligatti, L. (2014): SysML Distilled: A Brief Guide to the Systems Modeling Language, Addison-Wesley Professional.

Estefan, J. (2008): Survey of Model-Based Systems Engineering (MBSE) Methodologies (rev. B). Seattle, WA, US: International Council on Systems Engineering (INCOSE). INCOSE-TD-2007-003-

02. Available at: http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf (accessed Nov. 5, 2020).

Hoffmann, H.P. (2020): Systems Engineering Best Practices with the Rational Solution for Systems and Software Engineering. Deskbook Release 3.1.2 ed. IBM. Available at: https://www.ibm.com/support/pages/model-based-systems-engineering-rational-rhapsody-and-rational-harmony-systems-engineering-deskbook-312 (accessed Nov. 18, 2020).

Jacques, D. and Cox, A. (2019): The Use of MBSE and a Reference Architecture in a Rapid Prototyping Environment, Tech. Rep., Air Force Institute of Technology. Available at: https://ndiastorage.blob.core.usgovcloudapi.net/ndia/2019/systems/Wed_22502_Jacques.pdf (accessed Nov. 4, 2020).

Karvinen, M., Tikka, T., and Praks, J. (2015): Using Hobby Prototyping Boards and Commercial-Off-The-Shelf (COTS) Components for Developing Low-Cost, Fast-Delivery Satellite Subsystems. J. of Small Satellites, Vol. 4, No. 1, pp. 301–314. Available at: https://jossonline.com/using-hobby-prototyping-boards-and-commercial-off-the-shelf-components-for-developing-low-cost-and-fast-delivery-satellite-subsystems/ (accessed Dec. 3, 2020).

Kaslow, D., Ayres, B., Cahill, P. et al. (2017): Developing a CubeSat Model-Based Systems Engineering (MBSE) Reference Model - Interim Status 3, Tech. Rep., IEEE. doi 10.1109/AERO. 2017. 7943691 (accessed Nov. 5, 2020).

Office of the Assistant Secretary of Defense, Networks and Information Integration (2010a): The DoDAF Architecture Framework Version 2.02. Available at: https://dodcio.defense.gov/library/dod-architecture-framework/ (accessed Nov. 5, 2020).

Office of the Assistant Secretary of Defense, Networks and Information Integration (2010n): Reference Architecture Description. Available at: dodcio.defense.gov/Portals/0/Documents/Ref_Archi_Description_Final_v1_18Jun10.pdf (accessed Nov 5, 2020).

Object Management Group (2011): INCOSE Object-Oriented Systems Engineering Method (OOSEM). Available at: https://www.omgwiki.org/MBSE/doku.php?id=mbse:incoseoosem (accessed Oct. 15, 2020).

Pradhan, K. and Cho, M. (2020): Shortening of Delivery Time for University-Class Lean Satellites. J. of Small Satellites. Vol. 9, No. 1, pp. 881–896. Available at: https://jossonline.com/wp-content/uploads/2020/03/Final-Pradhan-Shortening-of-Delivery-Time-for-University-Class-Lean-Satellites.pdf (accessed Dec. 2, 2020).

Walden, D., Roedler, G., Forsberg, K. et al. (2015): Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities (4th ed.). San Diego, CA: INCOSE. Available at: https://www.incose.org/products-and-publications/se-handbook (accessed Oct. 21, 2020).

Weilkiens, T. (2016): SYSMOD - The Systems Modeling Toolbox (2nd ed.). MBSE4U Booklet. Available at: https://mbse4u.com/product/sysmod/ (accessed Dec. 16, 2020).

Wertz, J. R., Everett, D. F., and Puschell, J. J. (2011): Space Mission Engineering: The New SMAD (1st ed.). Torrance, CA: Microcosm Press.