A REFERENCE ARCHITECTURE FOR CUBESAT DEVELOPMENT

THESIS

Luke J. Farrell, Second Lieutenant, USAF

AFIT-ENV-MS-20-M-199

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENV-MS-20-M-199

A REFERENCE ARCHITECTURE FOR CUBESAT DEVELOPMENT

THESIS

Presented to the Faculty

Department of Systems Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Systems Engineering

Luke J. Farrell, BS

Second Lieutenant, USAF

March 2020

AFIT-ENV-MS-20-M-199

A REFERENCE ARCHITECTURE FOR CUBESAT DEVELOPMENT

Luke J. Farrell, BS

Second Lieutenant, USAF

Committee Membership:

Dr. D. R. Jacques
Chair

Dr. B. J. Ayres
Member

Dr. R. G. Cobb
Member

Abstract

Developing a space vehicle is a complex and detailed process, and while CubeSats are smaller and more accessible than traditional satellites the design process is relatively unchanged. Creating a viable space vehicle design requires detailed analysis of a set of mission needs in order to define the mission, with this need set used to then create the specific mission requirements. These requirements are used to formulate a concept of operations, and then move into developing a physical system for executing the mission. The successful production of CubeSats within an organization is contingent upon the accurate execution of the general CubeSat Development Process.

This research presents a tool to facilitate more complete, streamlined, and transferable products throughout the course of a general CubeSat Development Process. The reference architecture is capable of displaying both organizational and systems level architectures, both linked together and in support of consistent and repeatable structure to be given to users intending to produce a complete mission and system design. The architecture incorporates a suite of repositories to assist users in hardware integration and requirements traceability, including component, activity, and regulatory libraries; in addition to parametric diagrams to facilitate requirements verification and constraint analysis.

Acknowledgments

I am forever grateful to my partner in life for all the love, support, and patience she provided me throughout my time at AFIT. I would also like to express immense gratitude to my thesis advisor, Dr. Jacques, for his steady guidance and assistance throughout this venture, and again to my committee for their insights and support. Finally, a round of thanks to all of the people who helped me along the way at AFIT; I never walked alone.

Luke J. Farrell

Table of Contents

Page

List of Figures

Page

List of Tables

Page

A REFERENCE ARCHITECTURE FOR CUBESAT DEVELOPMENT

I. Introduction

General Issue

Working with spacecraft is a complex endeavor, based in the most dynamic environment known to man. The area of operations for spacecraft is constantly changing and evolving, and the operators of these incredible systems have to constantly be on guard in order to react to both adversaries and the complex physics of the orbital regime alike. To meet these operational needs spacecraft have historically been designed in an exquisite and unique manner, with mostly one-of-a-kind engineering to provide a set of mission capabilities. Designing a spacecraft that has the capabilities and operational profile necessary to succeed in this environment is exceedingly difficult, and this difficulty is compounded enormously when the design of satellites moves away from one-of-a-kind system design and into large scale vehicle reproduction. It is relatively easy to smooth out the quirks and shortcomings when there is a single vehicle. When there are potentially hundreds of interconnected vehicles, such as a CubeSat constellation, then any inconsistencies become glaringly obvious and can undermine the entire system. Proper spacecraft design is of the utmost importance in order to prevent and mitigate as many of these issues as possible, before anything is built or fielded on orbit.

CubeSat Definition

CubeSats were formally defined in 1999 at California Polytechnic State University to create a universal standard for the development of "pico-satellites," many orders of magnitude smaller than other satellites of the time [1]. CubeSats follow an additive system of cubic units, and multiple units (or "U's") can be combined to create a progressively larger satellite. Each unit of a CubeSat is a 10 centimeter cube, with a mass of no more than 1.33 kg; each addition of a unit to a CubeSat defines the class of vehicle it is: 1 unit is a 1U CubeSat, 3 units is a 3U CubeSat, and so on [1]. The standardization of CubeSat dimensions, and the different size classes, allowed for the design of a mechanism to carry and ultimately dispense many CubeSats in a single launch; this allows more payloads, designers, and operators to benefit from a single launch than ever was possible before [1].

The CubeSat Program was created to promote a reduced barrier to entry in space vehicle design and operation. While this was initially taken on whole heartedly by the academic community as a way to teach students the basics of spacecraft design, the commercial benefits of CubeSats were quickly illuminated thereafter. A CubeSat could foreseeably be used as a lower cost testbed for a prototype technology, allowing for an organization to test emerging technology in the actual intended operating conditions rather than a best approximation here on Earth. However, a CubeSat does not need to be limited to being solely a test platform. Accordingly, a CubeSat could be the main mission, and thought is being given to the creation of large CubeSat constellations (a

network of CubeSats in Earth's orbit) by some interested organizations. These thoughts are investigated further in chapter II.

Spacecraft Design Sequence

AFIT has a Spacecraft Design Sequence, comprised of three courses: ASYS 531, ASYS 631, and ASYS 632, all completed in this order. At the beginning of ASYS 531, Space Mission Analysis and Systems Design, the students are given a mission that they will then develop certain artifacts for. These artifacts include foundational pieces of the mission such as the concept of operations, stakeholder needs, mission requirements, space vehicle requirements, cost estimates and risk estimates [2]. These artifacts will are carried forward to ASYS 631, Spacecraft Systems Engineering, where different subsystem solutions are presented in order to understand the parts of a satellite and how these parts may be used to provide a capability and satisfy the documentation from ASYS 531 [2]. ASYS 632, Satellite Design and Test, offers "a comprehensive overview of the design, manufacture, and testing of complex space systems," all derived from the documentation from ASYS 531 and the subsystem knowledge from ASYS 631 [2]. These courses are synchronized to create a viable mission and spacecraft design, and show students how to derive everything needed to create a spacecraft from a set of stakeholder needs.

Problem Statement

As satellite procurement moves towards mass-production of smaller CubeSats, the Department of Defense needs to form a design baseline to better align with systems

engineering approaches used in the commercial sector, such as rapid prototyping and model-based systems engineering (MBSE). With commercial launch opportunities opening up and US adversary development advancing exponentially, the DoD cannot afford to lose a step in its development cycle. A systematic approach to the cultivation of an organization wide reference architecture can give engineers the advanced starting point that they need to consistently create viable CubeSat designs and mission solutions.

Investigative Questions

The research herein is based upon a general process of Space Vehicle Design, and there are a few investigative questions that guide the subsequent findings:

1. *How can engineers reduce the design time of a desired mission and system for CubeSats?*
2. *Is it possible to produce traceable and defensible system designs on a consistent basis?*
3. *Is there a way to accelerate the learning process involved with Space Vehicle Design?*

These questions have most likely been asked by many engineers in a wide variety of CubeSat design teams, and are the framing basis for the objectives of the research outlined below.

Research Objectives

The objectives for this research are to reduce CubeSat design time while still adhering to the AFIT spacecraft design sequence and allowing students to learn,

determine common CubeSat components/functions, incorporate parametric analysis of satellite design properties, and to contain the work done in a systems engineering model for further project development. Using a common MBSE tool will allow for the design borne out of the reference architecture to actually be created, and will ultimately align with best practices in the commercial CubeSat community.

Methodology

The research will analyze the general space vehicle design process, incorporating MBSE techniques and practices in order create a Reference Architecture within an MBSE tool. The tool will be used by students and researchers at AFIT to create better satellite designs in a much quicker manner. The tool will incorporate common design themes found in CubeSat design and production, to include a notional library of functions, components, and parametric evaluation. The modeling tool will be used by students, faculty, and researchers at AFIT to apply MBSE techniques to CubeSat design while advancing the current state of the art.

Assumptions

The modeling tool developed herein will be useful to all who are trying to build a CubeSat, but it is assumed that users are familiar with the aspects of satellite mission design and the systems engineering process. The modeling software used is Cameo Systems Modeler, and the accompanying files are coded in MathWorks' MATLAB. The modeling tool is not intended for small satellites or one-of-a-kind satellites, and as such may not be as useful for these operations. The Grissom 6U CubeSat bus is an AFIT in-

house design that meets the Cal Poly specifications outlined earlier in this chapter, and will be the foundation of the reference architecture in the development process.

Selection of the Grissom bus does not limit the utility of the tool, and it is readily extensible to meet the needs of different busses and platforms outside of AFIT. The necessary scope of the research is simply to include the Grissom bus for subsequent AFIT research. The tool is designed to integrate within the AFIT Space Vehicle Design Sequence; many of the techniques and procedures found herein are specific to the Air Force Institute of Technology, but there is value to be had in the modeling tool itself for other organizations, as well as many learning opportunities that are beneficial to all of those in the CubeSat community at large.

Implications

The tool will require an input of requirements documentation created during a standard Preliminary Design Review (PDR), and then will be used in the subsequent systems acquisitions phase of the design process (to include a Critical Design Review and a Production Readiness Review), as well as in keeping a digital model of upcoming AFIT CubeSat missions in which the Grissom bus will be used. In addition to aligning AFIT with commercial best practices, this digital baseline model of the 6U Grissom bus will allow research to develop in many divergent paths, but all still be based upon a similar foundation.

A tool based upon a similar foundation gives AFIT a unique ability to instruct students in the art of spacecraft design in a classroom, yet allows students to evolve into

researchers and apply what they have learned to take the model in a litany of directions. The model can be added to and kept updated to always reflect the most accurate representation of the common foundation, the Grissom bus, and can also be used to track the progress of every Grissom based research project with respect to their own design requirements. While the Grissom bus is the foundation of this particular effort, the reference architecture can easily adapt to a different bus. The elements in the architectural design are compatible with many different possible CubeSat bus configurations, and would be of use to any organization looking to reduce their design time, maintain a digital model of their system, and have a complete set of components traced to their requirements.

Preview

This chapter has outlined the objectives and motivation for this thesis. Chapter II describes a reference architecture in more detail, and the common needs and processes within CubeSats. Chapter III explains the methodology used in designing the reference architecture and a brief detailing of the modeling tool used. Chapter IV describes the results and creation of the reference architecture. Finally, the conclusions, potential applications, and recommendations for future research is presented in Chapter V.

II. Literature Review

Chapter Overview

The purpose of this chapter is to review current research on using Model Based Systems Engineering to create a baseline model for a system domain or product line, also known as a reference architecture. This chapter shall further define a CubeSat and its contemporary use, what a reference architecture is, an example of a reference architecture in the Small Unmanned Aircraft Systems domain, and key systems engineering topics. It will conclude with an overview of previous work done on CubeSat Reference Architectures.

CubeSats

Traditionally, satellites are designed in a very limited run of production models, if even more than one of a kind is being built. Intuitively, if a team is only going to build one operational model of a system then they must perform an extensive run of testing (both physical/digital using modeling and simulation) to ensure that the system is going to work once fielded. If there is only ever going to be one chance at getting something right, then it must be practically guaranteed to work. The cost of these programs can become extremely prohibitive, especially when the costs of all of the different segments (hardware, software, launch, etc.) are added up. All of the testing required to validate the design could drive the price up exponentially, especially as system complexity increases.

CubeSats, on the other hand, are relatively inexpensive and easily produced. The rigor of testing can be balanced with the reduced consequences of failure. Robert Twiggs,

co-creator of the CubeSat Design Specification, asserts in *Space Mission Engineering: The New SMAD*, that failure, to an extent, is acceptable in the efforts to push technology forward [3]. These potential failures, while costly up front, ultimately drive costs down due to the valuable information that is brought back. To a large organization in the space domain, an investment below $500,000 for a CubeSat would be a relative drop in the bucket in comparison to traditional research and development efforts. For a university effort Twiggs ventures that the all-inclusive cost of a CubeSat should be less than $100,000, which presents incredible value for the university in both education and research-based perspectives [3]. The adoption of the standard CubeSat form factor by the commercial sector has also opened up a myriad of launch options and ridesharing opportunities, further bringing the cost down for users.

The value of CubeSats extends beyond research, though, as evidenced by some contemporary work in industry. SpaceX, in a 2016 FCC filing, announced their intention of setting up a constellation of CubeSats in Low Earth Orbit to provide broadband internet to the entire globe [4]. Constellations are not a new technique; many communications satellites are set up in these networks to provide approximately global coverage. These constellations typically number no more than a dozen satellites, yet the proposed CubeSat constellation from SpaceX is set to contain more than 10,000 in a few different shell orbits around the Earth [4]. This allows SpaceX to distribute the capability being delivered among the entire network rather than a small number of satellites, and in this manner it also allows them to dilute the risk. In a constellation of 12 satellites any

single failure could cause the whole constellation to fail; a constellation of thousands would present far fewer opportunities for a single failure to degrade the entire network.

The idea of a distributed capability is based on a set of devices with heterogenous functional classes that integrate together to form a single cohesive network. These distributed capability constellations allow the network to provide a similar capability as the larger, more individually complex satellites at a much lower risk profile. CubeSats are uniquely suited to this endeavor due to their size and ease of reproducibility. Each failure is not prohibitively expensive to fix, based upon the economies of scale of producing another CubeSat. Furthermore, single satellite failures do not need to be fixed immediately, the gap can be covered by another satellite of a similar function class in the constellation until a constellation resupply launch can occur.

With their birth in the educational world and subsequent adaptation to commercial interest, the CubeSat form factor developed in 1999 is a unique and powerful tool to be used by any organization seeking to claim a slice of the space domain. There is much that can be done with a CubeSat, and potential uses will continue to appear as more organizations are beginning to experiment with their development. With benefits in education, research, and commercial capability the CubeSat will be a valuable asset to many organizations globally.

Systems Engineering Design Process

Problem solving, at the most basic level, involves looking at a problem and then defining all of the things that need to be done to solve the problem. Once the needs have been defined then it must be decided how those needs can then be satisfied, followed by

the mechanism for meeting the needs. Then it is necessary to define a way to ensure that all of the parts of the solution work together without interfering with each other, and finally a way to verify if the system even does what it is supposed to do. This abstract problem-solving definition can be refined into the Systems Engineering Design Process, detailed in the following steps:

Table 1. Systems Engineering Design Process [5]

1. *Define the problem to be solved*
2. *Define and evaluate alternative concepts for solving the problem*
3. *Define the system level design problem being solved*
4. *Develop the system functional architecture*
5. *Develop the system physical architecture*
6. *Develop the system allocated architecture*
7. *Develop the interface architecture*
8. *Define the qualification system for the system*

Interestingly enough, this is not a progressive series of steps, and all of these efforts must be taken upon concurrently and iteratively to achieve the best result of system design. This is best evidenced by the necessary table of outputs and inputs for the design process.

Table 2. Functions of the Design Process [5]

Design function	Major inputs	Major outputs
Define the problem to be solved	Concerns and complaints by the stakeholders Available data from stakeholders	Definitions of measures of effectiveness and desired ranges Constraints
Define and evaluate alternate concepts for solving problems	Ideas for concepts from all interested parties	Recommended Concept(s) Objective hierarchy and value parameters for meta-system
Define the system level design problem being solved	Stakeholders' inputs	Stakeholders' requirements Operational concept
Develop the system functional architecture	Stakeholders requirements Operational concept	Functional architecture
Develop the system physical architecture	Stakeholders requirements	Physical architecture
Develop the system allocated architecture	Stakeholders requirements Functional architecture Physical architecture	Allocated architecture
Develop the interface architecture	Draft allocated architecture	Interface architecture
Define the qualification system for the system	Stakeholders requirements	Qualification system design documentation

Within the AFIT Space Vehicle design sequence all of these steps are met, and in fact it is interesting to note that the steps themselves are iterative and concurrent. In

ASYS 531 the course will go through all of these steps to produce all of the major outputs listed above, all predicated on a set of stakeholder needs. These stakeholder needs are written using the terminology of the stakeholder, and it is up to the engineers to communicate with the stakeholders and derive a set of system requirements in definitive language. For example, a need from a CubeSat stakeholder may read, "The satellite needs to fit inside the dispenser we already have." The engineer must determine what dispenser the stakeholder has, then rewrite the requirement to define exactly how large the satellite may be. A properly written requirements statement uses definitive statements such as *shall* to identify what is needed out of a stakeholder need, "The space vehicle *shall* be properly sized as a 6U satellite in accordance with *6U CubeSat Design Specification* dated 04/20/16 [6]."

Once these requirements are derived the top-level stakeholders' requirements document can be formed. This is not the end all and be all of the requirements to build a system, and as time goes on and more systems knowledge is gained then more requirements will be written to refine the top-level requirements. Using the derived requirements to build the functional, physical, allocated, and interface architecture is an interdependent process, and getting an initial condition of the spacecraft system is the purpose of ASYS 531. Finally, the last step is developing a qualification system, using a verification and validation matrix. *Verification* is the process of ensuring that all of the components of the system are built to specification and meet all of the requirements outlined for the system [5]. *Validation* is the process of ensuring that the system meets all of the needs outlined by the stakeholders, a sanity check to determine if the right system

was built [5]. It is possible that the nature of the stakeholders needs got misinterpreted or have evolved throughout the process, so it is important to validate the reasons that this system is even being built.

Model Based Systems Engineering

A system is defined as a "collection of hardware, software, people, facilities, and procedures organized to accomplish some common objectives [5]." All of these things can be represented within the output artifacts of the systems design process outlined above; this is a lot to think about at once when all of those components are folded in together, and when addressed head on it can look ugly and confusing. Simply trying to grasp all of these parts, however, is the minds attempt at informally modeling the system in an approachable manner [7]. Mental modeling, or informal modeling, is acceptable for personal understanding, but once the system design needs to be shared with other people it would be exceedingly difficult to explain and define every personal decision made. To avoid this, the systems engineering community at large has adopted a set of languages to talk with one another about things that they are modeling, such as the Unified Modeling Language (UML) and its linguistic sibling Systems Modeling Language (SysML) [5]. An engineer can quickly become fluent in these modeling languages, and then systems knowledge can easily be shared to a mutual level of understanding. Once the languages are understood and ideas can be transferred between engineers with similar understanding, it stands that the system can start being defined using the language.

Systems are defined in both a physical and conceptual manner using models. A model is an abstraction of reality that begins by explaining the needs that a system should

meet, and then drilling down deeper to show how the system is going to meet those needs [5]. As the engineers get further into the system design process the elements of the model become more complex and hold more detail about the system. Models can be used to compare design alternatives that meet the same set of needs, verify that a design meets the given set of requirements, and depending upon the nature of the model it can even be used in simulation activity [5, 7].

Within the model there are specific things that must be present to provide an acceptable representation of what constitutes the system. In addition to the artifacts derived from the systems design process, the model must also have the associated functions of the system, components and subsystems, and expected inputs and outputs. All of the documentation from above will inform the design choices within the model, and the model will continue to evolve as the system is better understood by the creators. The activities of the system can be looked at as the manner in which a problem or need shall be solved. These activities are refined by system functions that show exactly how the activity shall occur. For example, an activity of a CubeSat would be that it communicates with its ground station. The refining function would be the precise manner in which it does so, to include component related specifics such as how it shall slew, when to begin communicating, when to end, and so on until activity completion.

The components of the system are identified to a certain level of abstraction, contingent upon the needs of the engineer. The components can be created notionally, such as identifying the need for an onboard processor, or they can be specific and identify the type, performance parameters, physical characteristics and so forth. The choice

associated with a specific component is a design decision which must be weighed against requirements and evaluated for compatibility, performance, and other desired objectives based upon mission and stakeholder needs.

Systems Modeling Language Overview

Systems Modeling Language, or SysML, can be used to provide a graphical representation of important aspects of a model, including structure, behavior, requirements and parametrics [8]. SysML was created and is managed by the Object Management Group, and is an extension of the Unified Modeling Language [8]. SysML is the language used to describe the model of a system, and is expressed graphically using nine different diagrams. SysML can show how the system is structured by using a *Block Definition Diagram,* where the parts of the system are shown in a hierarchical order flowing down from the total system to the subsystems and components. It can show how a system behaves by using an *Use-Case Diagram*, and also show what specific actions the system performs by using an *Activity Diagram.* Diagrams will be discussed as they are introduced in Chapter IV when the Reference Architecture is detailed.

These diagrams are created by using *elements*, specific pieces of a model that can be connected together and related to other elements. The basic foundation of SysML is the *block*, which can be used in many ways to show the structure of a system. A block can be a component of a system with associated value properties, or a composite of a multitude of blocks that are woven together and connected to create a representation of a subsystem or the system at large. Blocks can also represent activities or actions that the system is capable of doing. SysML organizes a model by using a series of packages to

contain specific information. These packages operate like folders, and they can contain both diagrams and the model elements that make up these diagrams. The strength of SysML is the ability to convey specific relationships between all of the different elements in the model. SysML has structural tie ins that link between the different model elements and can show how they interface with one another or how they are otherwise related.

Delligatti states that the final part needed to model a system is a tool, configured in the selected modelling language, to display all of the model elements and diagrams in a cohesive manner [8]. The tool in use at the Air Force Institute of Technology is Cameo Systems Modeler. Cameo is produced by No Magic, Inc, and is a SysML based modelling tool commonly used across many domains to display information, collaborate, simulate, manage requirements and many other systems packages [9]. This modeling tool is used for the architectural work completed throughout this thesis. While all of the artifacts generated throughout will be made through Cameo, they will be written using SysML and as such could be recreated in any other SysML based modelling tool.

Reference Architecture

Many models have been built to fit many different needs, but it would seem to be inefficient to create a new model for every new project when an organization has common stakeholders, goals, hardware, and facilities. It may be more efficient to have a common baseline that takes into account many of the aspects of the organization, and gives the engineers a more advanced starting point than they would otherwise have. This is the purpose of a reference architecture. A reference architecture is a cumulation of knowledge that gives guidance and rules for structuring, classifying, and organizing a

model, and can ultimately “capture the accumulated architectural knowledge of thousands of man years of work” [10]. Maier and Rechtin posit that “if engineering is the art and science of technical problem solving, systems architecting happens when you do not yet know what the problem is [11].” Architecture is foundational to cultivating a guided approach to problem solving that allows for consistent application of principles necessary for project success.

It stands to reason that a reference architecture only works if there is going to be many congruent points between projects. Internally, an enterprise may have a reference architecture to show all of its employees how they expect products to be created or how their business practices are to work. There may be a domain specific reference architecture to ensure that all products being developed in a domain have similar foundations to ensure greater consistency. In a specific domain the reference architecture must then capture the links to all of the relevant “standards, legislation, domain constraints, and mandatory frameworks [10].” There are external drivers of a reference architecture that make it beneficial to have one as well, such as increased interoperability, rapid adaptability, and shorter time to market for an individual product [10].

The effort to make a reference architecture is not to ensure that one project is done to an acceptable degree. Reference architectures are made so that many more models and projects capture the best of prior designs, and then to allow for the reinvestment of modelling knowledge back into the architecture to ensure that the reference architecture is always growing and evolving. This allows the reference architecture to not only incorporate existing architecture efforts, but to maintain an eye

towards the future and develop innovative new products [10]. It is in the opinion of the author that reference architectures are going to become even more useful as the DoD, and world as a whole, move towards cloud computing and decentralized project management. Nothing is designed and managed in a single location in a large program in the DoD, and the common language and vision applied in a reference architecture that is used across the program would greatly enhance the capability to produce a better product.

Architecture of a system should not be a result of designing the system, it needs to be carefully thought out and planned before a system can begin being modeled because the architecture ensures that the system is also meeting the business needs of the organization [12]. The reference architecture can exist at multiple levels of abstraction at the same time [10]. When it comes to organizing the architecture, it must then be separated into its hierarchical levels of abstraction to ensure that guidance is given appropriately at each level. These levels of abstraction correspond with the enterprise at the highest level, the organization at the middle, and the actual system at the lowest level [12]. This ensures that the same guidance and constraints flows between all of the parts of the hierarchy and promotes consistency across all efforts.

Small Unmanned Aircraft System Reference Architecture

A reference architecture for Small Unmanned Aircraft Systems (SUAS) was developed for use in the AFIT Graduate Specialization Track in Unmanned Systems Design, Development, and Flight Test [13]. The reference architecture is intended to emphasize “a disciplined, repeatable process using Model-Based Systems Engineering (MBSE)” while also increasing student learning and the ability to transition between

relevant organizations [13]. Jacques and Cox developed the architecture within Cameo Systems Modeler, and use the architecture at the objective level as described by the Army Research, Development, and Engineering Command [12], focused primarily on specific product output for the SUAS specialization track.

The SUAS reference architecture contains a Basic Ground Station Model, a Basic Multi-Rotor System Model, Functional and Component Libraries for common elements, and sample build using the architecture. In addition, the SUAS reference architecture also includes parametric diagrams in order to do mathematical analysis on the design choices of the user. Jacques and Cox take a low level, build-to approach in their reference architecture, giving users the ability to build to a design specification using the provided model elements. Users of the SUAS architecture will be able to design their mission and system using the reference architecture, but more focus is given to the system design of the SUAS rather than incorporating any high-level supporting artifacts.

Jacques and Cox used their reference architecture to supplement a culture of rapid prototyping, with the reference architecture used as a springboard for pushing innovation out of the researchers at a much faster pace. A common starting point and guiding vision for their architecture helps their interdisciplinary teams of researchers design, build, and test SUAS with more time spent on producing a quality product, and less time spent designing a novel architectural framework [13]. Empirically, the more time spent with attention towards quality design should ultimately yield a better product; a reference architecture gives engineers the guidance and constraint to ensure that the product fits the predetermined vision.

Jacques and Cox were able to capture a decade's worth of experience in the SUAS field inside their reference architecture, and this will ultimately grow as more knowledge is gained in the field. As the architecture is used the lessons learned will be reinvested back into the model, and this knowledge transfer and evolution will fuel rapid innovation to meet the necessarily high demands of the rapid prototyping environment.

Relevant CubeSat Reference Architecture Research

Kaslow et al. built a Cubesat Reference Model (CRM) providing a logical architecture to form the basis for many different CubeSat missions [14]. Their architecture describes three levels of architectural foundation that are necessary to capture the whole domain: the enterprise level, the space and ground segment, and the space and ground subsystem. This is similar to the *enterprise-organization-system* structure of Army RDECOM, but has been adapted to be space domain specific [12].

Kaslow et al. also used Cameo Systems Modeler to develop their CRM, using their approach to logical architecture as a way to provide a framework for future CubeSat developers. The goal of the reference model was to remove the burden of creating an acceptable architecture at a high level. Figure 1 indicates the structure for the CubeSat domain as described by Kaslow et al.

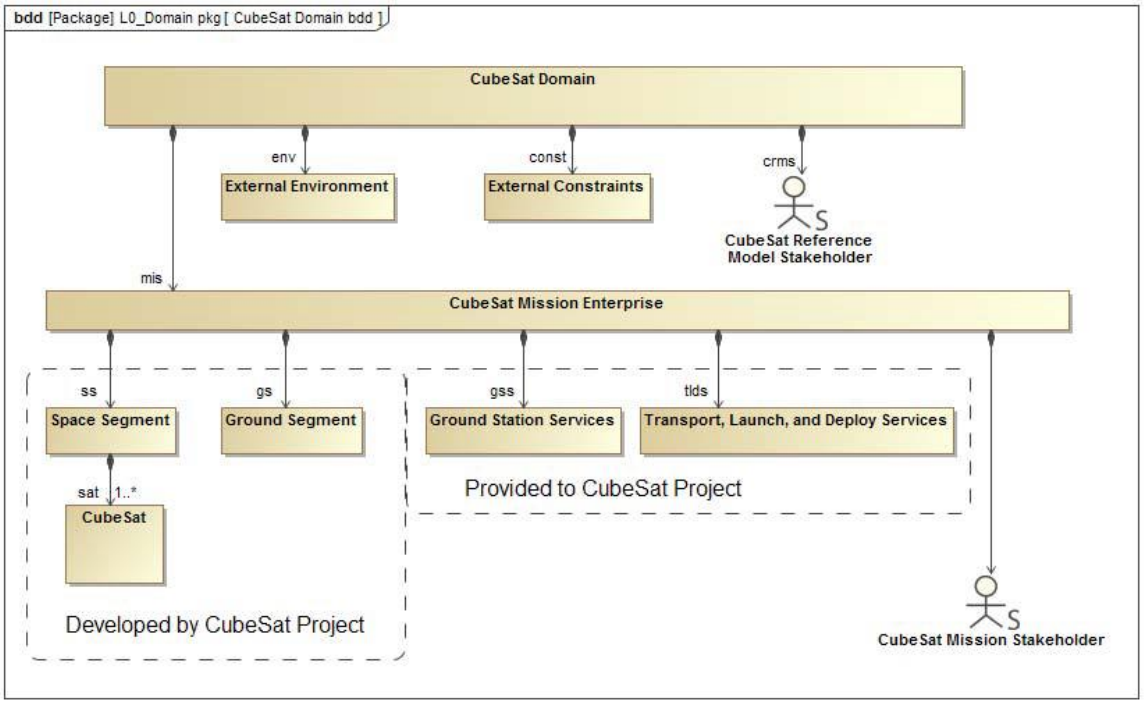


Figure 1. Kaslow et al. CubeSat Domain and Mission Enterprise [14]

Kaslow et al. used a block definition diagram to demonstrate the hierarchy of elements within the domain. They depict the CubeSat Mission Enterprise as being *composed* of a Space Segment, a Ground Segment, Ground Station Services, and Transport, Launch, and Deploy Services. This directed composition serves to indicate that if any of those elements are missing the CubeSat Mission Enterprise would no longer exist; they are critical components of its structure. Furthermore, they are able to identify what must be developed by the CubeSat Project in greater detail, as shown by Figure 2.

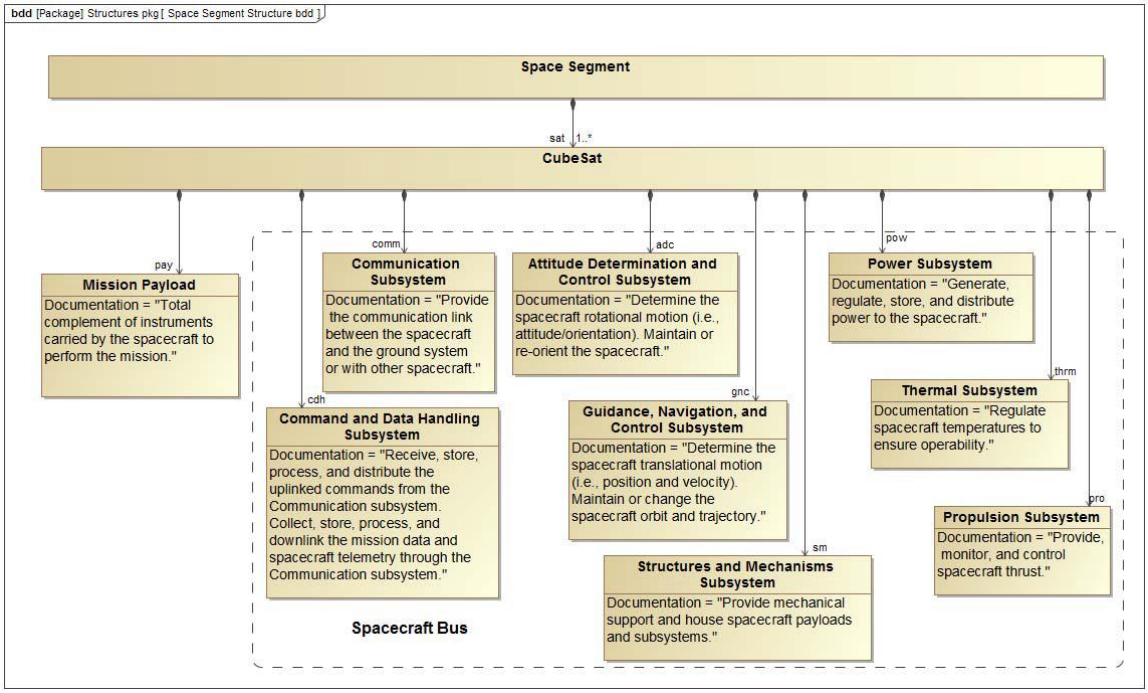


Figure 2. Kaslow et al. CubeSat Space Segment [14]

Much like in Figure 1, Kaslow et al. have described all of the parts that a CubeSat is composed of, and in this particular Block Definition Diagram they have also chosen to distinguish between the Mission Payload and the Spacecraft Bus (shown by the dashed line). This is done to show the necessary structure/components of a CubeSat (the Spacecraft Bus) and the on-orbit mission structure/components (the Mission Payload). This same process was also continued to formulate the composition for the Ground System Segment as well, using similar organization.

Kaslow et al. determined that this logical architecture would provide guidance for CubeSat developers to begin to formulate their own mission specific architectures, knowing that their model did not have and could not have the specificity required to support every type of mission [14]. It provides a top-level guide to how a CubeSat

enterprise is organized, and some of the external parties to the entire system as well. Their model is a starting point for mission specific teams to incorporate their unique knowledge to formulate their own reference architectures.

Upon investigation, the CubeSat Reference Model is missing much of the low-level exposition that was included in the SUAS Reference Architecture. A thorough reference architecture in this domain ought to include the high-level documentation and views of the CRM and the low-level componentry and functionality of the SUAS reference architecture. Combining the distinct approaches of these two architectures would yield a thorough model of the intended domain.

Summary

In summary, Chapter II defined the Systems Engineering design process and how it intertwines with the Spacecraft Design sequence at AFIT. The chapter gave a brief overview of the Systems Modeling Language and the appropriate use of the language to create models. The models that are created using SysML are hosted on a tool that is written in that modeling language, and these tools are used to create diagrams and elements to properly convey all of the information about a given model. It is tedious to create a unique model for every different system, so the chapter also explored the concept of a Reference Architecture, which provides a common vision, guidance, and constraints to focus the modelling and design efforts of a new project. A system specific reference architecture for Small Unmanned Aircraft Systems was examined, and a top-level reference architecture for a CubeSat Reference Model extended this exploration.

III. Methodology

Chapter Overview

The purpose of Chapter III is to describe the process behind the creation of a reference architecture for use by AFIT students in the Space Vehicle Design sequence. The reference architecture will take into account all of the background knowledge described in Chapter II. The current inputs to the Space Vehicle Design sequence will be described, as well as the desired system outputs from the proposed tool. The reasons for choosing a reference architecture as the desired form will be discussed. Once the system inputs and outputs have been identified, the process of creating a model to facilitate the desired output operations will be explored. A description of the intended use of the reference architecture within the Space Vehicle Design sequence will be identified.

System Inputs

At this point, there have been no labels placed upon the tool to be designed to supplement the Space Vehicle Design sequence at AFIT. The tool is to be designed to meet the objective as outlined in Chapter I, using Model Based Systems Engineering to reduce the time needed to design a space vehicle mission. The scope of the tool must be narrow enough that students are still able to go through requirements derivation and first pass iterative design, but large enough that the students can design and produce a wide variety of viable designs.

As the Space Vehicle Design sequence is described in Chapter II, there is a sequential flow through the three required courses. ASYS 531 is the first course, and the

basis of this course is mission design, culminating in a System Requirements Review (SRR). Leading up to the SRR, student efforts are focused through the use of *Space Mission Engineering: The New Space Mission Analysis and Design* [3]. Wertz et al. give much of their attention to the concept of mission engineering as opposed to systems engineering; given by the following sequence of events:

Table 3. Space Mission Engineering Process [3]

Step in Sequence	Action
Define Objectives and Constraints	*Define the Broad (Qualitative) Objectives and Constraints*
	Define the Principal Players
	Define the Program Timescale
	Estimate the Needs, Requirements, and Constraints
Define Alternative Mission Concepts or Designs	Define Alternative Mission Architectures
	Define Alternative Mission Concepts
	Define the Likely System Drivers and Key Requirements
Evaluate the Alternative Mission Concepts	Conduct Performance Assessments and System Trades
	Evaluate Mission Utility
	Define the Baseline Mission Concept and Architecture.
	Revise the Quantitative Requirements and Constraints
	Iterate and Explore Other Alternatives
Define and Allocate System Requirements	Define System Requirements
	Allocate the Requirements to System Elements

Wertz makes the distinction between mission engineering and systems engineering in his definition of the space vehicle design process. Whereas Systems Engineering theorists such as Maier, Rechtin, and Buede would argue that the process of systems engineering is one performed in totality from cradle to grave, Wertz describes

systems engineering as simply requirements definition and validation [3]. To use Wertz's terminology, the mission engineering sequence is the primary focus prior to a System Requirements Review. The approach taken at AFIT is more akin to the manner described by Maier, Rechtin, and Buede and is focused through a Systems Engineering process that closely follows the technical process of mission analysis, stakeholder needs, requirements definition, and system requirements definition [5, 11]. ASYS 531, the first course, is the source of all of the starting inputs for the proposed tool. The course products are as follows:

- Concept of Operations
- Engineering System Requirements
- System Functional and Physical Partitioning
- System Integration
- Verification and Validation
- Technical Reviews
- Configuration and Interface Management
- Cost Analysis
- Risk Management

Not all of these products are part of the Systems Engineering Design Outputs shown in Table 2, but that is acceptable since these are general outputs that are crucial to any project development process across the spacecraft development industry. All of these

outputs are necessary when conducting an SRR. Distilling the above list into the outputs of Table 2, the inputs into the system model are as follows:

Table 4. System Requirement Review Finalized Output Documents

PDR Output	Definition
Concept of Operations (CONOPS)	Describes how the system will fulfill the stakeholder needs and objectives [14].
System Requirements	All of the things that the system must have or do in order to meet the needs of the stakeholders.
Functional and Physical Partitioning	Functional parts of the system are abstract representations of whatever it is that the system is going to do. Separating system functions from physical attributes is necessary to ensure that system behaviors are accurately captured. During the PDR the system is still notional, this document shall be a first attempt and will not be complete.
Verification Documents	Verification means that the system requirements have been checked and all are satisfied. While the final verification methods are unknown at this point, this document shall give a general idea as to the methods in which the requirements will be verified.
Validation Documents	Validation means that the system is meeting the needs of the stakeholders. Validating a system entails proving that it is bridging the capability gap shown in the stakeholders needs document. This document shall give a general idea at how the system shall validate the stakeholders needs

Desired System Outputs

These documents shown above form all of the information that must be taken on by the proposed tool. The documents are not definitive, but are an excellent foundation for the inclusion of more systems knowledge to accurately flesh out the ideas contained within. The process of adding more substance to these initial documents will be done through the proposed reference architecture.

In addition to complete versions of the above documents, the stakeholders in the Space Vehicle Mission Design sequence need the system model to capture a completed functional, physical, and interface architecture for the given mission. Furthermore, the system needs to be centered around the in-house satellite bus developed at AFIT, the Grissom bus, but should remain extensible to accommodate for future CubeSat buses. The model needs to include common componentry that has all the defining parameter types listed, and must be built in a way that captures current laws, policies, and regulations as they pertain to CubeSat manufacture and use. The model needs to have complete descriptions of mission activities and use cases for the satellite, to include any interactions with external actors (such as other satellites not part of the mission profile, entities, etc.).

Although the output for many different missions will of course be different and mission dependent, the stakeholders at AFIT need this tool to be repeatable and able to grow with time. A repeatable tool is of great use to the stakeholders to accelerate learning, produce consistent and coherent decisions, and allow student users of the tool to take it in a multitude of directions. This initial concentration of learning and subsequent

rapid diversification of development is fantastic for the ability of AFIT to teach, learn, and innovate in this burgeoning domain.

This tool must then be applied to the physical CubeSat development process to get the satellite actually built, or at least test componentry and integration efforts in ASYS 632 Satellite Design and Test. This process is outlined in Table 5:

Table 5. CubeSat Development Process [15]

Step	Project Phase	Typical Timeframe
1	Concept Development	*1-6 months*
2	Securing Funding	*1-12 months*
3	Merit and Feasibility Review	*1-2 months*
4	CubeSat Design	*1-6 months*
5	Development and Submittal of Proposal	*3-4 months*
6	Selection and Manifesting	*1-36 months*
7	Mission Coordination	*9-18 months*
8	Licensing	*4-5 months*
9	Flight Specific Documentation Development	*10-12 months*
10	Ground Station Design, Development and Test	*2-12 months*
11	CubeSat Hardware Fabrication and Testing	*2-12 months*
12	Mission Readiness Review	*Half day*
13	CubeSat to Dispenser Integration and Testing	*1 day*
14	Dispenser and Launch Vehicle Integration	*1 day*
15	Launch	*1 day*
16	Mission Operations	*Variable, up to 2 years*

The inputs of the tool must be turned into outputs that can be used in this process. Most of these steps are covered in the AFIT Space Vehicle Design sequence, but even the

ones that are not covered (such as licensing) must be incorporated into the final tool so that users can use the tool for research and products that will need it.

Selection of a Reference Architecture

Creating some tool, template, or framework seemed like the clear path to take. It would be straightforward to create a template based upon the Grissom bus, with components that have already been vetted or tested. This may appear as a great solution, but in an academic institution this is stifling to the learning of the students, and completely detrimental to any research efforts that would be going on using the CubeSat. Furthermore, with the rapid rate of technological advancement and maturation in the CubeSat domain, any premade selections in a template would be obsolete within a relatively short period of time. Students would have little options to differentiate themselves from their peers, and little problem solving on the part of the students would occur as a result.

While in each situation the problem is unclear, and ostensibly different in every case, there is a clear commonality. Every student who is going to use this tool is coming through AFIT, with all of the same tools at their disposal. Each student would have the same foundation upon which to stand, so why not codify what that foundation is? Every model of a conceptual system must undergo some process of architecting, otherwise the model would be nonsensical and disorganized [7]. Giving the students an elevated platform upon which to start will surely accelerate their design process, and will allow the students to spend less time building the architecture themselves and more time focusing on developing a viable mission design and CubeSat.

Ultimately, knowing that the stakeholders needed a Model Based Systems Engineering solution to a problem that is repeatable, yet different every time led to the decision to create a CubeSat Reference Architecture. This architecture would capture the knowledge of many different viewpoints, and have the ability to grow as the CubeSat development at AFIT grows. This ability to continuously be updated to reflect the current state of the art in the domain is of critical importance if AFIT is to remain on pace with the current rate of technological advancement.

Creating the Architecture

Maier and Rechtin state that designing an architecture is an eclectic venture, and many approaches must be taken to create a single architecture [11]. They assert that creating an architecture is a stepwise reduction of a process that is already known to its most abstract level, and then re-synthesizing those abstractions into the architecture that is desired [11]. In this case the desired synthesis is space vehicle design, specifically CubeSat design. The space vehicle design process was synthesized from Wertz's *Space Mission Engineering: The New SMAD*, NASA's *CubeSat 101*, and the AFIT Space Vehicle Design sequence [3, 15]. Taking the input process and output processes that were defined above, reducing them to abstractions, and then reassembling them inside of a SysML Model Based framework is the result. These findings are discussed in Chapter IV.

Intended Use

The resulting reference architecture is intended to be used after the mission definition stage that has culminated in a Preliminary Design Review, and in every step after the PDR. The reference architecture could even be used leading up to the PDR as a "read-only" file until students have a firm grasp on their mission and the mission engineering process. Once students have a first pass at their mission design they may begin to use the reference architecture. There is tremendous value in the students designing their own mission from their particular set of stakeholder needs, and the reference architecture will help them understand their own mission better by giving them a foundation to build upon.

After the mission design process has been completed the architecture will take on all of the mission specifics for each design group. More components will be defined and the design will become more concrete. Component and hardware testing results can be input into the architecture, with the consecutive instances saved in instance tables in order to keep a record of all of the different configurations evaluated. The architecture can then follow the space vehicle design all the way until the completion of the design (both building and testing) and the progression towards actually getting a system launched into space, with a complete model of the system based upon the reference architecture.

Summary

Chapter III described the proposed inputs to the desired tool and the necessary outputs that should result from its use. Based upon these inputs, outputs, and the specifics of the Space Vehicle Design sequence at AFIT a reference architecture format was the

selected form of the tool to fit the desired function. The reasons for choosing a reference architecture format were discussed. Finally, the intended use of the reference architecture was outlined. Chapter IV will detail the resulting reference architecture and show some brief examples of projected uses.

IV. Analysis and Results

Chapter Overview

The purpose of Chapter IV is to outline the creation of a CubeSat Reference Architecture based upon the methodology described in Chapter III. The reference architecture is built with the background knowledge gleaned from the efforts described in Chapter II, and it is purpose built to meet the educational and research needs of the Air Force Institute of Technology and the Space Vehicle Design sequence within its graduate program. The chapter outlines how the reference architecture is organized within Cameo Systems Modeler, and describes the organizational level architecture and the system level architecture. The CubeSat repositories and structural diagrams will be detailed, and finally the inclusion of parametric diagrams will be explored. A brief case study of the use of the reference architecture based upon previous AFIT Space Vehicle Design Sequence Material will be shown after the Reference Architecture has been described.

Package Hierarchy

The reference architecture opens to a content diagram, showcasing all of the notable content within the architecture in a single view. It includes the two levels of architectural support, an organizational architecture framework and a systems level architectural framework. Within each package there are cascading package and diagram options to select, and all of the information therein is easily accessible to the user. Nestled beneath the structures is the CubeSat Reference Architecture User's Guide, attached in Appendix A.

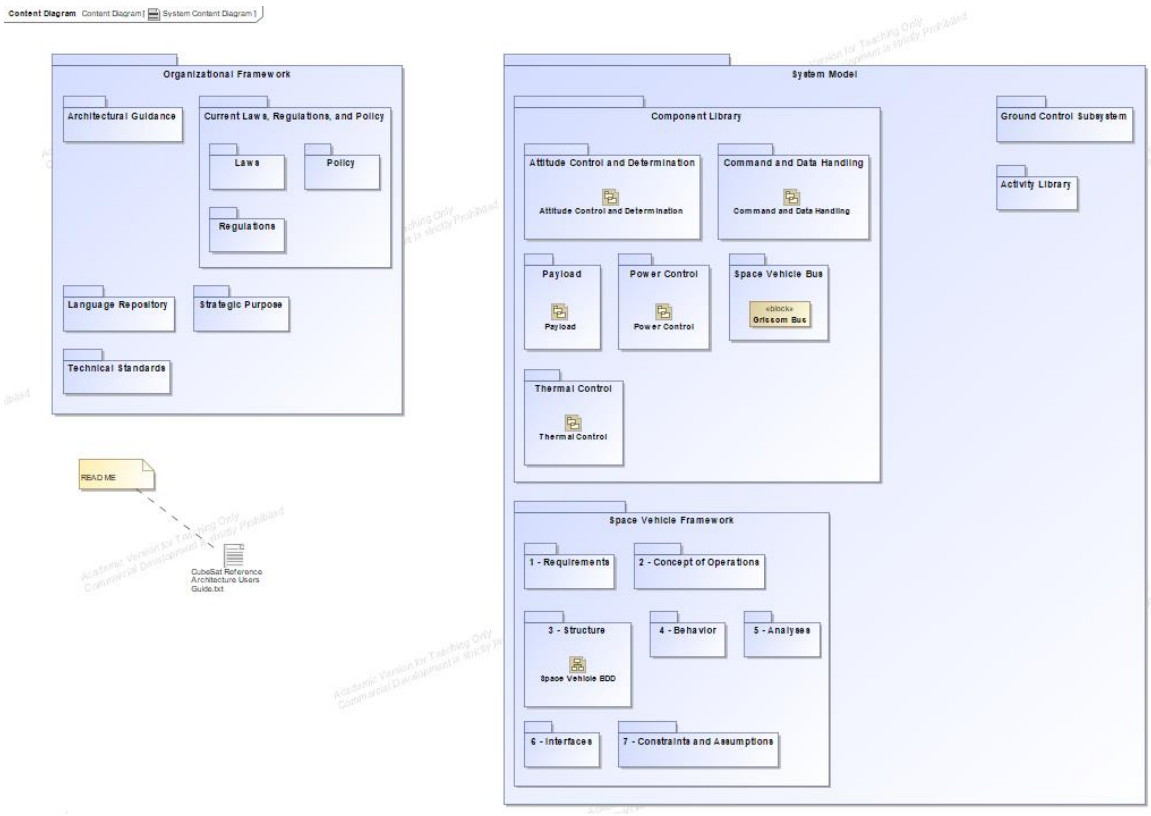


Figure 3. CubeSat Reference Architecture Content Diagram

Every element of the model is accessible through this content diagram without having to individually search throughout the containment tree, and is grouped with its common elements.

Organizational Level Architecture

Given the three level architectural framework outlined by Kaslow et al. as a starting point, it is beyond the scope of this research to include an enterprise level description of the CubeSat Reference Architecture. As such, the research only contains Organizational Level Architecture as the upper level of guidance for the user. The organizational level is meant to capture the vision of AFIT for its CubeSat research

applications, and is geared towards reuse and evolution. As AFIT begins to put more CubeSat research projects and missions into effect, the reference architecture will adapt to the increasing domain knowledge. The Grissom bus hasn't reached a high level of operational maturity yet, and with multiple missions planned in the future AFIT will have plenty of opportunities to grow.

In order to capture these lessons learned there is a library for Architectural Guidance. The current library contains a block definition diagram of the notional structure of the AFIT CubeSat Domain, and this can evolve to meet the changing needs of AFIT and its research sponsors.

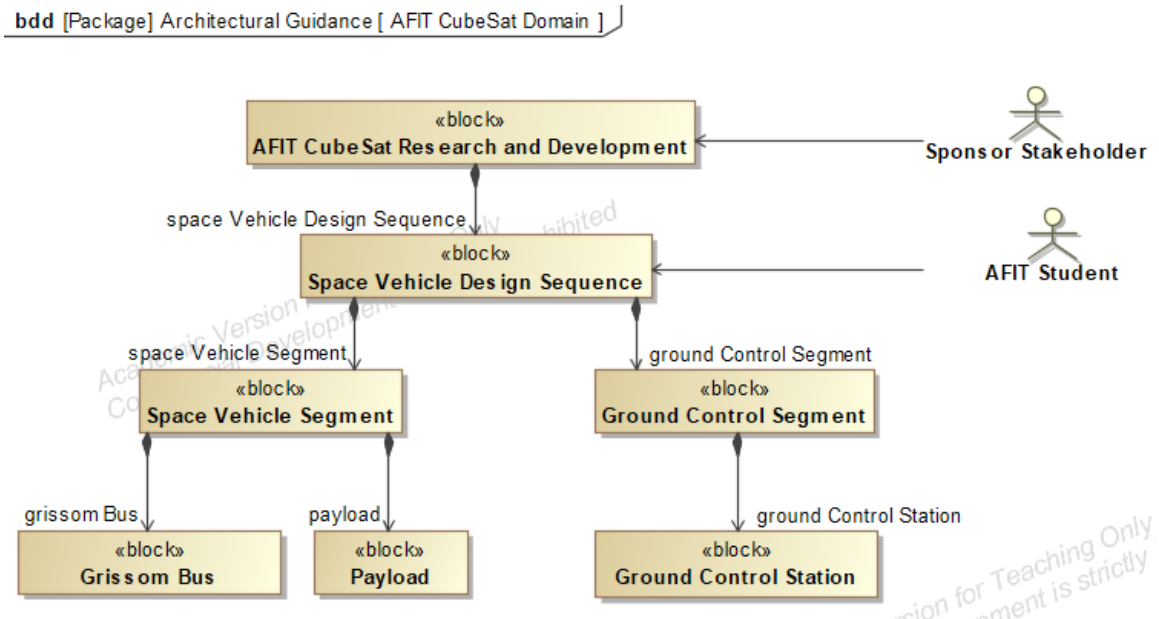


Figure 4. AFIT CubeSat Domain

The domain highlights the associated stakeholders within AFIT, denoted by the human shape in Figure 4. These stakeholders are the drivers behind what is being done at AFIT, the impact of the Sponsor Stakeholder flows down the composition hierarchy. The

Sponsor Stakeholder represents faculty, research technicians, and sponsoring organizations; their needs drive much the CubeSat Research and Development that happens at AFIT. Further down the hierarchy the AFIT students' educational needs push what happens during the Space Vehicle Design Sequence, while simultaneously meeting the needs of the Sponsor Stakeholder. This top-level view of the AFIT process outlines the domain within which the CubeSats are being developed, and who is pushing this research.

While the general CubeSat development field across the United States has a relatively limited system of regulatory oversight, AFIT has additional unique obligations due to being a part of the Air Force. While other university developers have to comply with NASA regulations and others of a similar vein, AFIT must also comply with all Air Force regulatory policy. This can lead to a requirements traceability nightmare, as the addition of these extra regulations can actually lead to a significant amount of overlap and unintentional replication of regulatory policies. In order to combat this the organizational level contains a repository for Laws, Policies, Regulations, and Technical Standards. The amount of documentation that could arguably fit into each of those containers is far too large to feasibly consider, so a narrowly scoped inclusion of documentation was initially included. Following guidance from Air Force regulatory subject matter experts and NASA's CubeSat 101 documentation the following Laws, Policies, Regulations, and Technical Standards were incorporated into the reference architecture [15]:

Table 6. Included Regulatory Documentation

Documentation Identifier	Description
NASA-STD-6016 **NASA Technical Standards Program**	Standard Materials and Processes Requirements for Spacecraft
NASA-STD-8719.14	Process for Limiting Orbital Debris/Procedural Requirements for Limiting Orbital Debris Generation
AFOSHSTD 48-9 **Air Force Occupational Safety and Health Standard**	Radio Frequency Safety Program
AFSPCMAN 91-710 **Air Force Space Command Manual**	Launch Vehicles, Payloads, and Ground Support Systems Requirements
GSFC-STD-7000 A	NASA General Environmental Verification Standard (GEVS) for Goddard Space Flight Center (GSFC) Flight Programs and Projects
SMC-S-016	Space and Missile Systems Center Standard Test Requirements for Launch, Upper-Stage, and Space Vehicles
National Space Policy	National Vision and Goals as it Pertains to the Use of Space
DA-13-445A1	Obtaining FCC Licensing Rights to Broadcast from Space

These policies produce requirements that are separate from the stakeholders needs in the AFIT CubeSat Domain. While one could look at these governing organizations as a form of a stakeholder, for the purposes of this architecture the documentation that the agencies have produced shall serve unattached to those organizations. The inclusion of these documentations as attached files to the reference architecture does increase the size,

but as of this moment it has not made it too unwieldy to transfer knowledge. To combat any ill-effects of potential future regulatory growth the documents have been attached to blocks; in the case that documentation has to be removed the block can still serve as the representative to trace requirements to.

While these documents will be applicable to every CubeSat mission produced through AFIT, not every possible requirement from these documents shall apply. As such, the mission specific adaptation of this reference architecture must identify the relevant requirements within the documents, and capture them in the model. These requirements should then be traced to the block associated with the document they were derived from. This will give a complete picture of regulatory requirement traceability, which is important for the verification of the developed system.

In order to enhance consistency of language across the users of the reference architecture there is a repository of common systems engineering terminology. This reference architecture is built knowing that the users of this tool will be multidisciplinary. While not everyone may have an extensive systems engineering background, this repository will level the playing field and make all terms clear to all users. A selection of the terms outlined in the language repository is shown in Table 7.

Table 7. Systems Language Glossary

#	Term	Description
1	Concept of Operations	Describes everything that the system is supposed to while on the mission. Includes operators, support, and necessary interfaces with objects external to the system.
2	Stakeholder Needs	Whatever capability gap that the stakeholder has, written in the language of the stakeholder.
3	Stakeholder Requirements Document	The distilled version of the stakeholder needs. The needs have been parsed and written into a definitive technical format for traceability purposes. This document provides the basis for many of the design decisions for the system, and may be refined as system knowledge increases.
4	Requirement	Something that needs to be done for the system to meet a stakeholder need
5	Objective Requirement	A secondary requirement that is not necessary to pass verification efforts, refines a threshold requirement and is normally set to a higher standard than the threshold requirement.
6	Threshold Requirement	A requirement that must be met at a specified minimum level, if the minimum level is not met then the requirement will fail verification efforts
7	Measure of Effectiveness	A quantitative effort to describe a qualitative function. If a system is being effective it is meeting the needs of the stakeholder, and these qualitative observations shall be codified by a measure of effectiveness. Typically used as a validation schema, these evaluate whether or not a system is meeting the operational need.
8	Technical Performance Measures	The method through which the technical performance of the system, subsystem, or component is being evaluated.
9	Key Performance Parameters	These are parameters within the overall performance of the system that are mission critical, and without these parameters meeting a certain threshold value the entire mission shall fail.
10	Use Case	The expected manner of use for the system within different mission scenarios. Delineates system factors and external factors that interact with the system.

The language repository also holds packages that can be filled with common object, signal, and value types that come up as a result of the repeated use at AFIT. These packages will be filled as the Grissom bus gains more flight heritage, and as missions using similar Commercial Off the Shelf parts (COTS) are undertaken there will be increasing familiarity with the necessary components, signals, and values for these parts.

The final piece of the Organizational Hierarchy is a package to hold the Strategic Purpose of each specific mission. The strategic purpose of each mission shall integrate with the overall purpose and vision of the AFIT CubeSat Domain, but will be tailored to each specific mission that it is derived from. The Department of Defense Reference Architecture Description calls for the inclusion of a Strategic Purpose within all DoD reference architectures; the strategic purpose shall identify goals and objectives of the architecture, and describe the specific purpose of the mission and the problems to be addressed [16].

System Level Architecture

The system level architecture encompasses all of the low-level specifics related to whatever the mission need is. The system level architecture is centered upon the AFIT designed bus, the Grissom bus, in order to provide a common framework upon which multiple missions can be based. Every potential mission cannot be explicitly forseen, so the system level architecture features a series of notional subsystems and componentry, with the expectation being that users will be able to select the specific hardware/software they intend upon using in their system instantiation; this process will be explored later on in the chapter.

The system architecture is divided into a set of repositories (activity library and component library) and subsystem segments (Space Vehicle and Ground Control) in a similar manner as Kaslow et al. [14]. The activity repository has sets of example activities that a CubeSat may perform as it goes about its orbital cycle, with most of the activities geared to simple orbital housekeeping activities. This intentional broadness allows for the activities at their most basic to function as a procedural check value, but also to become the foundation for more specialization in order to fit a specific mission need.

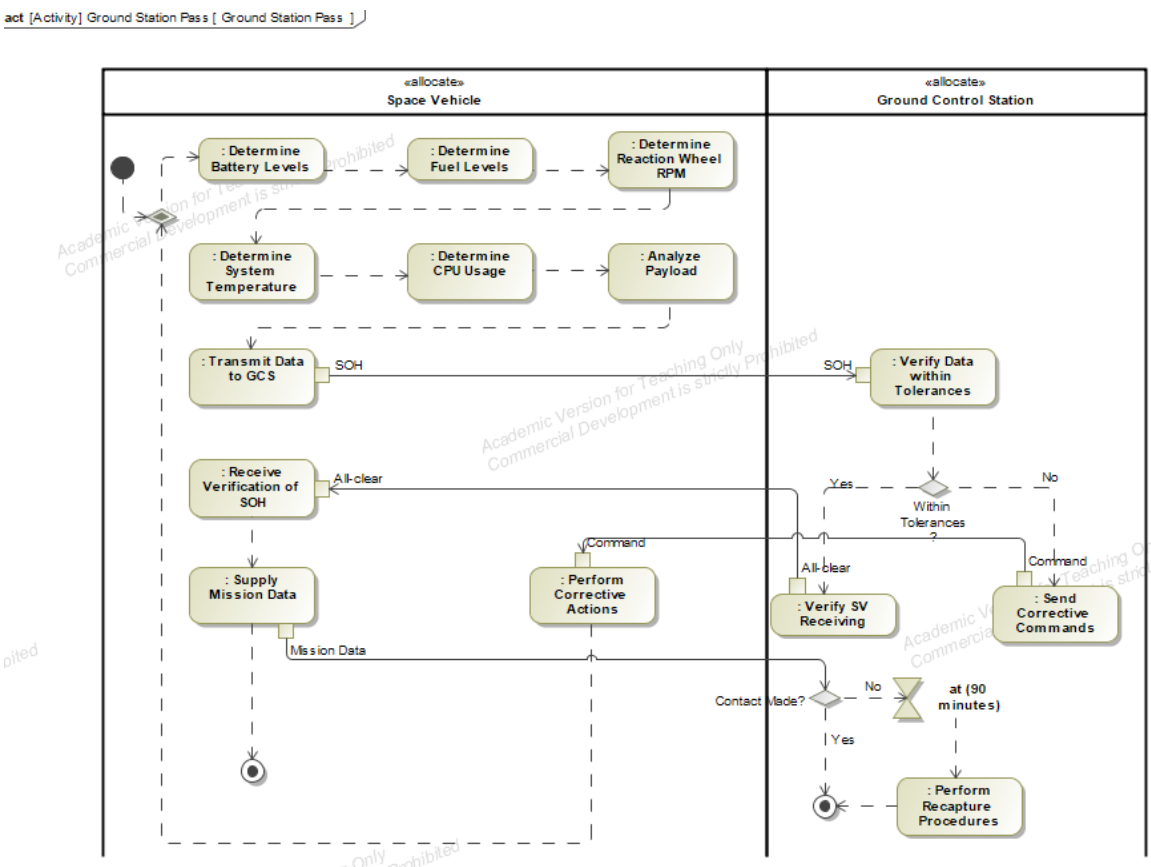


Figure 5. Example Activity Diagram: Ground Station Pass

The activity diagram shown in figure 5 shows a few key activities in a basic ground station pass. The activity begins with the shaded dot in the upper left corner, and follows the SysML notation throughout to indicate what is happening as the satellite goes through its pass of the ground station. The dark border lines, called swimlanes, indicate which block, or segment, is performing the activities, and then these activities are allocated to whichever segment is performing them. The colon in front of all of the names of the activity bubbles indicate that that activity is coming from a specific block, that the behavior is being called with respect to the block that owns the behavior. Ownership implies that the activity isn't happening on its own, and that the activity is being done by the block at the top of the swimlane.

This semantic distinction is necessary to show that the activity has consequences, and ultimately aids in simulating that activity if necessary. Since the activities are owned, the results of that simulation can then be attributed to the block that owns each activity.

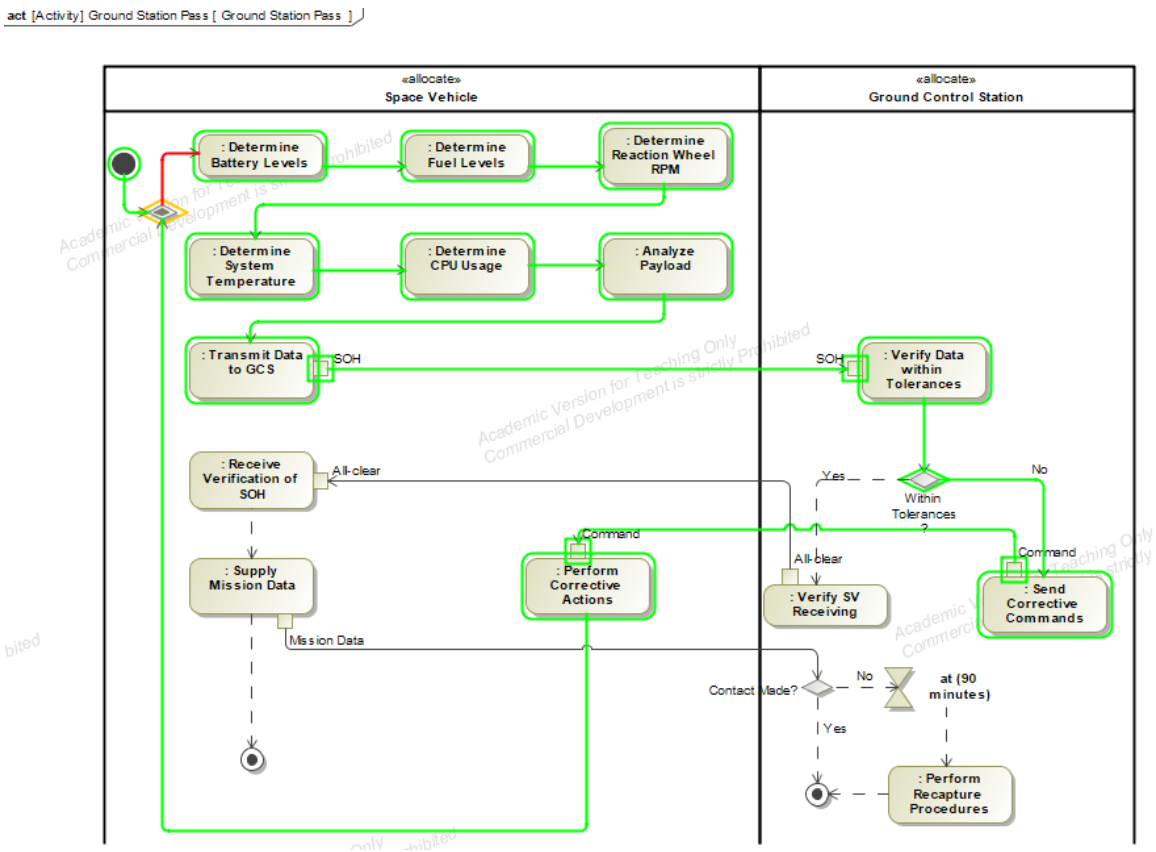
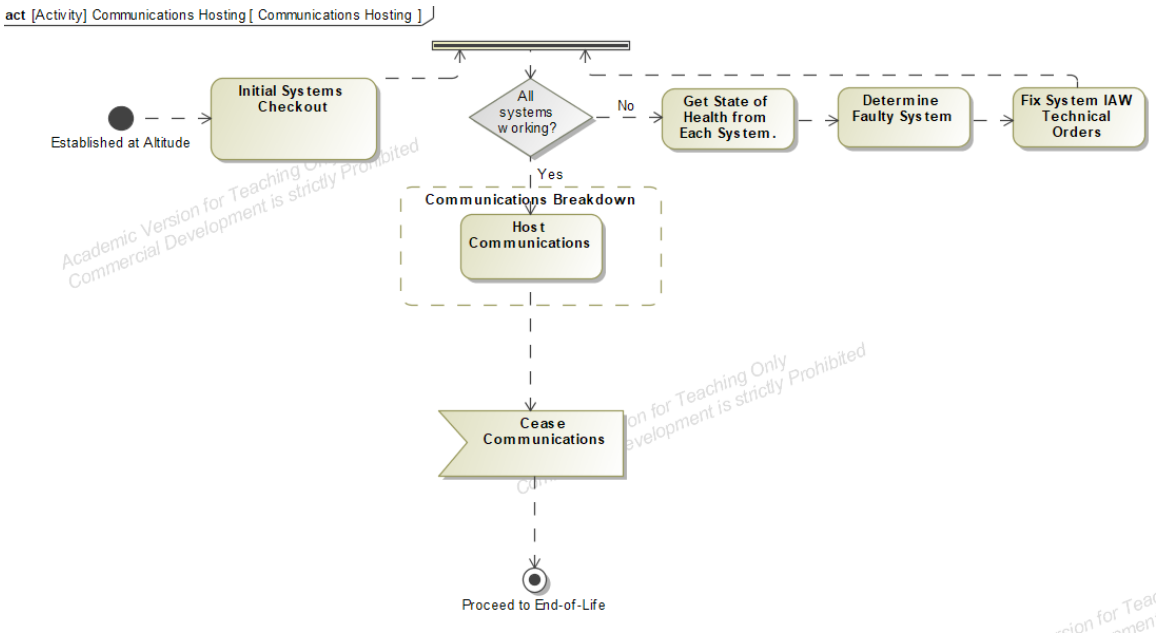


Figure 6. Activity Diagram Simulation

The activity diagram has now been simulated in order to show the logical progression of the flow between activities. This logical flow is evaluated in green, with the current step shown in orange in the top left-hand corner. As can be seen in this instance, the simulation has been structured to show that the satellite is not within tolerance limits for some value in its orbit, and has been instructed to perform corrective actions. The simulation has shown that it is operating in a logical manner when presented with this set of conditions, and as such the activities assigned to each swimlane would pass a logical check.

The activity library also carries with it a set of example mission activity profiles to supplement the basic housekeeping tasks. These example mission profiles are to give researchers the ability to start with a template, or at least a direction, to then refine and fit to their specific mission needs. Figure 7 shows an example for a communications hosting mission profile.

Figure 7. Communications Hosting Mission Profile



The activity diagram shows the progression of steps as the CubeSat performs its communications hosting activities to support the mission. The diagram includes an interruptible zone around the "Host Communications" activity, denoted by the dotted line with the header "Communications Breakdown." The interruptible zone serves to show that the "Host Communications" activity can be interrupted, and the logical flow of the diagram takes it to the "Communications Breakdown" activity profile as an automatic

response. This does not inherently imply the use of autonomy, but rather the existence of external factors that can cause a communications breakdown and predicate the need to address those factors.

This diagram is also a great example of the benefits brought to the modelling effort by the concept of multiplicity. Multiplicity is the number of instances of an element that can exist at any one time, and this can be either a finite or infinite number [8]. In this case the activity can be repeated across the many instances of the space vehicle block as it appears in the CubeSat constellation necessary to provide a communications capability. As the space vehicle block has an assigned multiplicity value (most likely equal to the constellation size), this activity will be shown as a mission behavior for each space vehicle block and does not require repeating. This is one of the functional benefits of this reference architecture, that it need not be repeated for a large-scale production effort; it must simply be maintained to ensure accurate version control.

The systems architecture from here is broken down into the Ground Control Segment and Space Vehicle Segments. The ground control segment contains the Ground Control Station block and its components, and the Space Vehicle Segment contains the Space vehicle, its components, and the framework for design. The scope of this reference architecture mainly focuses on the space vehicle given that the ground control station at AFIT is generally static and is not the topic of current student design efforts.

The space vehicle segment is divided into two portions, the component library and the design framework. The component library is composed of the set of subsystems that make up the space vehicle, and the space vehicle framework includes all of the

documentation that supports the space vehicle (requirements documents, concept of operations, etc.), allocated behaviors, analysis functions, interface diagrams, and the actual space vehicle block.

All of the space vehicle requirements can be placed into this framework, and allocated directly to the space vehicle block. The space vehicle block definition diagram is shown in Figure 8, and shows that the space vehicle is composed of an Attitude Determination and Control Subsystem (ADCS), Command and Data Handling Subsystem (CDHS), Thermal Control Subsystem, Power Control Subsystem, the Grissom bus, and the Selected Payload for the mission. This is refined by the Orbital Parameters and is supported by the Ground Control Station.

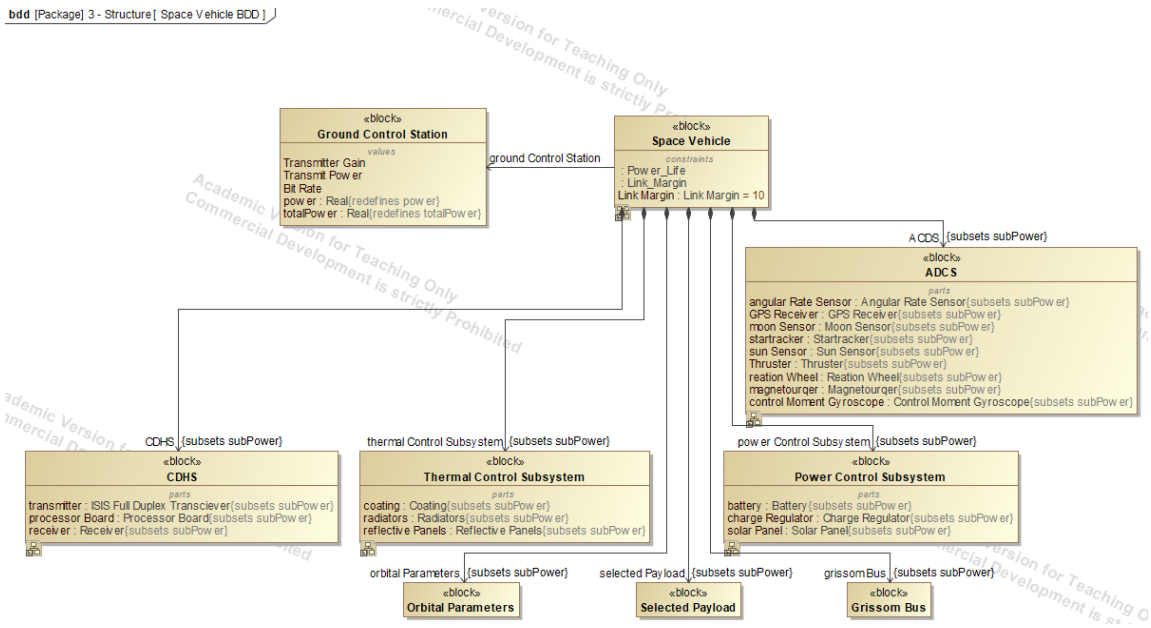


Figure 8. Space Vehicle Block Definition Diagram

The hierarchy shows the structural decomposition of the Space Vehicle, to include all subsystems. While this does not give a specific layout of how the subsystems shall be

integrated into the bus, it does show what subsystems are necessary to create a space vehicle in accordance with the best practices of space vehicle design from the AFIT Space Vehicle Design sequence. The space vehicle design is also refined by the orbital parameters that the satellite is going to be subject to. These characteristics give the regime in which the satellite will be operating, with values assigned for elements such as altitude, inclination, and proposed mission duration. All of these values will be open to ensure that the space vehicle is adequately described in its orbital regime, and there is a sufficient amount of descriptive date to use in the orbital parametric diagrams.

The Ground Control Station is only *associated* with the Space Vehicle, as can be seen by the straight arrow linking the two blocks. Whereas the rest of the blocks in Figure 8 are shown to compose the space vehicle (black diamond from Space Vehicle block connect to subsystem by arrow), the Ground Control system exists as a separate entity. The space vehicle is still able to interact with the ground station, but if for whatever reason the ground station were to cease operations the space vehicle would still exist as we know it, albeit with decreased functionality.

Extent of Component Repositories

The component subsystems outlined in the Space Vehicle Block Definition Diagram in Figure 8 are broken down further using elements from the component library. Each set of components is organized by its respective subsystem from the space vehicle; for example, all of the antennas for the satellite are housed under the communications side of the Command and Data Handling Subsystem. All of the subsystems are defined

by their own block definition diagram to mirror the hierarchy displayed in Figure 8. An example of a subsystem block definition diagram is given in Figure 9.

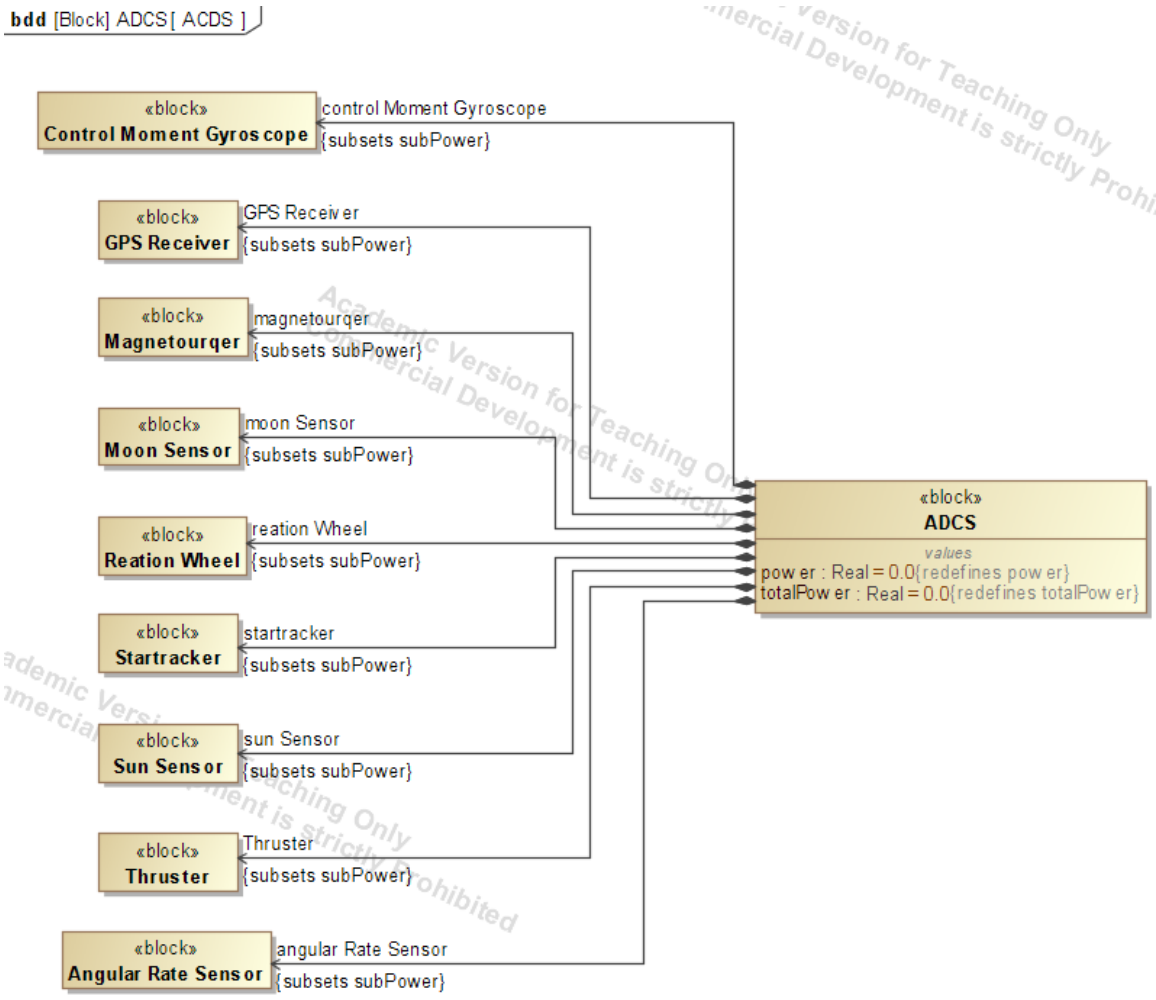


Figure 9. Attitude Determination and Control Subsystem Block Definition Diagram

The Attitude Determination and Control Subsystem (ADCS) is composed of many different components, and all of the different value properties associated with those components can be found within their respective blocks (they are hidden in the interest of readability). Each component is shown within the ADCS as a composition piece, however in all likelihood not every listed component here will be used. Knowing this, all

default values for all parameters are listed as zero in order to not confound any parametric equations or rollup calculations. As the user goes through and selects which specific componentry is going to be required within their mission specific build they will only populate the blocks which they intend to integrate into the vehicle. This will ensure the most accurate cost, mass, and power system totals are reached when the Space Vehicle is fully specified.

Each subsystem from Figure 8 has its own block definition diagram to denote the components inside. Each subsystem also has an associated content diagram to show the individual components, its value properties, and its ports. The ports on a component indicate interfacing connection points and are conduits for information and/or resource flow. While the interface documentation will be specific to each mission the ports will be relatively the same, so they are included in the model. The content diagram for the Command and Data Handling Subsystem is shown in Figure 10.

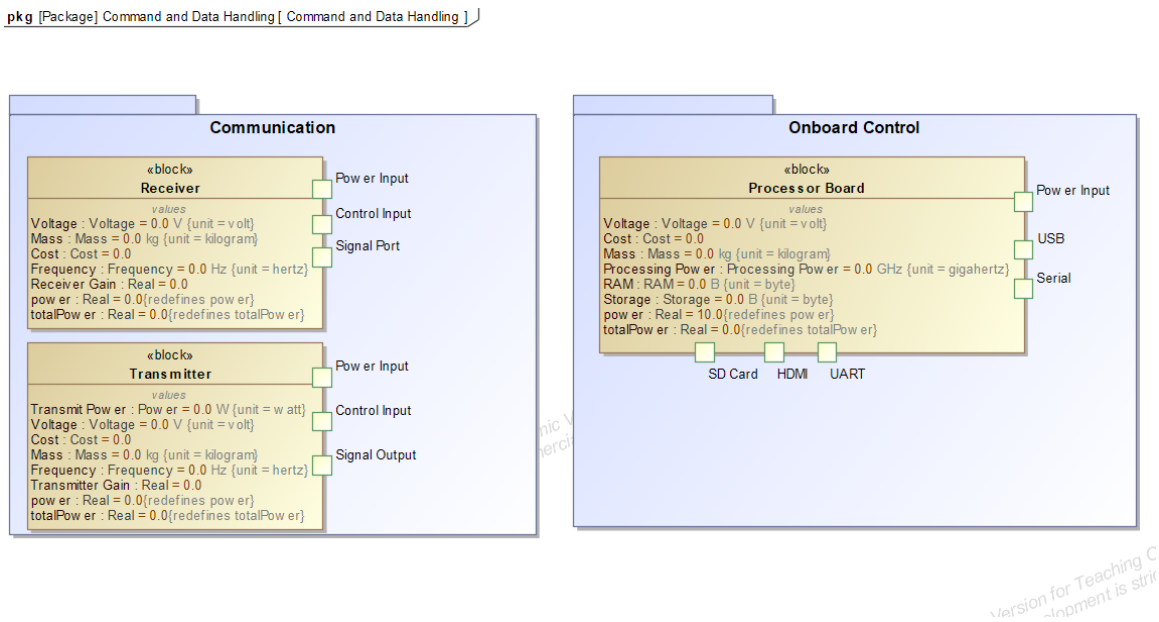


Figure 10. Command and Data Handling Subsystem Content Diagram

The ports in the content diagram are not connected to anything yet; the system architect shall make that design choice. A port can pass many things through it, such as power or data, and the architect shall designate which ports do which things. In this view it is plain for the architect to asses all of their value properties and ports, and make sure that they are logical and fit the mission needs that they have. All of the value properties are defined in common units, so that as the user fills out the value properties they are able to ensure that if anything must be passed from one port to another then there will be no unit related incongruencies. If a user wishes to change a unit, then all units in all other interfacing blocks must also be changed, or at a minimum identified whenever their respective ports are connected.

Use of Parametric Diagrams to Simulate Results

An extremely helpful benefit of using Cameo Systems Modeler is the ability to integrate other software to take on the bulk of the math engine. Beyond simple calculations, Cameo's math software becomes especially unwieldy to use and is not easily repeatable. However, Cameo allows for the integration of MathWorks' MATLAB, and this formed the basis for the parametric diagrams used in the CubeSat Reference Architecture. A parametric diagram incorporates the value properties, constraints, and parts of a component block and performs mathematical equations and inequalities to provide data to the user [8].

Parametric diagrams are a useful tool to show how a complex mathematical relationship can be modeled using the components that provide the values for the calculation. The components are identified in the diagram, and their relevant values are displayed within the components. A parametric equation is then written, which has a series of one or more inputs necessary to perform the calculation, and then a series of one or more outputs for the results of the calculation. The results of the calculation are typically assigned to a block as another type of value property for review or comparison with requirements and design constraints.

For the purposes of this architecture the equations were coded into MATLAB, since it is readily available at AFIT and the code is easily amended and shared. To link MATLAB into Cameo Systems Modeler is relatively straightforward, and once this is done simply dropping the function into the constraint block of the parametric diagram will automatically generate all input and output ports for the function to perform its

calculations. The MATLAB code used in the parametric diagrams in the architecture are contained in Appendix B.

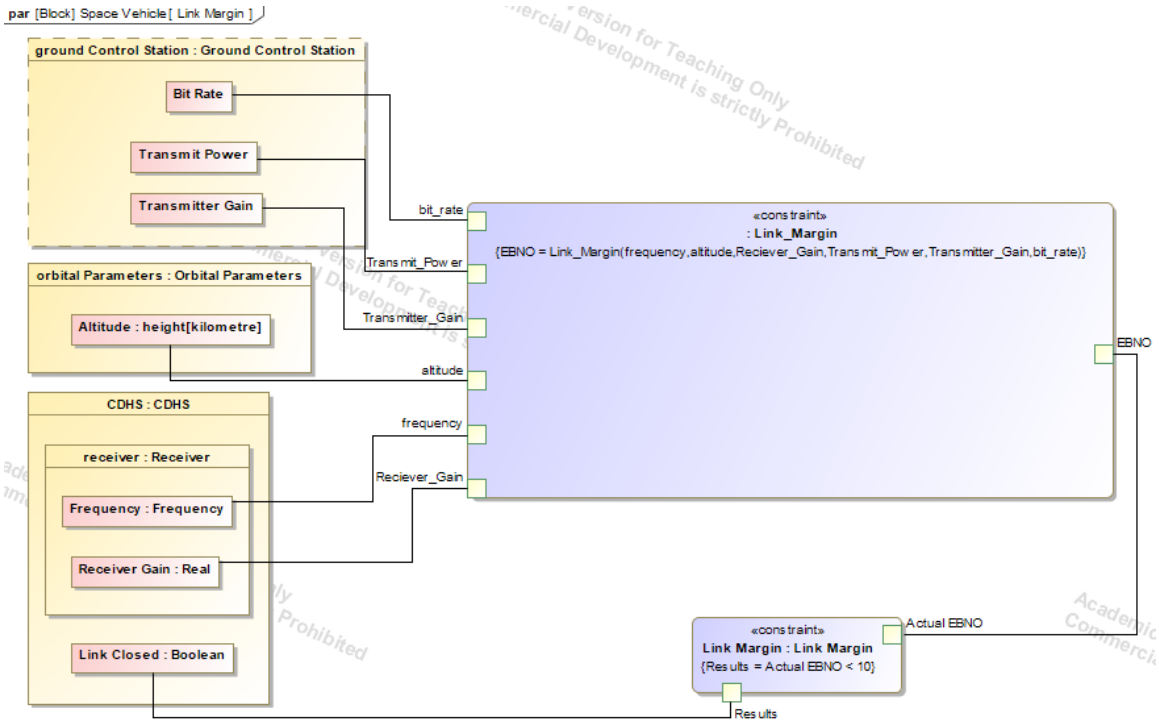


Figure 11. CubeSat Link Margin Parametric Diagram

The parametric diagram shown in Figure 11 details the information necessary to determine if the CubeSat has sufficient transmitting power to close a link with a margin of 10 dB, meaning that a satellite can experience 10dB of communication signal loss before communications are interrupted with the ground station. Since this calculation concerns both the space vehicle and the ground station, there are two different parametric port operations happening at once. In Figure 11 the block containing the Ground Control Station value properties is outlined in a dashed line to indicate that although the

parametric diagram and equation are owned by the Space Vehicle, the value properties are still compatible with the equation.

As all the value properties are input to the parametric equation, the resulting output is put into the second parametric equation on the right. This parametric equation is comparing the calculated E_b/N_0 (energy per bit to noise power spectral density ratio) to the required minimum of 10 dB. The parametric equation will then output a Boolean value to Space Vehicle Command and Data Handling Subsystem to show if the link will close. If the calculated E_b/N_0 is greater than 10, the parametric diagram outputs a value of 1, TRUE, to the CDHS and the user can see that their parameters for the associated components does allow the link to close.

The other parametric diagrams in the model follow this same flow of inputs and outputs. It is expected that the number of parametric diagrams in the model will grow as the need for continued satellite component calculation grows. The results of the parametric equations are kept in a set of instance tables held in the analysis portion of the Space Vehicle Framework. These instance tables allow for the user to test a whole range of configurations and component specifications, and then keep an itemized and dated list of all of the changes. The user can then look through these instance tables and highlight configurations that meet the constraints, and compare between the successful instances. This is a valuable tool in assessing the viability of specific components, and also for comparative component selection based upon performance.

Capturing Mission Inputs: Firefly Use Case

In order to verify that the designed CubeSat Reference Architecture is able to incorporate the inputs as described in Chapter III a sample test case was undertaken, based upon dissertation work to determine the viability of celestial navigation using manmade stars [17]. Based on efforts prior to a Preliminary Design Review undertaken by students at AFIT, the designed system came with a Concept of Operations, Space Vehicle Requirements, Stakeholder Needs, Verification Framework, Functional Architecture, Behaviors, Orbital Parameters and Constellation Design documentation. In short, the mission was completely designed, but it was component and hardware agnostic.

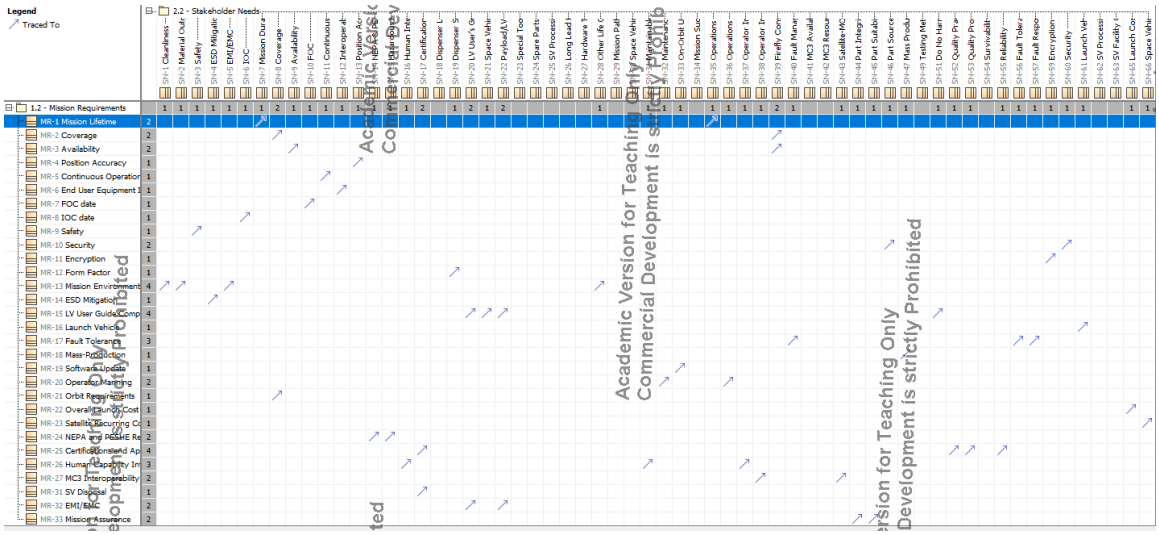


Figure 12. Mission Requirements Traced to Stakeholder Requirements

All of the mission requirements and operational needs were traced to their stakeholder needs, ensuring that traceability was maintained throughout the whole process. The next steps to be taken using the reference architecture would be to piece together some componentry that would meet the requirements of the system while still

satisfying the mission specific functional behavior described in the concept of operations. This mission specific behavior is can be detailed in diagrams such as a State Machine Diagram, shown in Figure 13. The State Machine Diagram gives solution agnostic descriptions of how the satellite is supposed to behave, and is a visual description of the CONOPS.

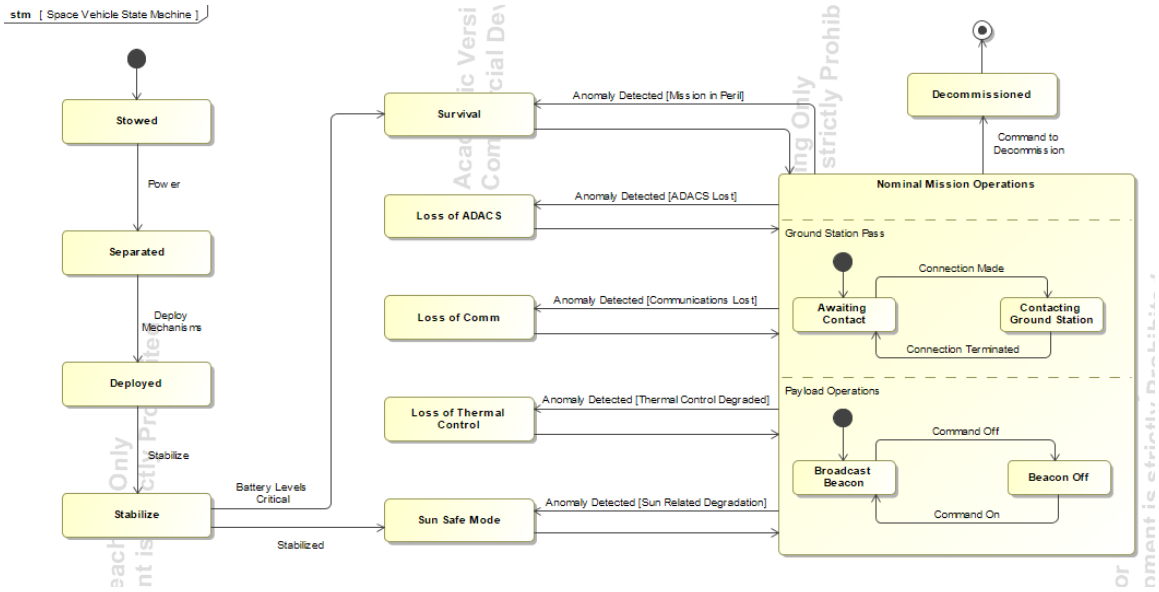


Figure 13. Project Firefly State Machine Diagram

With all of the mission engineering finished, components and hardware needed to be selected and integrated into the model. The components and hardware for this mission had already been selected after the Preliminary Design Review, and further efforts show that these components are in fact able to be integrated into the model and have the requirements allocated to them. All of these efforts were captured inside the reference architecture, and now they can be used to document the utility of parametric diagrams in the reference architecture.

Parametric diagrams in the reference architecture all have a similar structure, shown in Figure 11, that allows users to take a set of system value properties and perform calculations on them to ensure that requirements are being met and constraints aren't being broken. In this case, the requirement needing to be satisfied is "The Firefly Space Vehicle shall have a positive link budget with a link margin greater than 6 dB." As long as the link margin is greater than 0 the CubeSat will be able to communicate with the ground station, but an acceptable safety factor is added to ensure no loss in communications, typically 6-10 dB required link margin [3].

Table 8. Insufficient Link Margin (Space Vehicle to Ground Station) Parameters

Model Element	Parameter	Unit
Orbital Altitude	700	Kilometers
SV Transmitter Power	4.77	Decibel-Watts
SV Transmitter Gain	1	Decibels
Frequency	455	Megahertz
Bitrate	4000	Bits-per-Second
GCS Receiver Gain	10	Decibel-Watts
GCS Antenna Noise	150	Kelvin
Noise Factor	7	Decibels

If the link margin is less than 6dB then the output of the parametric diagram is a Boolean result printed to the CDHS block, and this will show FALSE in an instance table. The instance table in Figure 14 shows what the user will see in the ISIS Full

Duplex Transciever row if the parameters involved do not meet a link margin constraint of 6dB:

Classifier	Link Closed : Boolean	Bit Rate : Real	Altitude : height[kilometre]	Frequency : Frequency
Orbital Parameters			700 km	
Ground Control Station		4000		
ISIS Full Duplex Transciever	☐ false			4.55E8 Hz
Processor Board				

Figure 14. Insufficient Link Margin Instance Table

If the values of the respective blocks are changed to provide a satisfactory link margin the instance table will show a Boolean value of TRUE to indicate to the user that this specific instance will satisfy the constraint. In this instance the Orbital Altitude was decreased, the Frequency was increased, and the Ground Control Station Receiver Gain was increased.

Table 9. Sufficient Link Margin (Space Vehicle to Ground Station) Parameters

Model Element	Parameter	Unit
Orbital Altitude	621	Kilometers
SV Transmitter Power	4.77	Decibel-Watts
SV Transmitter Gain	1	Decibels
Frequency	940	Megahertz
Bitrate	4000	Bits-per-Second
GCS Receiver Gain	18	Decibel-Watts
GCS Antenna Noise	150	Kelvin
GCS Noise Factor	7	Decibels

These values are kept in the instance table for future reference if the user still wants to work on the component value properties.

Classifier	Link Closed : Boolean	Bit Rate : Real	Altitude : height[kilometre]	Frequency : Frequency
Orbital Parameters			621 km	
Ground Control Station		4000		
ISIS Full Duplex Transciever	☑ true			9.4E8 Hz
Processor Board				

Figure 15. Sufficient Link Margin Instance Table

Summary

This chapter outlined the total hierarchy of the CubeSat Reference Architecture, and outlined the Organizational Level Architecture and the System Level Architecture. Both levels of architecture were explored and their contents detailed, with specific attention shown to the flexibility of the model elements to adapt to the needs of the specific mission sets of the user. The activity, language, and component libraries were explored and their element hierarchies were outline to give clarity to the interconnected nature of the reference architecture. The concept of readily available simulation in the form of parametric diagrams was discussed, as well as the usefulness of instance tables.

V. Conclusions and Recommendations

Significance of Research

Any model that is created has some form of architecture attached to it, intentional or otherwise. If the architecture is organized and planned with guided intent it can save significant amounts of time for the user. AFIT has a systematic approach to teaching students how to design a mission in space and how to create a hardware solution to meet the designed mission. The AFIT Space Vehicle Design sequence does not happen in a vacuum, and the time needed to create viable missions and spacecraft is extremely valuable. AFIT students are expected to be taking other classes and working on other assorted research ventures at most points during the sequence. Any time saved on the design of the architecture of the system is time that can then be reapplied for better mission understanding, component testing, hardware selection, and so on.

The time scale necessary to create this reference architecture is now proportional to the amount of time saved by users not having to create their own system architecture, and this can be reinvested into better system design. Furthermore, having a consistent and reusable architectural baseline to apply over a multitude of potential mission scenarios gives the AFIT Space Vehicle Design sequence wide latitude to capture knowledge gleaned from the many iterations of the design sequence. As classes come through AFIT and more students use the architecture, it will continue to grow and adapt to fit the future needs of the institution.

An adaptable and evolving architecture is a necessary capability. The state of the art in the space domain is increasing exponentially, and this reference architecture is

capable of growing with time and incorporating the advances of technology, standards, policy, and common practices/techniques. The reference architecture is an open-ended tool, and while it enables users to take the system design in many different directions it also has a few elements that are specific to AFIT. These elements are readily extensible to all parties interested in developing CubeSats, and there is still value in the ability of the model to show system level and organizational level hierarchies and frameworks for use by non-AFIT entities.

Investigative Questions Answered

Chapter I introduced three investigative questions to guide the research efforts throughout this process:

1. *How can engineers reduce the design time of a desired mission and system?*
2. *Is it possible to produce traceable and defensible system designs on a consistent basis?*
3. *Is there a way to accelerate the learning process involved with Space Vehicle Design?*

The first question is aided by the inherent repetition found in the CubeSat development process. Since many of the events and process steps are similar from one build to another it made sense to create a notional model of a CubeSat mission that could easily be expanded upon with the mission specific details. Giving this advanced starting point to engineers is a quick and useful manner in which to reduce the design time for a desired mission and system. The foundation of the framework is capable of hosting a wide variety of missions and builds, and the lessons learned from these builds can be

reincorporated to ensure that subsequent users remain on the forward edge of the development process.

Given that the framework has already been created for the engineers, it gives an integrated and intuitive workspace to create a fully traceable system design. All of the elements in the framework are related to one another, and additions and changes can be related to show the interconnectedness of the system. These relations form the network of traceability from requirements to components, capabilities, and uses of the system. Being able to show how a set of mission needs resulted in a specific system is valuable for the design team to prove that the system they have built is the correct system for the stakeholders and their mission, and the Reference Architecture that was built is the tool with which all of these artifacts are captured and displayed.

The advanced starting point given to users is also a catalyst for the increased comprehension of the general space vehicle design process and understanding of the necessity to integrate clear and fluid systems engineering in a mission design. These factors being impressed upon the students from an early stage in the educational process truly accelerates their learning and results in the production of more knowledgeable space professionals within a decreased timeframe. Ultimately the creation of a Reference Architecture for the CubeSat domain was the simultaneous answer to all of these questions, and gives users the capability to rapidly produce traceable and defensible designs while enhancing the learning outcomes of the space vehicle design process.

Limitations

Currently the reference architecture has not been used to build a new mission from the start of the Space Vehicle Design Sequence. The system was validated using prior data to show that the system could produce the desired outputs, but the componentry was already selected to create these outputs. The process of testing, failing, and retesting components was not able to be captured as a result, and the functionality of model elements such as instance tables were not able to be explored outside of this static environment. These elements will be tested in dynamic environments in the near future at AFIT.

Recommendations for Future Research

Incorporating the reference architecture into the AFIT Space Vehicle Design sequence is the primary objective moving forward. Use of the reference architecture to facilitate the introduction of key systems engineering concepts and best practices to new students is essential, and early introduction (after mission engineering has happened) would be of the most benefit to the students.

What benefits the architecture has also have to be measured with the reality the architecture needs "operational testing." It needs to be validated by using dynamic data points, subject to the uncertainty inherent to real-time system design. Once the reference architecture is used to capture componentry and system testing in a space vehicle design and build sequence the ability of the reference architecture to show changes over time would be put to the test, and this level of stress would reveal the capability to capture the needs of the students. Thought needs to be given to establishing a set of measures to

determine the utility and effectiveness of a reference architecture in CubeSat design. In addition to this, there could be added benefit in comparison of the reference architecture to current satellite architectures and subject matter experts in the field. This would be a powerful step towards validating that the initial content of the reference architecture is structurally capable of becoming a full fledge satellite architecture.

Furthermore, as different groups of students within the space vehicle design sequence are subject to different constraints the model would be able to be taken in multiple directions at once. This would be the best way to validate the ability of the reference architecture to be used in concurrent projects, especially with different groups using the architecture to model different mission sets. Using the architectural framework to model multiple missions, knowing that these missions came from the same originating architecture, should yield many artifacts for the architecture to grow upon.

The Small Unmanned Aircraft System Reference Architecture is beginning to incorporate elements of autonomy into its capability suite, and this is another direction that could be taken with the CubeSat Reference Architecture [13]. While the benefits are numerous, autonomy is difficult to implement well in many situations, and is even more so in space. Modeling autonomous functions, behaviors, and states within the architecture would be beneficial to understanding the manner in which it could reliably be created and implemented within the space domain.

A long-term research goal would be the integration of a visual modelling software such as Systems Tool Kit (STK). STK can create space vehicles and other domain elements to visualize orbits, sensors, location, and many more aspects related to satellite

operations. Incorporating template STK models into the reference architecture, and then using STK to model the way that a satellite moves and operates based upon different parameters would be a valuable capability to show changes in a three-dimensional representation of the satellite.

The biggest future research opportunity for the architecture is to become the basis for a mission that will actually be designed, built, and launched using the Grissom bus. Having an AFIT CubeSat mission use this reference architecture "cradle to grave" would be an ideal use of the tool, and would provide the users with plenty of capability to maintain a model of the system and the mission at all times. This model could be given to stakeholders and operators to increase system understanding, and would allow for increased system knowledge from all parties.

Summary

Designing a CubeSat begins with understanding the mission that needs to be performed and the functions needed to perform that mission; only then can the process move on to capturing hardware decisions. A CubeSat Reference Architecture gives the ability to connect mission definition elements to hardware choices, and provides clear traceability between the two elements. The Reference Architecture captures all of the architecting knowledge at AFIT and forms the architectural baseline for CubeSat missions both within AFIT and in the general CubeSat design community. The Reference Architecture is keeping in practice with Model Based Systems Engineering best practices, and has the capability to evolve and grow in order to facilitate innovation and rapid development within the CubeSat domain.

Appendix A. CubeSat Reference Architecture AFIT User's Guide

Welcome!

This is not intended to teach you how to use Cameo Systems Modeler, rather to focus and guide your efforts in designing a viable spacecraft platform using a reference architecture. To learn how to use Cameo, please refer to ASYS 531 laboratory lessons. There are a few documents that you need to have before you start using the CubeSat Reference Architecture, primarily the Stakeholder Analysis Documentation. This includes:

- *Stakeholder Needs Document (SND)*
- Derived *Stakeholder Requirements Document (SRD)*
    - Optional: *Space Vehicle Requirements Document (SVRD)*
- *Measures of Effectiveness (MOE)*
- *Technical Performance Measures (TPM)*
- *Key Performance Parameters (KPP)*
- *Constraints* (budget, schedule, etc.)
- *Concept of Operations (CONOPS)*

All of these documents should be products of ASYS 531, Space Mission Analysis and System Design. Furthermore, ASYS 531 students should also have first attempts at creation of a functional, physical, and interface architecture, as well as potential Verification and Validation concepts. All of these extra elements will add into the

reference architecture nicely, and will only give a greater starting point for system design. Some additional elements such as State Machine Diagrams, mission specific Activity and Use Case Diagrams, and Block Definition Diagrams are also useful, but not necessary. If the above documents have been completed and signed off by the sponsor then you are ready to go!

- Using the Content Diagram the model opened up to (where you probably saw this User's Guide), right-click the "Architectural Guidance" package inside the Organizational Framework on the left
- Click "Select in Containment Tree." This is showing you where to place the *SND.*
- Right click on the "Requirements" package inside the System Model and place the *SRD* (and *SVRD* if you have it) inside.

Use this opportunity to trace the Stakeholder Requirements Document to the Stakeholder Needs document.

- Right click on the *SRD* and select "Specification,"
- When this pane opens up select "Relations."
- The relations page will give you the option to create an outgoing relation, select this and then select the "Trace" option.
- A pop-up of the model containment tree will now show, select the "Architectural Guidance" package that you just put the *SND* inside of to trace the *SRD* to the *SND.*

Maintaining requirements traceability is essential to performing verification and validation, and this traceability will be able to eventually prove that all requirements have been satisfied in one way or another. Be sure to trace every requirement to the need that it was derived from. You will use this same method to satisfy requirements by connecting the satisfying model element to the requirement.

- Right click on the model element that is satisfying a particular requirement
- Follow the same procedure as outlined above to get to the relations page
- Create an outgoing relation, this time select “satisfy”
- Connect the outgoing relation to the requirement that the model element is meeting

All of these relationships can be shown in a requirements satisfaction table for verification and validation purposes. A relationship table of any kind (trace, allocate, refine, satisfy, etc.) can be shown to create a summary of all of the incoming and outgoing relationships and can be key in understanding how the foundational documentation of the satellite resulted in specific design choices being made.

Next it will be up to you to turn the *SRD* and *SVRD* into requirement model elements in Cameo, if you have not already done so in ASYS 531. In addition, you must go through the Laws, Policies, Regulations, and Technical Standards repositories and parse those documents to get your regulatory set of requirements for the space vehicle. Place all of these requirement elements into the “Requirements” package inside the Space Vehicle Framework of the System Model. Trace these requirements to the block that the document is attached to, not the document itself. This package is associated with the

Space Vehicle, so all requirements will tie into that block. Also put the *CONOPS* into the "Concept of Operations" Package, separate from the requirements package. The requirements from the *CONOPS* simply need to be traced back to the CNOPS. You can check up on your progress and continually monitor it by creating a requirements table, it will be helpful to keep this table and have a single diagram to look at to check your requirements.

Once all of the requirements are created, outlined, and traced, go ahead and take the designed mission parameters (altitude, duration, inclination, etc.) and input them into the "Concept of Operations" Package using the "Orbital Parameters" block. Value properties have already been created for the most common orbital parameters, thes contain a default value of zero. You must input your desired mission parameters to the value property slots that match each parameter. These parameters will be used for parametric analysis later.

Now you may begin working on the functions of the space vehicle. Create activity diagrams and use cases to show what is expected of the vehicle. There are already activity diagrams and mission profiles for you to choose from in the Activity Library. Use these as you wish, or expand upon them to fit your mission need. Describing the activities and functions of your system may drive you to realize that there are more requirements than you originally thought. As you create your activities add new requirements to your list as necessary. Don't forget to trace!

Once your activities are created you may move on to the components of your space vehicle. As you have developed your knowledge of the different subsystems of a

CubeSat throughout ASYS 631 you will have arrived at certain components that lend themselves best to the mission you have been tasked with. Looking at the component library you can see that it is broken up into the relevant subsystems. Go into each subsystem and fill in the value properties for each component that you intend to have on your CubeSat. As before, the default values of these components are zero, so if there is a component in the library that you do not plan to use it will not affect any calculations. The payload section of the component library contains a blank payload block that is connected to the space vehicle. There are several example payloads to guide your choices as you fill in the blank payload block, but you must fill in the blank payload in order for it to connect with the space vehicle block.

All of the components link together, and this can be seen in the Space Vehicle Block Definition Diagram under Section 3 of the Space Vehicle Framework. As you finish filling in all of your componentry you can now step out of the space vehicle and go to the Ground Control Station Block to ensure that all of the information as it pertains to ground system equipment is up to date. This will be important to perform parametric calculations using the most accurate representation of the equipment at AFIT.

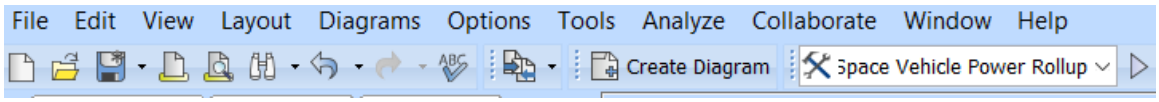
Once you have identified your componentry you need to ensure that any technical requirements are being met. If you have requirements that relate to performance of a certain subsystem or component then you need to create that relation at this time.

- Right click on the component that is satisfying a particular requirement
- Go to the element specification, and click on relationships
- Create an outgoing relation, this time select “satisfy”

- Connect the outgoing relation to the requirement that the Space Vehicle component is satisfying

Now that you have specified componentry for the Space Vehicle you can run simulations upon your satellite to see if the specific values of your components meet requirements. These simulations are already pre-configured, all you need to do is select the simulation configuration control at the top of the model.

Figure 16. Simulation Configuration Management



In this case the simulation is configured for a Space Vehicle Power Rollup, and you can click on the drop-down arrow to change the configuration each parametric equation. Simulations are performed upon the space vehicle as a whole, and are executed through the parametric equations attached to the space vehicle. These parametric equations are started by pressing the play button next to the simulation configuration selected, and a window at the bottom of the Cameo screen shall pop up to show you how the simulation is resulting. In this case, the simulation will show how much total power the space vehicle will use in a given configuration. This configuration is then saved to an Instance Table, found in Section 8 of the Space Vehicle Framework. These instance tables can be used to show iterations of the components of the space vehicle, and is a

great way to show how failures or successes in the analysis of the components of the space vehicle have driven change in the system.

You have now used the reference architecture to create and outline your own Space Vehicle. This is not a start to finish guide, many of the processes identified must be iterated, often frequently, to arrive at an acceptable solution that satisfies all requirements. However, by using this reference architecture you are putting yourself head and shoulders above the standard document based approach, and you have a solid foundation upon which to stand as you move forward with this system.

If you have created any of your own parametric diagrams, please look to upload them (with a simulation configuration) to the master copy of the reference architecture so that others may benefit. Any other lessons learned are also greatly appreciated. This reference architecture is a living, breathing, and evolving representation of all of the systems knowledge here at AFIT, and it is up to you to keep it in a position where it is able to allow you to innovate. If you do decide to upload your findings and lessons learned into the diagram, please identify the updates that you have made and ensure that you do not disrupt the normal functionality of the model when you include your revisions.

Appendix B. Parametric Diagram MATLAB functions

Function 1: Satellite Link Margin (Link_Margin.m)

```
function [EBNO] = Link_Margin(frequency,altitude, Reciever_Gain, Transmit_Power,
Transmitter_Gain, bit_rate )

%Link Margin Calculation
%   altitude must be in km, frequency must be in GHz, Receiver_Gain is in
%   dB, Transmit_Power is in dB, Transmitter_Gain is in dB, bit_rate is in
%   bits per second

% Free Space Losses
        sin_roh = sind(6371/(6371+altitude)); %roh is the angular radius of earth
        cos_eps = (sind(30)/sin_roh); %spacecraft elevation angle
        lambda = 90-30-acosd(cos_eps);   %nadir angle+angular radius+elevation angle
=90deg

        transmit_distance = 6371*(sind(lambda)/sind(30)); %in km

        Ls = 92.45 + 20*log(transmit_distance) + 20*log(frequency); %Free Space Loss
in dB form

% At this point calculate Atmospheric and Rain losses based upon mission
% parameters. From here on out they will be calculated as zero, given that
% free space loss is the largest loss factor and the other loss factors are
% usually negligible on a clear day

        La = 0;  %atmospheric loss (update using log chart based on your specific
mission)
        Lr = 0;  %rain loss (update using log chart based on your specific mission)

        Lcomb = Ls + La + Lr; %Combined Losses in dB form

        System_Temp = 21.3; %given in SMAD Table 13-10, units in dB-K

        GT = Reciever_Gain - 10*log(System_Temp) ; %G/T Ratio in dB

        EIRP = Transmit_Power + 0.5 + Transmitter_Gain; %Equivalent Isotropic
Radiated power in dB

        Carrier_to_Noise = EIRP + GT - Lcomb +228.6 %carrier to noise ratio in dB

        Rb = 10*log(bit_rate); %converts bit rate from bits per second to dB-Hz
```

```
        EBNO = Carrier_to_Noise - Rb; %gives Eb/No predicted, will now compare to
give link margin
end
```

Function 2. Solar Array Area Validation (Power_Life.m)

```
function [Solar_Array_Area,Power_Beginning_Life,Power_End_Life,Te,Td] =
Power_Life(Power_Required,Orbit_Altitude,
Mission_Length,Solar_Efficiency,Degradation_Rate)
%   Luke Farrell Nov 7 2019
%   Gives beginning and end of life solar array power production density per square
%   meter, required area of the solar array given power needs, and the time
%   in eclipse (Te) and the time in daylight (Td) per orbital period
%   Orbit Altitude in km, Mission Length in Years, Solar efficiency is a percentage,

   lambda = rad2deg(asin(6378.1/(Orbit_Altitude+6378.1))) ; %frequency
   Orbital_Period = 2*pi()*sqrt(((Orbit_Altitude+6378.1)^3)/398600) ; %orbital period in
seconds
   Te = (2*lambda)/360 ; %time in eclipse in seconds
   Td = Orbital_Period-Te ; %time in daylight in seconds

   Xd = 0.65 ; %power efficiency getting from solar panels to battery to load in daylight
   Xe = 0.85 ; %power efficiency getting from battery to load in eclipse
   Power_From_Solar_Array =
(((Power_Required*Te)/Xe)+((Power_Required*Td)/Xd))/Td ; %total amount of power
needed per orbit from solar panels

   Id = 0.72 ;  %inherent degredation
   P0 = 1358*Solar_Efficiency  ;  %W/m2, power density output standard
   ang = deg2rad(23.5);    %worst case sun incidence angle
   PBOL = P0*Id*cos(ang);      %Power density per square meter at beginning of life
   Ld = (1-Degradation_Rate)^Mission_Length;   %lifetime degradation
   PEOL = PBOL*Ld;     %Power density per square meter at end of life
   Solar_Array_Area = Power_From_Solar_Array/PEOL
   Power_Beginning_Life = PBOL ;
   Power_End_Life = PEOL ;
end
```

Bibliography

[1] California Polytechnic, "CubeSat Design Specification (CDS)," CalPoly, San Luis Obispo, CA, 2014.

[2] AFIT Graduate School of Engineering and Management, "Air Force Institute of Technology," 28 August 2019. [Online]. Available: https://www.afit.edu/docs/19-20%20Catalog.pdf.

[3] J. R. Wertz, D. F. Everett and J. J. Puschell, Space Mission Engineering: The New Space Mission Analysis and Design, Hawthorne, CA: Microcosm Press, 2011.

[4] Space Exploration Holdings, LLC, "Federal Communications Commission Licensing," 15 November 2016. [Online]. Available: https://licensing.fcc.gov/cgi-bin/ws.exe/prod/ib/forms/reports/swr031b.hts?q_set=V_SITE_ANTENNA_FREQ.file_numberC/File%20Number/=/SATLOA2016111500118&prepare=&column=V_SITE_ANTENNA_FREQ.file_numberC/.

[5] D. M. Buede and W. D. Miller, The Engineering Design of Systems: Models and Methods, Hoboken, New Jersey: John Wiley and Sons, 2016.

[6] California Polytechnic, "6U CubeSat Design Specification," NASA, San Luis Obispo, 2016.

[7] S. Mandutianu, M. Mehrdad and K. Donahue, "Conceptual Model for Space Mission Systems Design," Jet Propulsion Laboratory, Pasadena, California, 2009.

[8] L. Delligatti, SysML Distilled: A Brief Guide to the Systems Modeling Language, Upper Saddle River, New Jersey: Addison-Wesley, 2014.

[9] No Magic, Inc, "Cameo Systems Modeler User Guide," No Magic, Inc, 2020.

[10] R. Cloutier, G. Muller, D. Verma, R. Nilchiani, E. Hole and M. Bone, "The Concept of Referece Architectures," Wiley Interscience, Hoboken, New Jersey, 2008.

[11] M. W. Maier and E. Rechtin, The Art of Systems Architecting, Boca Raton, Florida: CRC Press, 2009.

[12] US Army RDECOM, "Comprehensive Architecture Strategy," US Army, Aberdeen Proving Ground, Maryland, 2018.

[13] D. Jacques and A. Cox, "The Use of MBSE and a Reference Architecture in a Rapid Prototyping Environment," Air Force Institute of Technology, Dayton, Ohio, 2019.

[14] D. Kaslow, B. Ayres, P. Cahill, Hart, Laura and R. Yntema, "Developing a CubeSat Model-Based System Engineering (MBSE) Reference Model - Interm Status #3," in *IEEE Aerospace Conference*, Big Sky, Montana, 2017.

[15] National Aeronautics and Space Administration, "CubeSat 101: Basic Concepts and Processes for First Time CubeSat Developers," NASA CubeSat Launch Initiative, San Luis Obispo, California, 2017.

[16] Office of the Secretary of Defense, Networks and Information Integration (OASD/NII), "DoD Reference Architecture Description," Office of the DoD Chief Information Officer, 2010.

[17] S. J. Pierce, "Modeling Navigation System Performance of a Satellite-Observing Star Tracker Tightly Integrated with an Inertial Measurement Unit," AFIT Dissertation, Dayton, 2015.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
21-02-2020	Master's Thesis	March 2019 – March 2020

TITLE AND SUBTITLE

A Reference Architecture for CubeSat Development

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Farrell, Luke J., Second Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/ENV)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENV-MS-20-M-199

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
DISTRUBTION STATEMENT A. APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES
This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States.

14. ABSTRACT

Developing a space vehicle is a complex and detailed process, and while CubeSats are smaller and more accessible than traditional satellites the design process is relatively unchanged. Creating a viable space vehicle design requires detailed analysis of a set of mission needs in order to define the mission, with this need set used to then create the specific mission requirements. These requirements are used to formulate a concept of operations, and then move into developing a physical system for executing the mission. The successful production of CubeSats within an organization is contingent upon the accurate execution of the general CubeSat Development Process.

This research presents a tool to facilitate more complete, streamlined, and transferable products throughout the course of a general CubeSat Development Process. The reference architecture is capable of displaying both organizational and systems level architectures, both linked together and in support of consistent and repeatable structure to be given to users intending to produce a complete mission and system design. The architecture incorporates a suite of repositories to assist users in hardware integration and requirements traceability, including component, activity, and regulatory libraries; in addition to parametric diagrams to facilitate requirements verification and constraint analysis.

15. SUBJECT TERMS
Cubesat, Reference Architecture, Model Based Systems Engineering, MBSE, System Architecture

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			David R. Jacques, AFIT/ENV
U	**U**	**U**	**UU**	83 95	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-3636 x3329