Spring 2012

Dynamic model-based systems engineering using XML metadata interchange

Dustin Scott Nottage

Department:

Dynamic Model-Based Systems Engineering using XML Metadata Interchange

by

DUSTIN SCOTT NOTTAGE

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

2012

Approved by

Steven Corns, Advisor
Henry Pernicka
Elizabeth Cudney

ABSTRACT

Designing systems presents an engineer with numerous design choices based on multiple conditions and constraints. Base camp planning must take into account the number of soldiers, the permanency of the base, the location of the base, etc. To help alleviate the complexity of constructing a base camp, a model-based systems engineering approach is used. This method creates and integrates models for all of the facilities that can make a base camp, as well as interactions between facilities and required utilities for each. The goal is to have a camp design, and then solve a system of equations to solve for the total resources required for the specified camp. The issue that arises is that the models are static and non-executable. The proposed method takes the components and values of the model and exports them in an XML format. The information in the XML file is parsed to extract the relevant information so it can be used in an analysis application. The application results provide the total resources required for camp and facilities. The XML file is then updated with the analysis results, and imported back into the model. This essentially makes the model executable and dynamic. This approach is also successfully applied to a satellite design process. The goal of this work is to use the information in a virtual engineering toolkit, with the toolkit integrating multiple analysis tools, to achieve a fully executable design architecture.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr. Steven Corns, for his valuable guidance, encouragement and support throughout my graduate studies. I would also like to thank Dr. Hank Pernicka and Dr. Elizabeth Cudney for their input and participation on my committee. Special thanks for Dr. Pernicka for his support and motivation through my undergraduate and graduate studies.

I would like to express my gratitude to my family for all of their love and support. Finally, I would like to thank my daughter for keeping me entertained when times were tough, and giving me the motivation to continue in my studies.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Page

1. INTRODUCTION

Model-Based Systems Engineering (MBSE) moves the document-focused approach into a single, computer model approach which supports analysis, specification, design, verification, and validation of complex systems [1]. Advantages of this method are better communication between people working on different subsystems and the ability to easily reuse designs. A. Wayne Wymore introduced the concept of model-based systems engineering in 1993 with his book on the subject [2]. It is only recently that the concept has been coming to fruition, especially as the computer-aided approach. Computers are now powerful enough, and can store enough information, to be a viable option for system design and analysis. Much of system simulations and analysis is done using computer software now. It would be beneficial to be able to use information in a system model in those different analysis programs. There would be no need to duplicate efforts of inputting information. Also, the system specifications, analysis, and results would all be connected in the same place.

Base camps are locations outside the United States that support military forces for combat and peacetime missions. They provide all the equipment, facilities, and personnel required to support a specified number of troops on various missions. No two base camps will be exactly the same. Variables to consider when designing a base camp include location, mission, duration, and troop population. Changes to any of those parameters change the structure and requirements of a camp, so planning those facilities is an iterative process. The population is affected as the number of facilities is increased, and the number of facilities is affected by a population increase. Some facilities require personnel to operate them, and those extra personnel add to the total population that needs to be provided for. This increases the strain on the current facilities. Additional facilities may need to be brought in, possibly increasing the number of personnel on the base to keep it operational. Taking into account the need to provide for the support personnel adds to the degree of difficulty in planning a base camp.

Each facility type will have different utility requirements. Knowing the total resources required to keep a camp operational will allow the logistics to be planned and set up to keep the camp supplied. These numbers are also hard to pin down. There are no

solid numbers published anywhere. There are estimations for some utilities like water consumption, but could vary from 25 to 60 gallons of water per day per soldier depending on the source [3]. Those are just overall estimations too. This project also requires utility estimations for each individual facility. This gets very tricky at the larger camps with a Post Exchange where soldiers can buy consumer electronics. Soldiers could have televisions and videogame consoles in their housing. Each housing barrack would then have a different power requirement depending on soldier activity within. Some utility information is not published for security reasons. So, many of the numbers that could not be found were estimated and verified by people familiar with operational camps.

The difficulty and number of variables to consider leads into using a Systems Engineering design approach. Because of the large number of computer models that must be integrated, a Model-Based Systems Engineering approach is used. In the model-based approach, camps can be designed easily and quickly. Different variations to the camp can be modeled and compared against each other. This allows planners to conduct trade studies on different camp designs. Also, many of the scenarios to which a base camp is subjected can be modeled and analyzed before construction starts. With all the information already on a computer format, computer-aided analysis tools could be employed. However, MBSE tools are typically just static information models. There is no way to perform significant analysis on a model without using other software. A method is needed that takes the static information from the model and allows an analysis tool to use it and then update the static information based on the result.

In order to perform analysis on the camp design, model information is input to an additional analysis tool. There are numerous methods for the transfer of information [4]. The method used in this research is through the XML Metadata Interchange (XMI). XMI provides a standard format for exchanging information across tools using the Extensible Markup Language (XML). It defines the representation of objects, standard mechanisms for linking objects, object identification, and validation using XML Schemas [5]. The analysis that must be performed is solving a system of equations for obtaining the total resources required for the entire base camp, and the individual facilities. This is performed using a graphical user interface developed specifically for the base camp

project and this research. The results are then transferred back into the modeling tool, again using XMI.

Workshops have been conducted to get a knowledge management system in place so information can grow and evolve instead of remain static. The ArmyBaseCamp/JFOB.net knowledge management system contains information from all branches of the military on base camps through the use of briefings, interviews, documents, books, best practices, and policies [6]. With the knowledge management system and the ability to store system models in a repository, collaboration among different planners is now possible. One issue currently being addressed by the military is passing on the knowledge of base camp planning. A new planner might have to go through numerous learning experiences that senior planners have already went through.

2. MODEL-BASED SYSTEMS ENGINEERING

The MBSE Initiative was started in 2007 during the International Council on Systems Engineering (INCOSE) International Workshop. As part of the INCOSE SE Vision 2020 statement, MBSE is "part of a long-term trend toward model-centric approaches adopted by other engineering disciplines…(and) is expected to replace the document-centric approach…by becoming fully integrated into the definition of systems engineering processes [7]." The MBSE environment is made up a modeling language, tools, methods, and way to incorporate them all. There are multiple MBSE methodologies that have been developed and adopted including IBM Telelogic Harmony-SE, INCOSE Object-Oriented Systems Engineering Method (OOSEM), IBM Rational Unified Process for Systems Engineering (RUP SE) for Model-Driven Systems Development (MSDS), Vitech MBSE Methodology, and JPL State Analysis [8]. OOSEM uses a traditional top-down systems engineering approach with the Systems Modeling Language (SysML). The core activities for development of a system include analysis of stakeholder needs, definition of system requirements, definition of logical architecture, synthesis of candidate allocated architectures, optimization and evaluation of alternatives, and validation and verification of the system [8]. OOSEM utilizes systems engineering as a base, and builds upon it with some common object-oriented techniques. Finally, it introduces unique techniques, (Fig. 2.1), such as causal analysis and requirements variation analysis.

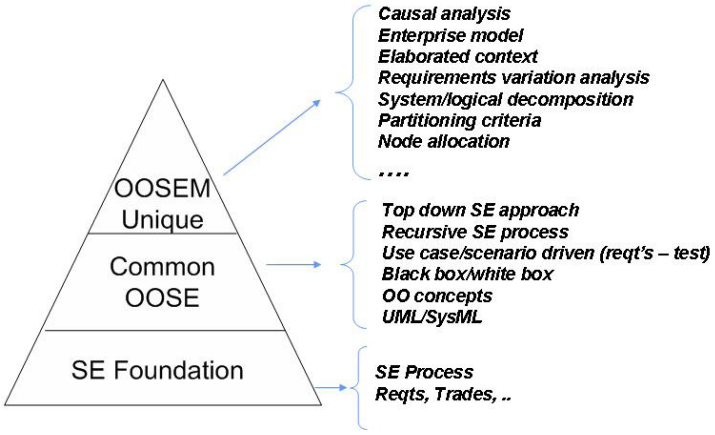

Figure 2.1. Foundation of OOSEM.

2.1. SYSTEMS MODELING LANGUAGE

MBSE utilizes SysML, developed by the Object Management Group (OMG) as request to expand the Unified Modeling Language (UML) for Systems Engineers [9]. UML is prominent in software development for modeling software systems. SysML adds to the functionality so that engineers can model physical systems as well. Version 1.0 of the specification was made available in September 2007. The language helps with architecting systems and specifying components of a system through a graphical representation with a semantic base for structural composition, behavior, constraints, allocations between the three previous representations, and requirements [9].

As part of the additional functionality, new diagrams were created and others were modified from UML specifications (Fig. 2.2). The block definition diagram, internal block diagram, and parametric diagram are the main focus in this research. The block definition diagram represents the "system hierarchy and system/component classification," and the internal block diagram "describes the internal structure of a system in terms of its parts, ports, and connectors." [9] The parametric diagram is used to describe the mathematical relationships within the system.

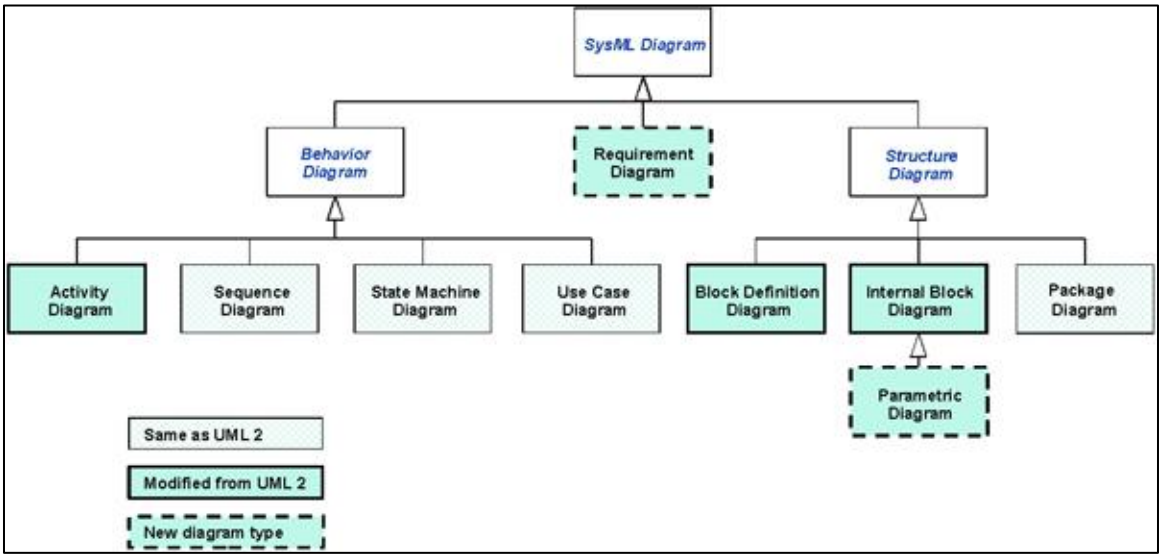

Figure 2.2. SysML Diagram Types [9].

2.2. MBSE INTEROPERABILITY

There are numerous ways of exchanging data between tools including manual entry, file based exchange, interaction based exchange, and repository based exchange [1]. The manual method involves typing in the data in each tool separately. A dual screen setup would be beneficial so each tool could have a screen to display its information. This would end up being a very time consuming approach to data exchange. The file based exchange uses applications that can understand similar file types. This would be like document applications being able to open different formats like .txt, .rtf, or .doc. The interaction based exchange needs a tool's application programming interface (API). The API allows other tools to access and filter its data. This method has the most overhead and difficulty in terms of setup. The last method, repository based exchange, uses a database accessible by multiple tools.

In SysML, all components a model can be represented as metadata. XMI is a file based exchange method based on the industry standards XML, Meta Object Facility (MOF), and UML. It is a set of rules for transforming model information into a unique set of tags in XML [1]. Patel, et al. [10] goes further into using the XMI format to allow for executing SysML models. The information can also be transformed for use by Modelica, as shown in [11]. A second model interchange standard is ISO 10303 and its specific application protocol 233 (AP233). ISO 10303 is also known as the Standard for the Exchange of Product Model Data, or STEP. It is an international standard used to describe "describe product data throughout the life cycle of a product, independent of any particular system" [1]. AP233 was created to support systems engineering, and was developed in coordination with SysML.

XML is a flexible text format developed for the exchange of information. It is machine-readable while also being able to be easily understood by a person [12]. It is also not tied to any specific software application. XML is organized in a hierarchical structure made up elements. The elements can be specified by the user under any name, or tag. This allows the user to create and organize data in a specific manner. However, this also means any application using the data will have to know the structure and tags of the data. These mappings of the data can be supplied by an associated schema.

2.3. PREVIOUS WORK

Model-based approaches have been implemented on other projects as well, ranging from large telescopes [13] to disaster management systems [14]. There are also numerous challenge teams for using MBSE to solve particular problems in the areas of Modeling and Simulation Interoperability, Space System Modeling, Telescope Modeling, and GEOSS Modeling [15]. In [16], Haiar and co-author found that design and analysis could be performed simultaneously by modeling objects in an abstract manner and later develop the physical model as it was finalized. This allowed greater flexibility in design changes. They found that model-based engineering provides a way to reduce design cycle time. This type of approach is beneficial for base camp planning since the environment that a base camp operates is always changing. Populations, missions, threat levels will never be constant. So any big changes could be implemented on the model to anticipate changes required on the camp.

3. BASE CAMPS

3.1. BASE CAMP PLANNING

Plans for developing Army base camps must take into account up to a 20-year lifespan, sustainability due to limited local resources, community outreach, and changing mission requirements. [17] The camp may have to transition from a force projection mission into a humanitarian relief mission if a natural disaster were to occur nearby. Plans should also try to have minimal impact on the surrounding community and environment, or improve the existing conditions. Political, social, and weather environments also need to be considered for planning.

Depending on the size, the base camps can be classified as a Patrol Base, Combat Outpost (COP), Forward Operating Base (FOB), or Super FOB. Base camps must also take into account any supporting camps that will need to be continuously supplied with exhaustible resources and equipment. Each FOB may have multiple COPs that it has to keep supplied, and additionally each COP may have to supply multiple patrol bases that logistics must be pushed out to.

The base camp can have many possible facility types. Each type performs a different function for the camp. Each type may also have multiple instances of it. For example, there would likely be multiple housing facilities to accommodate the population of the camp. These facilities can be grouped under four main types: (1) Living facilities provide the basic necessities like sleeping, cleaning, and eating; (2) Support facilities are optional and provide for the soldiers' morale, welfare, and recreation. They could include a chapel, activity center, fitness center, etc.; (3) Operational facilities such as motor pool, aviation, and tactical operations center support military operations, and (4) Utility facilities supply or process electricity, water, and waste. Each of these needs to be modeled with an emphasis on how they interact with the other base camp facilities.

3.2. PREVIOUS WORK

The Theater Construction Management System (TCMS) is a tool used for computer-aided "planning, design, and management of contingency construction mission in a theater of operations and for emergency construction support during disaster relief

operations [18].” The tool contains a repository of facility designs, component designs, and some base camp designs. One of the drawbacks found with the system is the lack of life cycle analysis of the base camps.

The Geographical Base Engineer Support Tool (GeoBEST) is a separately developed decision support tool for base camp planning developed by the US Army Engineer Research and Development Center (ERDC) and the Air Force. GeoBEST determines the required assets to deploy a base camp based on a given population. The tool can also “spatially visualize a layout” and help with spacing requirements between facilities [19]. Both of the tools, however, lack an ability to analyze utility requirements like electricity and water.

3.3. DEFINING THE MATHEMATICAL MODEL

During the process of defining the system, 12 parameters are identified that can be used to define the personnel and resource requirement and waste generation of each facility. These parameters are:

- Electricity required
- Fuel required
- Potable water required
- Bottled water required
- Storage area
- Number of personnel to operate facility
- Gray water produced
- Black water produced
- Solid waste produced
- Food stuffs required
- Footprint of facility
- Maintenance hours per day.

Each parameter is estimated with a total consumption/production per day per solider. Then, each facility’s parameter is given an estimate of the percentage it uses of the total amount. Many of the values are derived from field manuals like the Sand Book [20], Red Book [21], and other reports [3]. Other values are given using engineering

approximations until totals resemble anticipated totals. All values, estimations and totals, are verified for general accuracy by subject matter experts familiar with operational camps.

It should be noted that the estimations for these parameters are not linearly scalable. Values for the larger size camps will not always work for smaller camps. Each value has an associated soldier population range it is accurate for. Also, some of the smaller camp's facilities have constants instead of percentages. For example, a dining facility requires 2 personnel, regardless if there are 100 soldiers or 150 soldiers. Parameter values will also differ by geographic location. A camp in the arctic or desert will need more fuel to produce more electricity for heating or cooling. Meanwhile, a camp in a moderate temperate zone will not require much power for heating and cooling.

3.4. DEFINING THE MODEL

3.4.1. Base Camp Domain. The base camp domain is modeled to set up what will be affecting the system, internally and externally, and their relationships with each other and the given system (Fig. 3.1). Internally, actors are made for soldiers, civilian workers, and vehicles. Actors are defined as representations of person, organizations, or external systems that participate in the system [1]. Actors are chosen as the method for modeling them because they are able to act as consumers of utilities within the base camp, and also are able to leave the boundaries of the system. A subsystem block is created to represent the facilities that make up the components of the base camp. The last block is for the environment, which includes influences outside the boundaries of the system. The environment is made up of parts which include social, weather, political, and other bases. It also includes an actor representation for enemy combatants.

3.4.2. Base Camp Components. To keep the initial model simple, facilities are modeled as abstract objects where utility requirements can be changed as needed. This makes it possible to create the base camp model without having to model every tent, or structure, or variation of a facility. For example, the dining facility is created as just dining facility, and not 'Tent Type 1' dining facility or 'Tent Type 2' dining facility. Its utility requirements can be altered as required.

The first step is to create a package organization and hierarchy (Fig. 3.2). Packages for each facility are made and placed into their appropriate category. Packages for system actors and facility variables are also created. This is similar to the domain diagram, but with greater detail for the facility components and utilities.

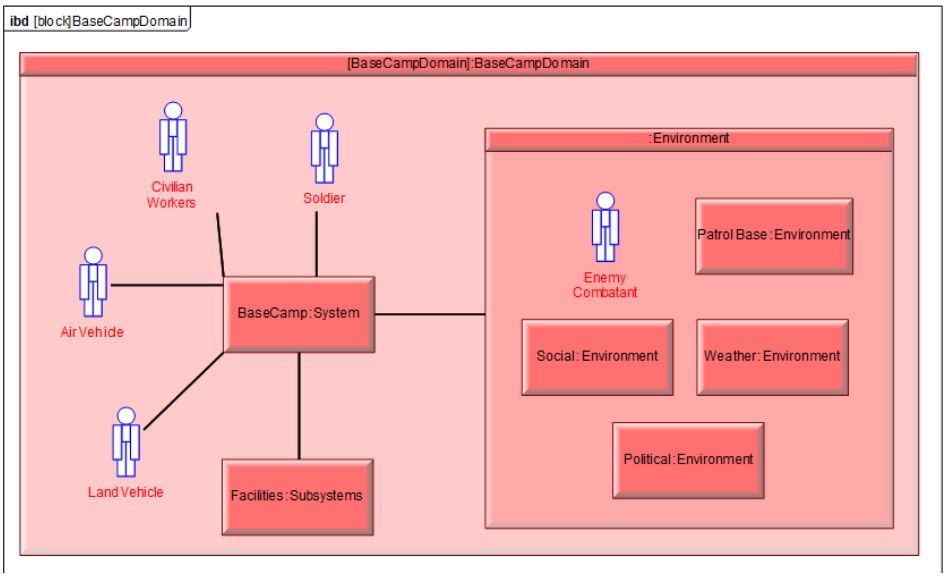

Figure 3.1. Base Camp Domain.

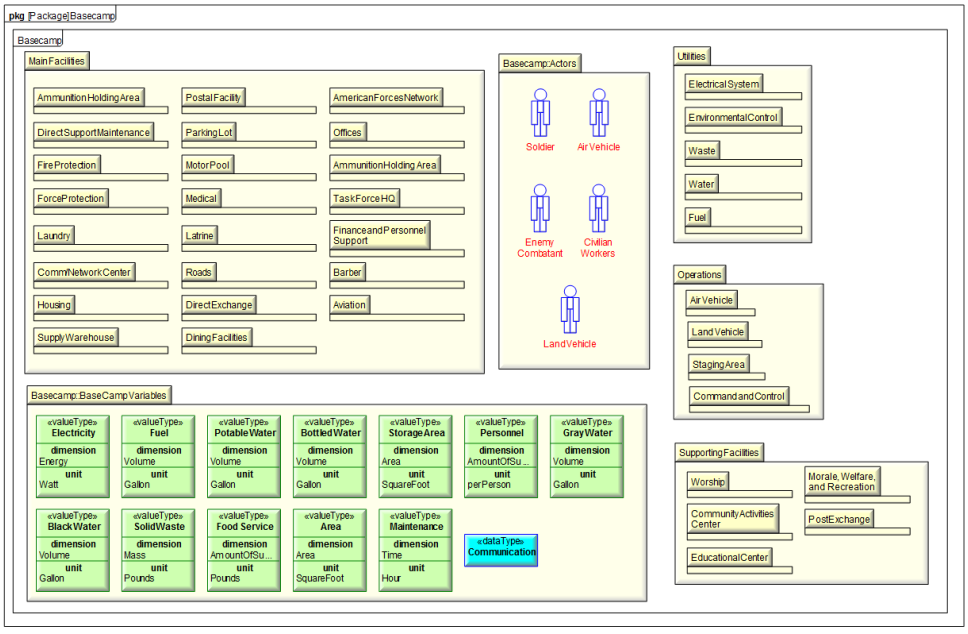

Figure 3.2. Package Diagram showing facilities and utilities.

These packages include all possible components and flows that could make up a base camp. The vehicles are modeled as actors and block because they have the ability to enter and leave the boundaries of the base camp. When the vehicles are within the boundaries, they will act as like a facility consuming resources and producing waste. Creating them as blocks as well is a way to show that dynamic role.

The next step to begin creating blocks for each of the facilities, and their 'flows' of the resources and wastes. Blocks are a way to represent components of the system. Blocks can be composed of other blocks. In order to model the characteristics of a block, a value property for each parameter is modeled as part of the block. Each value property is associated to a value type. A value type describes the quantities [1], in terms of dimensions and units. SysML is flexible enough that it allows the modeler to define new units, like gal/day or gal/soldier, if needed. Properties for all of the parameters are added to the individual blocks regardless of whether the facility consumes or produces the flow. If the facility does not consume or generate a flow, then the value property's default value is set to zero. The default value for each value property is set to the percentage defined in the mathematical model. When the value is a constant, like a facility needing two personnel no matter what the environment variables are, the 'Is Constant' option is set to true and the constant value set as the default value. Blocks are created for each facility and the parameter flows added to each block (Fig. 3.3).

Depending on the amount of details desired, parts within a facility can be modeled as well. In the dining facility, the 'Kitchen' and 'Eating Area' are added. The 'Kitchen' is where the food will be prepared and served, and the 'Eating Area' is where the soldiers sit to eat. The flows are connected to the certain area that uses them. In this example, the eating area only generates solid waste. The kitchen requires electricity and potable water, and produces solid waste and gray water. The issue with adding parts is that it begins to affect the amount of abstraction in the model. For a different dining facility, the 'Eating Area' may require electricity as well for lighting. A completely different block would need to be created to represent the facility. A method for distinguishing these potentially different blocks is covered later.

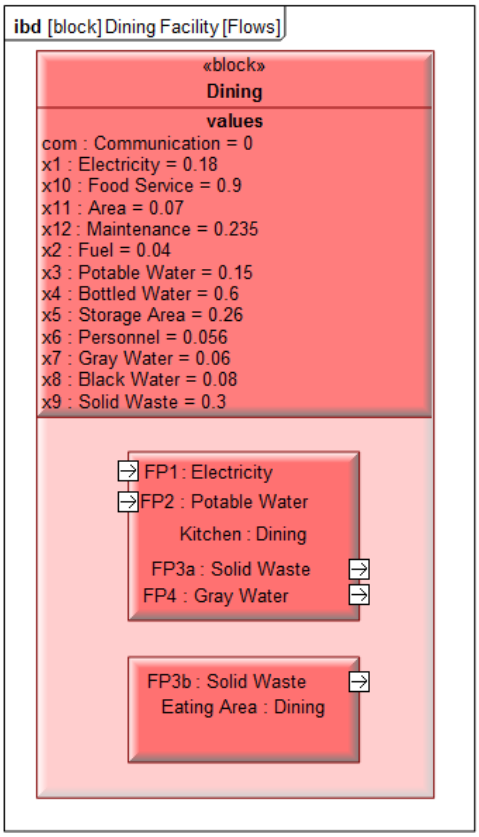

Figure 3.3. Block Definition Diagram of dining facility.

3.4.3. Identification of Interactions. There are two types of interactions modeled for this project, the physical and mathematical. They physical interactions show where the utilities flow from and where they flow to. Two different views can be generated: how all utilities flow in and out of a specific facility (Fig. 3.4) and which facilities are connected to a utility facility (Fig. 3.5). In the first view, the dining facility is modeled in a 150 soldier camp.

It shows how potable water will come from a storage tank, through a distribution system, a pump in this case, and ends at the dining facility. The distribution system and the dining facility also get electricity from a generator. Finally, the solid waste produced from eating gets taken to a burn pit for incineration. This can easily be generated for any facility. The relevant blocks are added to the diagram. For each flow into or out of a facility, a flowport is added. There are three types of flowports: in, out, inout. Connectors

display which ports are connected to each. Then, an ‘ItemFlow’ is added to the connectors to model which type of flow and the direction.

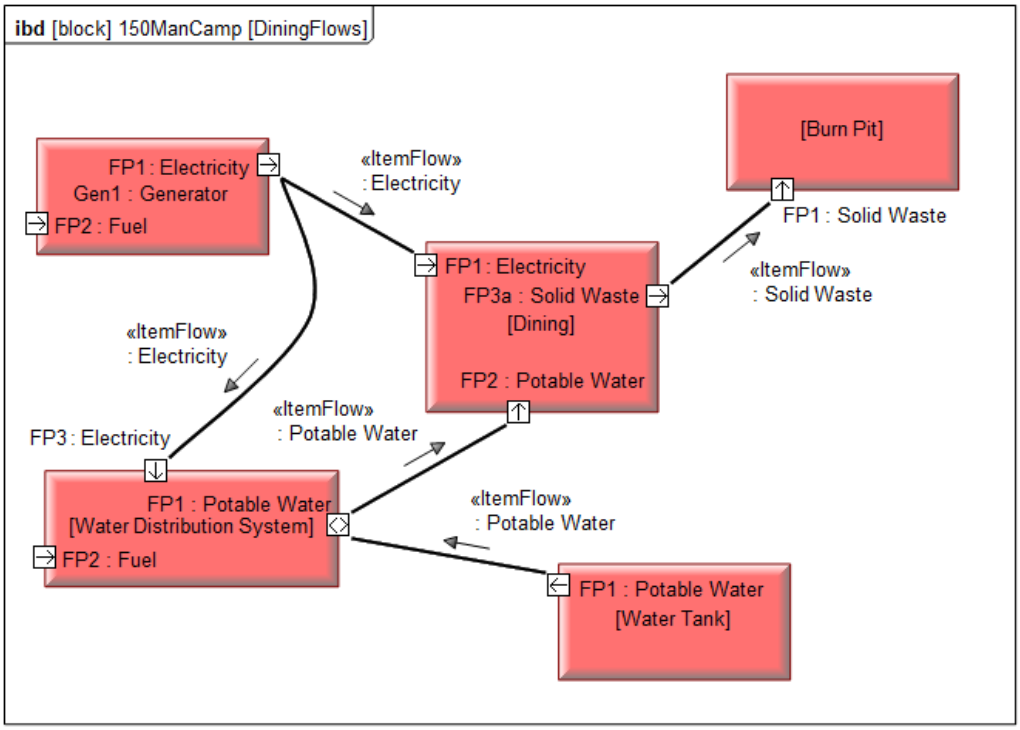

Figure 3.4. Dining facility flows for 150 soldier camp.

The second view is beneficial to show which utility facilities are responsible for which facilities. In the example, Generator 1 is responsible for many of the living facilities. Generator 2 and Generator 3 are sole sources for the more operationally import Tactical Operations Center (C4ISR) and Force Protection, respectively. A priority could be given to the generators for which need to be monitored more than the others. Analysis could also be performed on how mission effectiveness is affected due to Generator 2 going offline.

The second types of interactions, the mathematical interactions, begin to show the requirements and production of the utilities. They represent the mathematical model developed earlier, and are made through parametric diagrams (Fig. 3.6). Each base camp parameter has an estimated usage per person per day. These values and the soldier population are modeled as value properties. Two constraint blocks are created for

calculating the total base population and each facility's utility totals. Constraint blocks allow for the reuse of constraints. Constraints can be any mathematical expression [1]. The variables of these mathematical expressions are the constraint parameters, shown as input and output in figure 8.

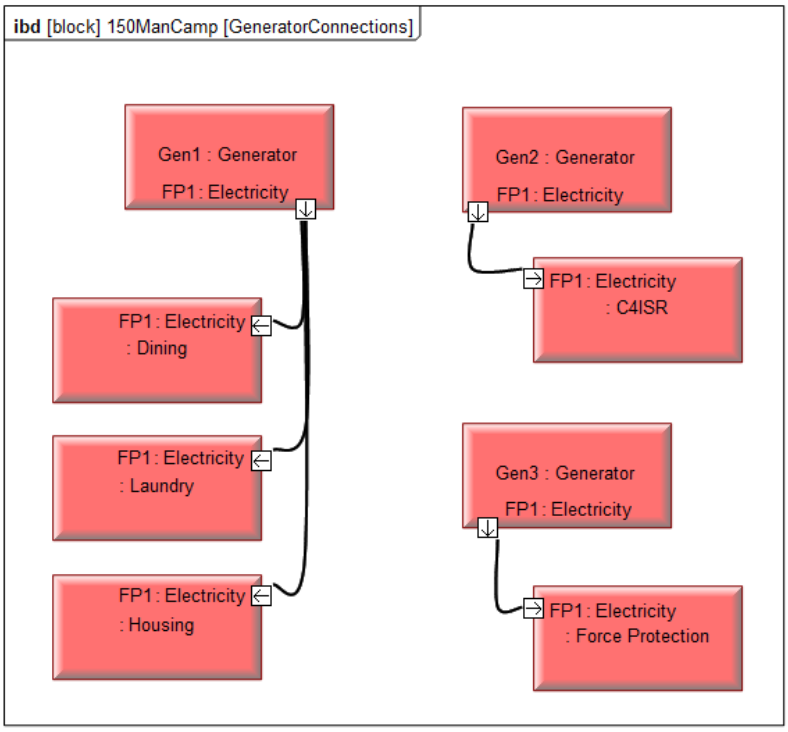

Figure 3.5. Generator connections to facilities for 150 soldier camp.

In order to get the total utilities required by each facility, the percentages provided by the individual facilities for the individual utilities are multiplied by the corresponding usage estimates and total population. Parametric diagrams are created for each facility. Also, the summation of the totals from the facilities will have their own parametric diagrams to calculate the total utility requirements of the entire base camp. The problem can end up being a very large system of linear equations as number of facilities used increases. This system would need to be modeled on the same diagram to understand the math, but the diagram would be unreadable and of no use to a human user.

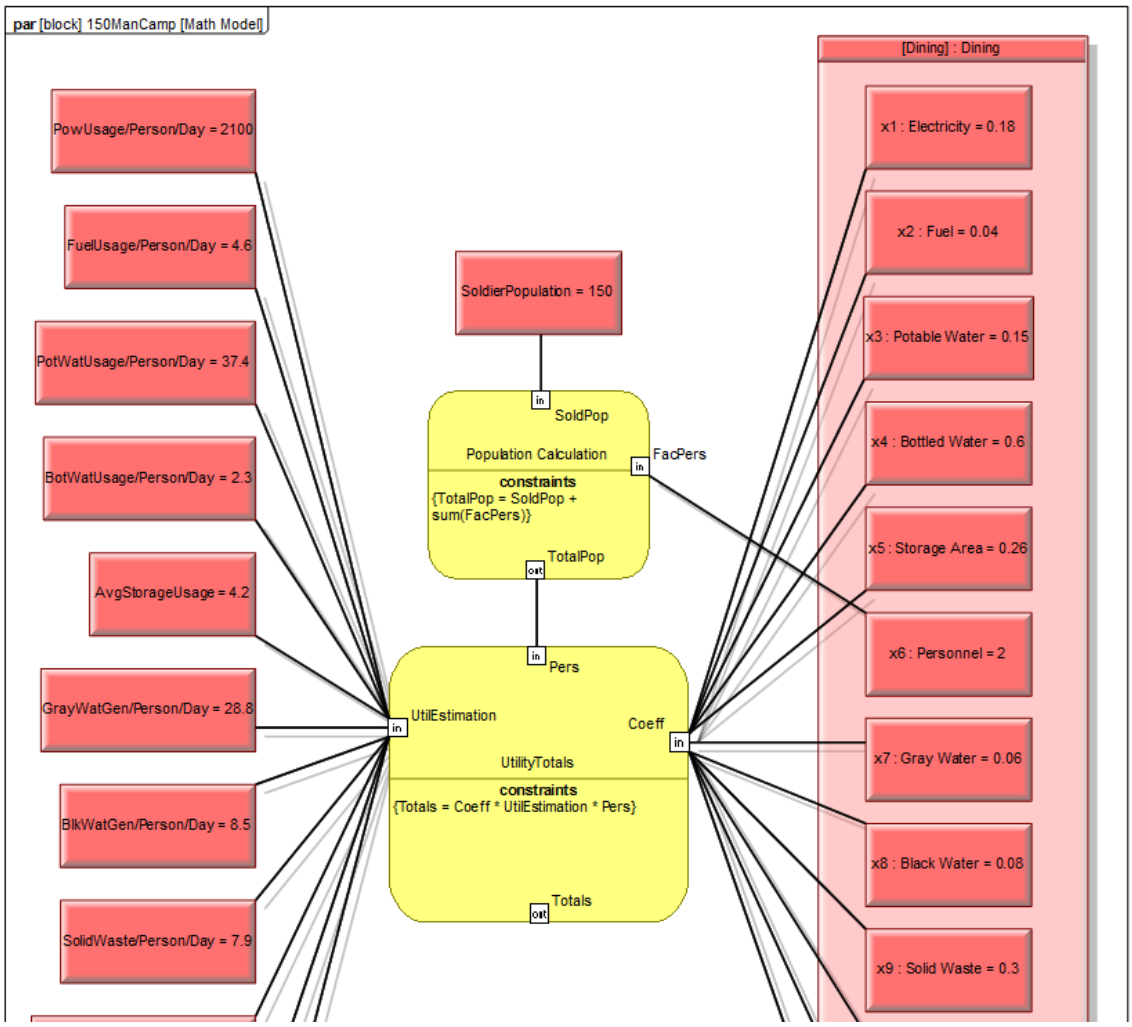

Figure 3.6. Parametric diagram of dining facility parameters.

3.5. LIBRARY OF FACILITY CHOICES

As determined earlier, there is a range of soldier populations for each facility for which the given requirements are accurate. Each of these different facilities is created as separate blocks in the model, adding to a library of choices for creating a base camp. Due to the number of possible facility types, it helps to have a way to differentiate the facility levels so that during base camp planning each facility's parameters does not need to be changed individually and facilities of the same level could easily be pulled from the library. There are two possible methods for differentiating the structures: separation through packages and creating a domain-specific profile for the model.

3.5.1. Separation Through Packages. The simplest and quickest method for separating the facility levels is by placing blocks in their respectively tiered packages, as in figure 3.7. In this figure, the blocks are given notional values and do not represent factual data. First, a package is created for the first tier. Then, blocks for the facilities are created, including their value properties. When this is finished, the package can be cloned and renamed into another tier level. The final step is altering all the value properties to reflect the correct information per tier level.

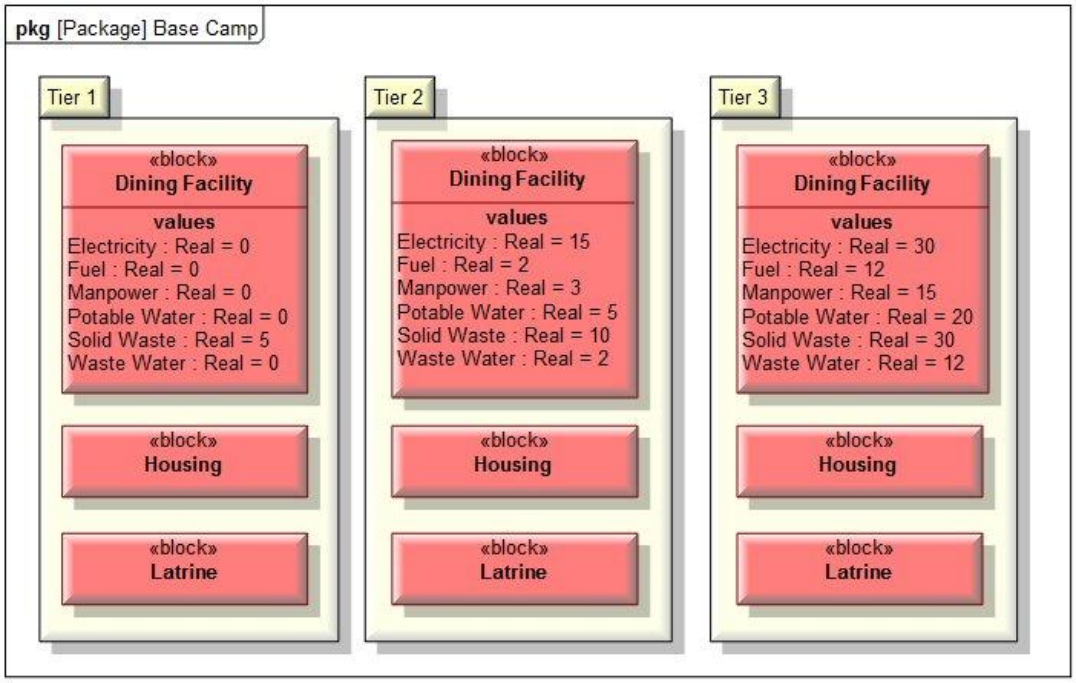

Figure 3.7. Facility levels separated through packages.

The value properties are used to measure any quantifiable unit. In this example, manpower is the number of personnel required to run the facility, in addition to the soldiers they are providing for. So a tier 1 dining facility requires no personnel while tier 2 and tier 3 facilities require 3 and 15 personnel, respectively. The advantage of having the tiers separated into packages is the packages can be exported and imported into new basecamps that are being planned. The planner determines which tier would be used based on the number of soldiers and imports the package from a database. The base camp is essentially all there, and only minor alterations may need to be performed.

The problem with this approach is that not every facility will fit nicely into a package. The housing facility may only require two tiers, a tent and a prefabricated structure. The real calculation of the required utilities would come from the multiplicities of the tents or prefab structures. There would also be redundant housing blocks in tiers greater than level two, since any of those facility blocks would contain the same value properties as the tier 2 block. Another problem is where a 2nd tier level of a structure may start being used when other structures have 3rd tier levels already in use, based on the number of soldiers. A 2nd tier housing facility may start around 500 soldiers, where a 3rd tier dining facility would start around 300 soldiers.

3.5.2. Separation with Domain-Specific Profile. A method to circumvent the problems with tiered packages is creating a domain-specific profile to apply to blocks. Creating a new profile is a common method to use when more domain-specific information is desired within the model, as seen in [22] and [23]. SysML is actually a profile for UML to help extend it beyond the software domain [1]. A profile is a kind of package of stereotypes and tag definitions. Stereotypes are specific metaclasses, and tag definitions are properties of a stereotype [24]. A stereotype would be like a block (a SysML stereotype that extends UML's class), and a tag definition would be the value property. In order to use tags, they have to be linked to a stereotype, and the stereotype applied to a model item or items, as shown in figure 3.8. A stereotype called 'Facility' and two tags, 'MinSoldiers' and 'MaxSoldiers', are created in order to specify a minimum and maximum number of soldiers that a certain level of facility can handle. In addition, the new stereotype also allows a planner to specify that the block is a facility, rather than a utility or structure.

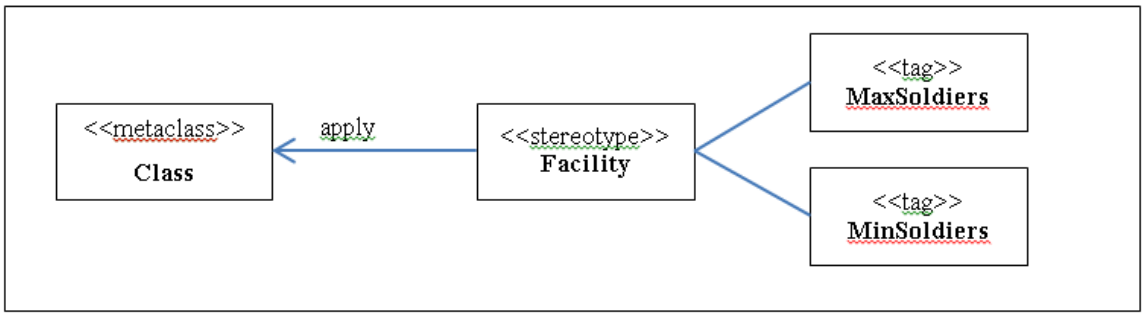

Figure 3.8. Linking the new stereotype and its associated properties.

The next step is to create a new stereotype and link it to the ‘Class’ model item, or blocks in this case. By linking the stereotype to the block, the block is associated with the tag definitions. In the stereotype properties, allow the stereotype to be browsable and shown on all diagrams. Next, create two new tag definitions and link them to the new stereotype. Now the blocks are created in the same package. Again, the dining facility is used for the example in figure 3.9. It should be noted that the blocks must have different names if they are in the same package. The final step is to apply the ‘Facility’ stereotype to the blocks that are to have the ‘Facility’ stereotype applied to them, which is all three Dining Facility blocks in this case. When the new stereotype is applied, a new tab in the properties displays the two tag definitions and a way to input their values. In order to get the min and max soldiers to display on the diagram, the respective compartments are selected and toggled on in the style options. Now the tiered facility options can be displayed so planners are able to see the different facilities available and how many soldiers each supports.

The plan is that a program is given design choices like soldier population, location, longevity, and mission by an end user at the beginning of execution. It then looks through the facilities and picks out the facilities that are required and optional based on those initial choices. It is similar to a search engine where the user inputs keywords, and the engine returns relevant information. The database of facilities and how they are organized would be done by an expert in the field of base camp planning. In XMI, each component is given a unique identification string (Fig. 3.10). The profile will make searching through the design parameters easier. The ID that fits the design choices is then used to search for the more detailed information associated with the SysML block. The results and any changes from the program would be added or updated in the XMI file. Then when the model is imported back into the modeling tool, those changes would be reflected in the blocks and diagrams

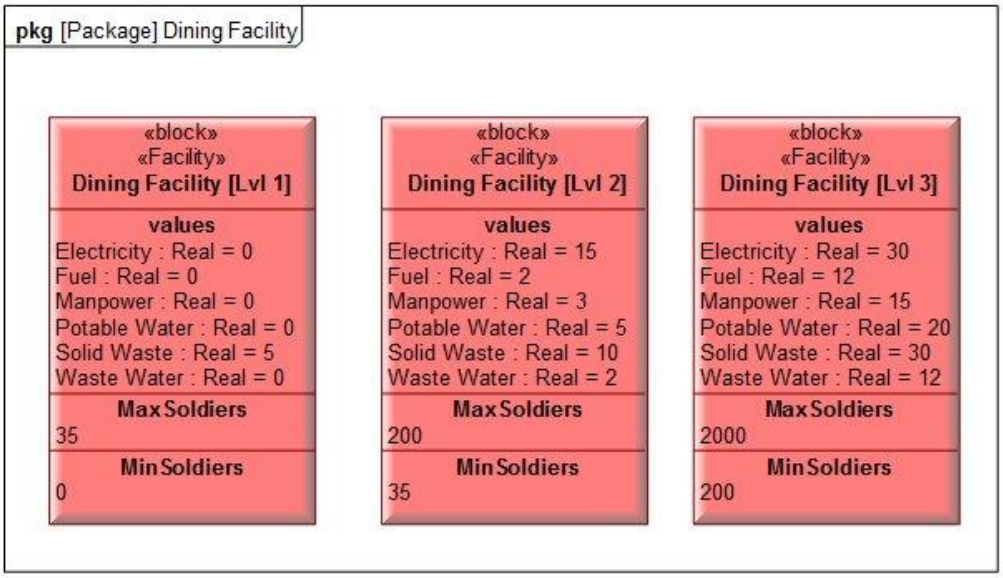

Figure 3.9. Dining facilities with new tag definitions.

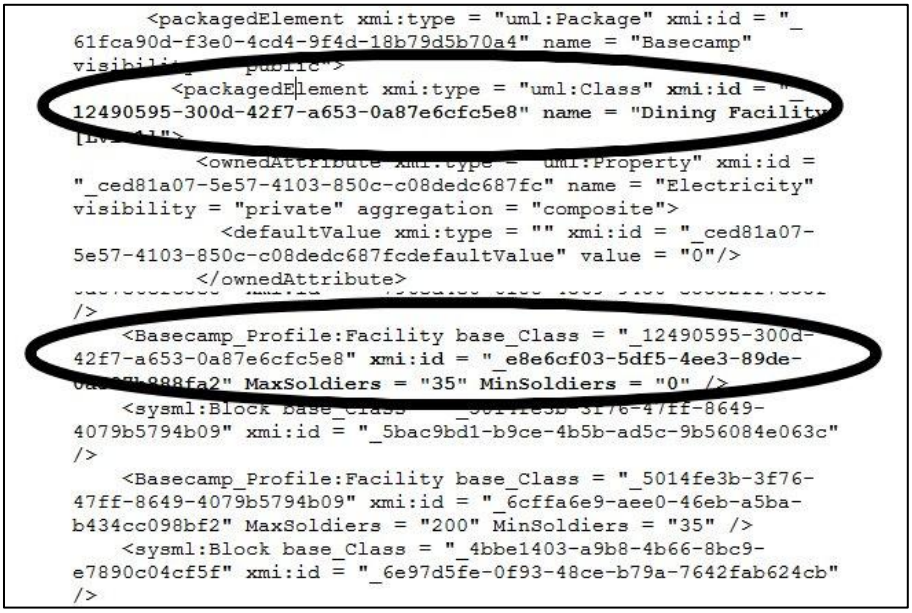

Figure 3.10. Matching IDs for referencing in the XMI file.

3.6. USER INTERFACE

A user interface is developed to set up the variables for the mathematical model, solve the mathematical model, and display the results (Fig. 3.11). The interface is separated into five sections: overview, calculated totals, facility selection, anticipated usage, and facility details. The overview section contains variables about the environment

the camp is operating under. This includes the operational soldiers, mission type, and geographical location of the camp. The calculated totals are the results from solving the mathematical model. The facility selection section is where facilities can be ‘turned on’ or ‘turned off’ for the model. This means if the camp does not have a facility, then it would be unchecked. For each base camp size, a specified list is initially added with all facilities that would typically be on a camp of specified size. For a small camp, around 100 operational soldiers, there are only about 9 facility types to choose. The larger camps will have around the full 40 types of facilities listed. The anticipated totals section is for displaying or modifying the anticipated usage per person per day. The facility details section displays all the parameter values of a selected facility type. The values can be altered and changed from a percentage to a constant. The section also displays the total utility usage/production for the individual facility. Any changes to values that can be modified will trigger the totals to be automatically updated. At this time, changing the number of operational soldiers does not affect the facilities that are listed. This means that it cannot be changed from a small base camp to a large base camp after the small base camp is initially set. The graphical interface is made using WxPython, with Python as the base programming language. Two additional libraries, SciPy and NumPy, are required for solving the system of linear equations.

In order to get the user interface developed and working, a simplified XML file is created with the facility information (Fig. 3.12). Three different versions were created for 150, 300, and 600 sized camps. The 150 size is accurate for between 100 and 200 operational soldiers. The 300 size is accurate for between 200 and 400, and the 600 size is accurate for between 450 and 2000. They contain only the information that is required to run the user interface and solve the mathematical model. The root, or top level element, contains attributes of the anticipated usage values. Its child elements are all the facilities for the given size. Within the Facility elements are the usage percentages or constants, and the total that would be required given the total population. The total is calculated in the solver, so they act as placeholders. Each usage element also has an attribute to specify whether the value is a constant or not.

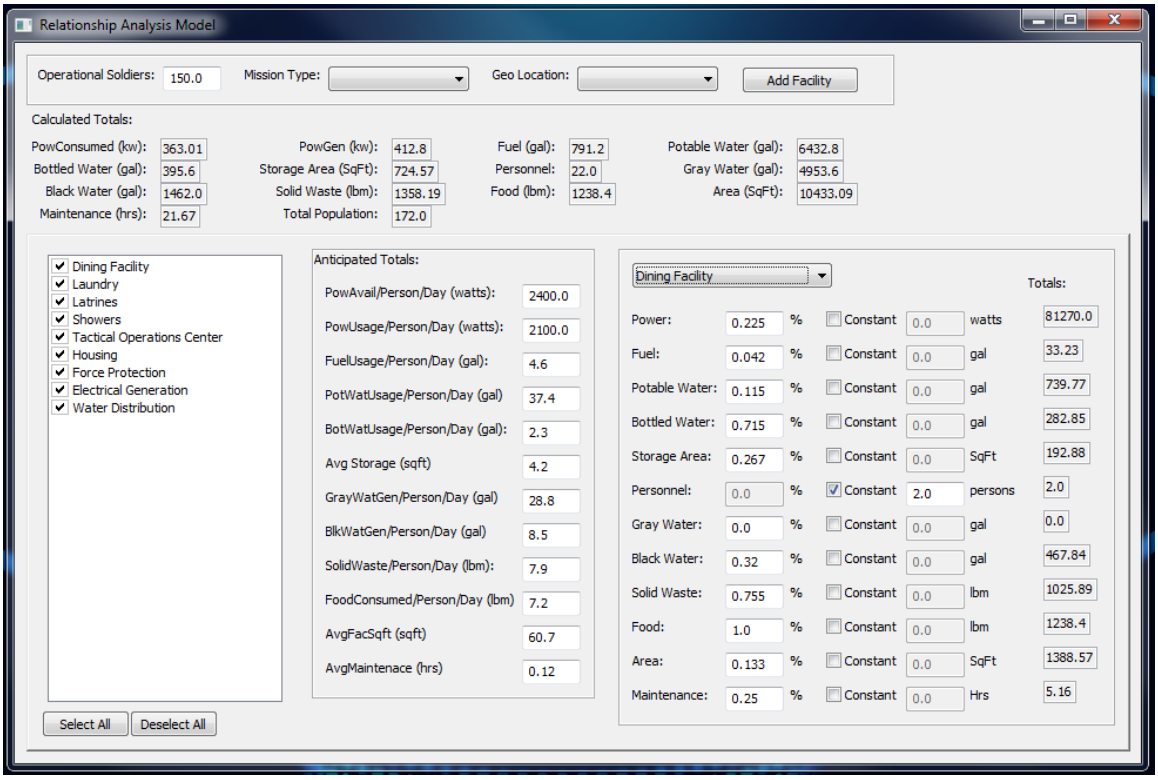

Figure 3.11. User interface that solves the mathematical model and displays the results.

There are numerous methods of retrieving data from an XML file in Python. The main two methods are through Simple API for XML (SAX) or the Document Object Model (DOM). SAX requires more details to manage for the programmer, but requires less memory and lower overhead [12]. SAX processes events, where content-related events would be the prominent type in this case. It only looks at a small portion of the document at a time. It has start and end events, with content represented by events between those two. DOM requires more memory because it loads the entire document into memory, where it is made into an object for the application. Nodes represent parts of the document, and there are separate node types for each of the structure types of a XML document. When a node is 'grabbed' by the application, the DOM provides methods to get the child nodes which contains the content of the element [12]. The original process of getting the data involves using DOM (Fig. 3.13). Each parameter for the facility is made into a node, creating an array. The data from the element are then extracted, added to a list, and made into a programming class.

```
<?xml version="1.0"?>
<Base PowAvail='2400' PowUsage='2100' FuelUsage='4.6' PotW
 BotWatUsage='2.3' AvgStorage='4.2' GrayWatGen='28.8' BlkW
 AvgFacSqFt='60.7' AvgMaint='0.12' >
<Facility>
    <Name>Dining Facility</Name>
    <PowerUse IsConstant='0'>0.225</PowerUse>
    <FuelUse IsConstant='0'>0.042</FuelUse>
    <PotWaterUse IsConstant='0'>0.115</PotWaterUse>
    <BotWaterUse IsConstant='0'>0.715</BotWaterUse>
    <StorageUse IsConstant='0'>0.267</StorageUse>
    <PersonnelUse IsConstant='1'>2</PersonnelUse>
    <GrayWaterUse IsConstant='0'>0.0</GrayWaterUse>
    <BlkWaterUse IsConstant='0'>0.32</BlkWaterUse>
    <SolidWasteUse IsConstant='0'>0.755</SolidWasteUse>
    <FoodUse IsConstant='0'>1.0</FoodUse>
    <FootprintUse IsConstant='0'>0.133</FootprintUse>
    <MaintenanceUse IsConstant='0'>0.25</MaintenanceUse>
    <PowerReq>0.0</PowerReq>
    <FuelReq>0.0</FuelReq>
    <PotWaterReq>0.0</PotWaterReq>
```

Figure 3.12. Sample of the XML formatting that goes into the user interface.

```
class GetData:
    def __init__(self, XMLFile, Fac):

        doc = parse(XMLFile)

        Faclist = doc.getElementsByTagName('Name')
        PowerUseList = doc.getElementsByTagName('PowerUse')
        PowerReqList = doc.getElementsByTagName('PowerReq')
        for node in range(len(Faclist)):
            List = []
            List.append(str(Faclist[node].firstChild.data))
            List.append(float(PowerUseList[node].firstChild.data))
            List.append(float(PowerReqList[node].firstChild.data))
```

Figure 3.13. Parsing the XML file using DOM.

The next version of the application uses the ElementTree method (Fig. 3.14), which is becoming popular due to it being lightweight and fast when compared to DOM [25]. It is similar to DOM in that it stores the data as a hierarchical structure in memory, and described as a combination “between a list and a dictionary” [26]. Each element can

have a tag, attribute, text, tail, and child elements. ‘Tag’ and ‘Text’ were used to get the representative string of the data and the content, respectively. ElementTree also provides support for writing XML files. This is used to add the calculated totals from the solver. Each time the totals are calculated, the information is written to an XML file that would be used to import back into the modeling tool. Figure 3.15 shows the totals added into the XML file.

```
class GetData:
    def __init__(self,XMLFile, Fac):

        tree = ET.parse(XMLFile)

        #Get Facility Values
        element = tree.find('Facility')
        x = tree.findall('Facility')

        for index in x:
            List = []
            const = []
            for node in index:
                check = node.get('IsConstant')
                if check == '1':
                    const.append(len(List)-1)
                List.append(node.text)
                FacilityVar.append(node.tag)
            Fac.append(System(List))
            Fac[x.index(index)].SetConstants(const)
```

Figure 3.14. Parsing the XML file using ElementTree.

```
<Base AvgFacSqFt="78.1" AvgMaint="0.25" AvgStorage="7.1" Bl
  PotWatUsage="38.4" PowAvail="4800.0" PowUsage="2600.0"
  <TotalPower>1045.68</TotalPower>
  <TotalFuel>2278.33</TotalFuel>
  <TotalPotWat>15308.87</TotalPotWat>
  <TotalBotWat>997.67</TotalBotWat>
- <Facility>
    <Name IsConstant="0">Dining Facility</Name>
    <PowerUse IsConstant="0">0.183</PowerUse>
    <PotWaterUse IsConstant="0">0.033</PotWaterUse>
    <PowerReq>0.212</PowerReq>
    <FuelReq>190066.58</FuelReq>
```

Figure 3.15. Created XML file with totals.

4. APPLICATION TO SATELLITE SYSTEM

University design teams typically suffer from problems unique to academia including high personnel turnover, a limited time commitment due to classes, lack of experience and knowledge, and keeping consistency across all subsystems. The issue, like in base camp planning, is with knowledge management and educating inexperienced designers so they can begin contributing. Here we will apply the methods used for base camps to the M-SAT Design Team. The method typically used on projects of this nature to help with this issue is a document tree that includes documentation of all designs, tests, procedures, code, etc. Each document includes a revision section in order to track changes. Occasionally, errors are found in the documents. Design changes may be overlooked, or understanding of design choices may be lost with new members. The model-based approach is a possible method to also help address the consistency and knowledge problems, and better verify requirements with simulation analysis.

The main focus for modeling the base camp was on the blocks representing the facilities. In the case of the satellite project, all aspects of MBSE and SysML are taken into account and modeled. This includes requirements, structure, behaviors, parametric analysis, and traceability. Particularly with the parametric analysis, there is some overlap with the base camp project with using the XMI generated file in an analysis application.

4.1. BACKGROUND INFORMATION

The M-SAT Design Team from Missouri University of Science & Technology is currently participating in the Nanosat 7 program which is a joint program between the Air Force Research Laboratory's Space Vehicles Directorate, the Air Force Office of Scientific Research and the American Institute of Aeronautics and Astronautics. The purpose is to "educate and train the future workforce through a national student satellite design and fabrication competition and to enable small satellite R&D, payload development, integration and flight test, Air Force related technologies." [27] The competition lasts two years and challenges student teams to design and fabricate a prototype satellite. There are multiple design reviews throughout the two years where documentation is submitted and reviewed, and presentations are given. The winning team

will get the chance to launch their satellite as a secondary payload on a launch vehicle into orbit about the Earth.

The primary objective of the M-SAT team is to fly two satellites that will operate in close proximity. One satellite will act a Resident Space Object (RSO), and the other as an Inspector satellite. The Inspector satellite will attempt to calculate the Ballistic Coefficient of the RSO. The secondary objective is to circumnavigate the RSO and create a 3D model from images to ascertain the RSO's capabilities. Much of the design is heritage from previous competitions to reduce the design time required, helping the satellite be completed on time. The team does not exactly follow the systems engineering process, however, each of the design choices has been researched thoroughly through trade studies to ensure that they still meet the requirements and mission. At the time of this writing, many of the designs have been selected, and details about those designs documented.

Previous members of the satellite team have laid some ground work for establishing systems engineering on the project [28], and methods of management for developing subsystems [29]. Stewart talks about creating the role of the Chief Engineer, which is a lead Systems Engineer position, and using standard practices for setting up the mission, requirements, and functions. The Chief Engineer position is tasked with verifying that requirements are met and facilitating communication between subsystems. In work outside the team and university, Cole, et al [30] lays out techniques using MBSE in early formulation of spacecraft concepts, from large projects to small satellites. A long term goal for the team would be to have a similar 'virtual satellite' where simulations can be run before hardware is procured. These simulations would include orbital, thermal, structural, and operational analysis.

4.2. DIAGRAM CREATION

4.2.1. Requirements. Since the NS-7 program started seven months prior to starting the model, some of the information had already been developed and documented. First, the requirements are put into the model. These are already in a worksheet format that includes id#, requirement, source, verification and testing documentation. This is a straight-forward transfer of information from document to model. However, it does allow

for review of the requirements and any modifications that were made. One area that is missing in the document is a rationale for the requirements. In the MBSE tool, adding rationales is incorporated in nearly all aspects of the model. The benefit of adding the rationales is added details about where the requirement is developed from aside from the source requirement. For example, many of the structures requirements come from University Nanosat Program requirements provided in the User's Guide. The rationale can point any reader to the specific section of the User's Guide with the requirement. Another example is the minimum time required to operate in orbit in order to complete the mission. There was a specific value, but no information as to where the value came from. In the rationale, estimates are added to include time required for detumble and status checks, mission 1, and mission 2.

In the model, it is also easy to separate and display information for anyone who wants extra information about a particular area. A diagram is created that shows all system level requirements of the Inspector satellite and which mission requirement they are derived from. If someone wants more information about system requirement 2, which relates to operational period in orbit, they can double click on that requirement to open the S1-2 requirement diagram. (Provided the modeler creates that particular diagram and links it). The diagram (Fig. 4.1) displays the rationale, 'satisfied by,' and 'verified by' information. Depending on the depth of the model, either the 'satisfied by' or 'verified by' could have linked diagrams that contain even more information. The 'verified by' test case could include information obtained from an analysis tool, like a power budget.

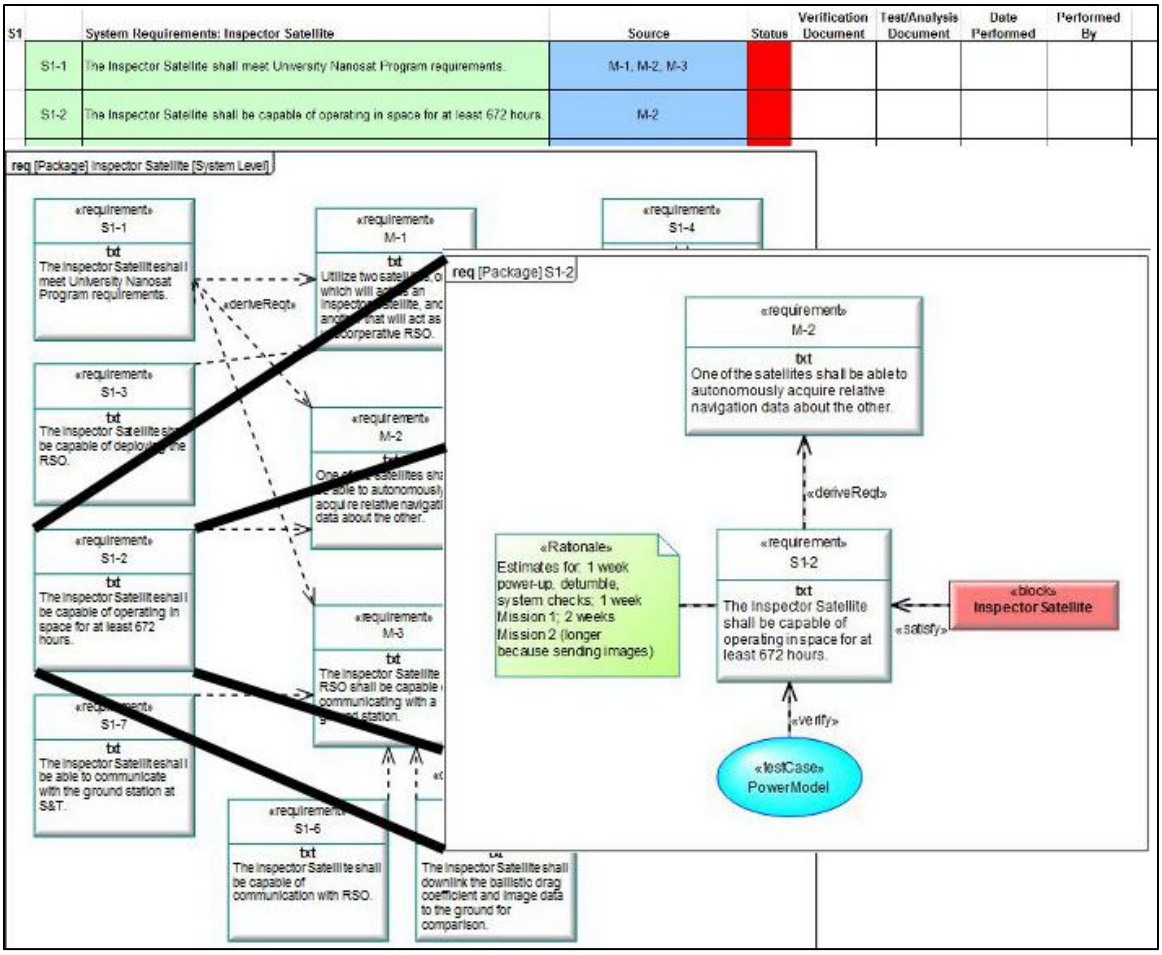

Figure 4.1. Transition from documented requirements to model requirements.

4.2.2. Physical System. Like the base camp, the physical system is modeled going from a level of abstraction to details. First, a block definition diagram is created with the different subsystems on each satellite (Fig. 4.2).

Each subsystem block opens up to another block definition diagram containing the hardware for the particular subsystem (Fig. 4.3a). Some are more detailed than others, as not all designs have been finalized. However, the lack of design information was occasionally due to a lack of documentation. Documents might contain design choices but lack specifications about the hardware. One issue with an MBSE approach noticed early on was determining what should be a block and what should be a part. For example, in the attitude determination and control subsystem, there are three magnetic torque coils used for control. Typically they are thought of as all being the same and would be

modeled using one block with a multiplicity of three. However, each torque coil has different dimensions, leading to different mass and power properties. So, each torque coil must be modeled as its own block. Each block can now be given property values (Fig. 4.3b) about its weight, power, dimensions, etc. Like in the base camp model, this is information that can later be used in an analysis tool.

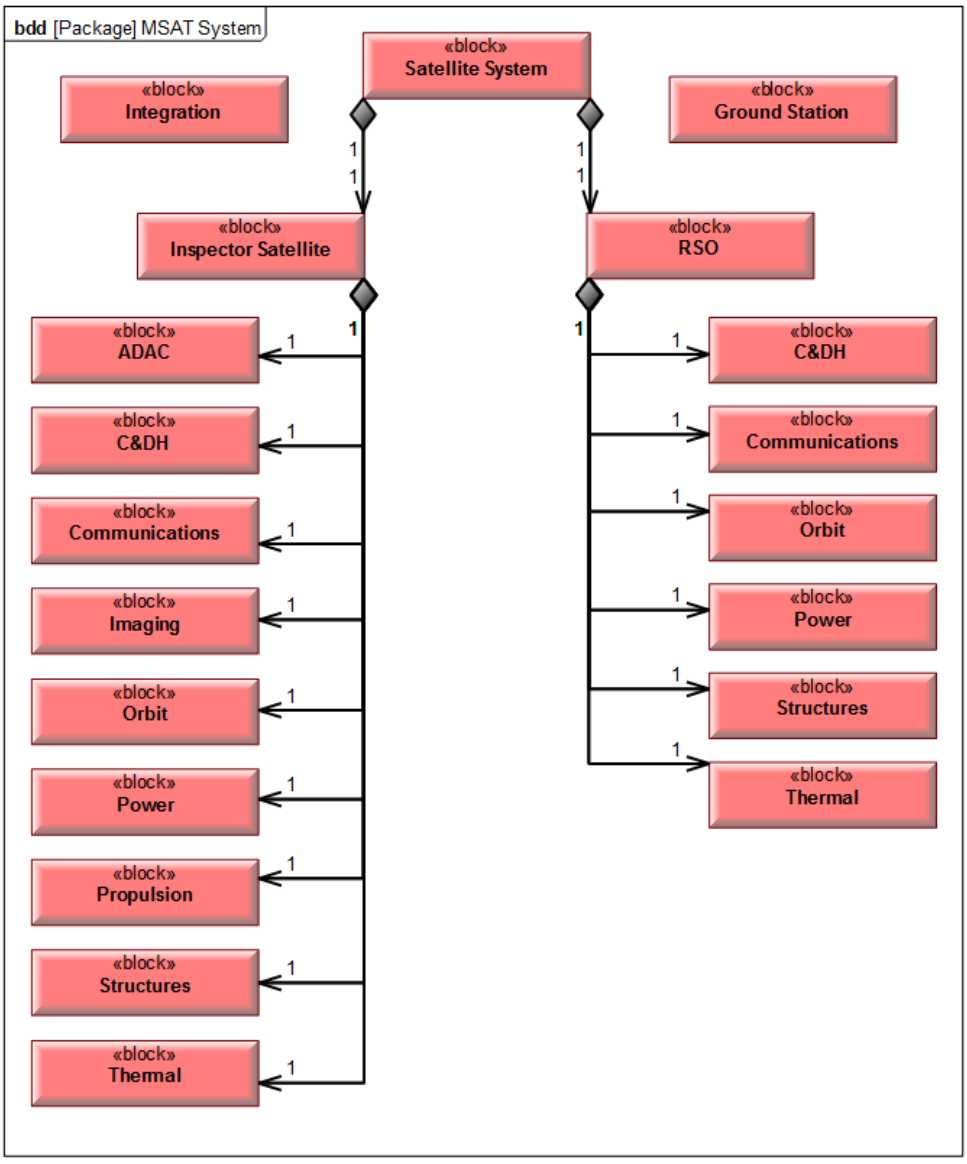

Figure 4.2. Block diagram of satellite subsystems.

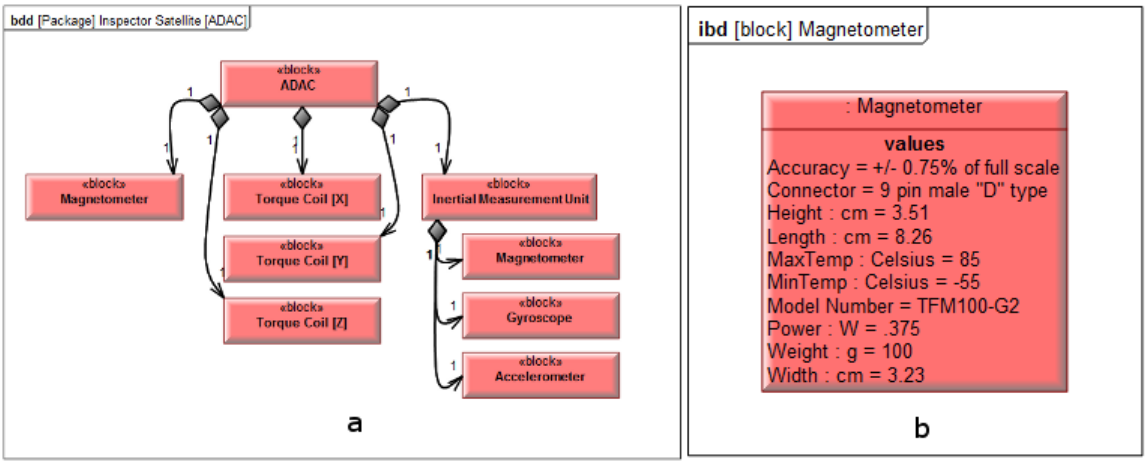

Figure 4.3. Block definition diagrams for (a) ADAC structural composition and (b) Magnetometer specifications.

4.2.3. Mission Modeling. A use case diagram is created and functions added that each satellite will have to perform are added. There is some overlap between the two satellites, such as ‘Provide Power’ and ‘Determine Position.’ The majority of capabilities belong to the Inspector satellite because the RSO will just be a beacon that transmits its position. As the depth of the model increases, activities and sequence diagrams will be linked to these capabilities, and satisfied by physical system components (Fig. 4.4). Associating physical components is beneficial to knowing which components will be running while performing a specified task.

The states, or operational modes, of the system are based on the previous competition’s operational modes. They are then modified, deleted, or added to as required to fulfill the new mission. The first step was to create a state diagram that included all states, and why, or when, the system will move from one state to another (Fig. 4.5a). Each state is then linked to an activity diagram that walks through the general activities performed during that state (Fig. 4.5b). Eventually, many of these activities will link to diagrams that contain more activities required to perform that higher level activity. This is already a defined process within systems engineering.

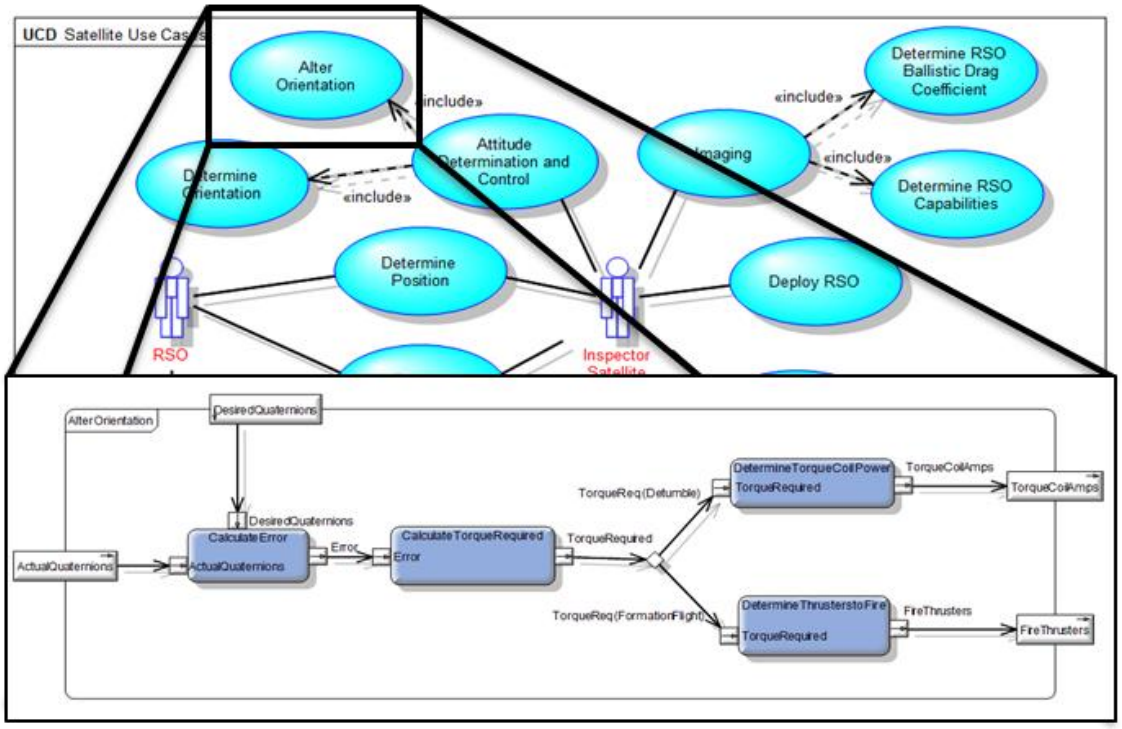

Figure 4.4. Use Case Diagram showing a Use Case linked with an Activity Diagram.

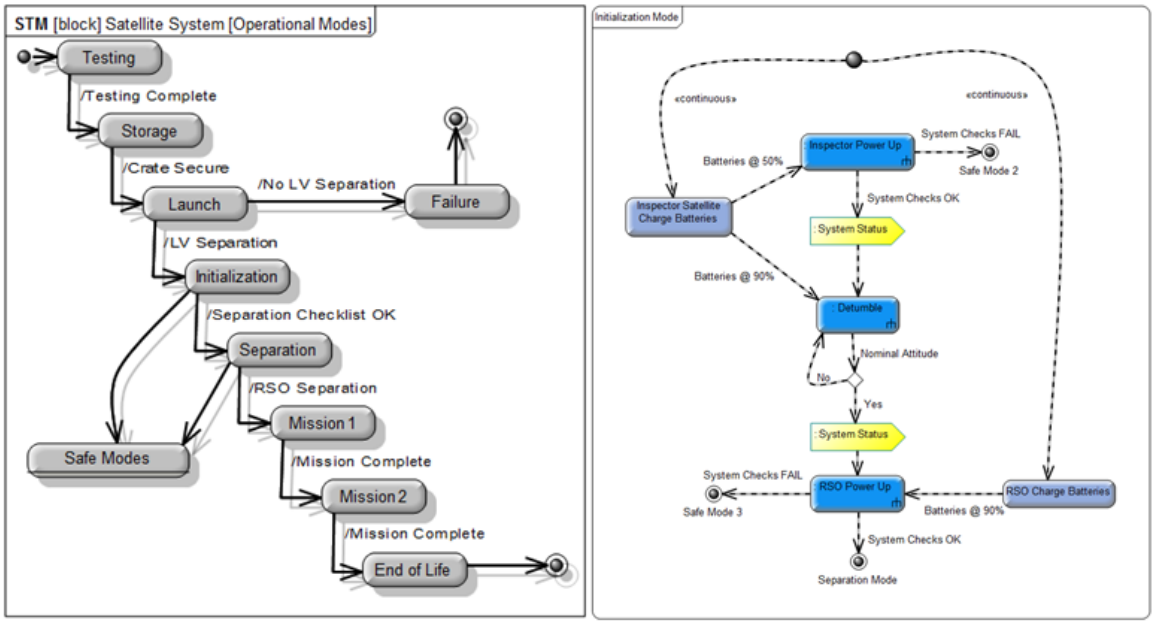

Figure 4.5. Behavioral analysis showing (a) Operational Modes of the satellite and (b) Initialization Mode activities.

The benefits of using a model in this project is providing an easier method to walk through the lifecycle of the system. Each of the high level activities terminates at the start of another state, and by double clicking the termination point the next state's activity diagram opens. A person looking through the model would be able to navigate through the functional flow like they would navigate through a website. Another benefit is that processes are beginning to be determined for each state. Now the state will have a set of activities associated to it, and activities associated to hardware. So, a list of hardware running during each state could be generated for analysis. Depending on the work that is put into the details of the model, this could help produce either a rough calculation for power consumption or an accurate model of power consumption.

4.3. ANALYSIS

Some of the common analysis performed on a satellite early on is the power budget, mass budget, and data budget. These are performed to know how much power is being consumed, the mass of the satellite, and memory that needs to be stored. These all have constraints, or requirements from the customer in the case of mass.

The same process used for the army base camp can be applied to the satellite project. A domain-specific profile for satellites may also need to be created. The model could be exported out in the XML format, and the hardware and state information extracted. The information could then be used in a totals analysis tool based off the base camp one (Fig. 4.6).

The interface would again be separated in sections for mission mode, totals, component selection, and component details. The mission mode could contain the selection of modes. When a state is selected, components that are active during the state would be automatically selected. Totals for that state would then be calculated and displayed. There would also be the option to turn on/off components. This analysis tool would work well for power mostly because mass will not change due to a component being active or inactive. Additional types of analysis tools could be developed as well for each type of analysis that needs to be performed. The results could then be imported back into the model as those test cases that verify requirements.

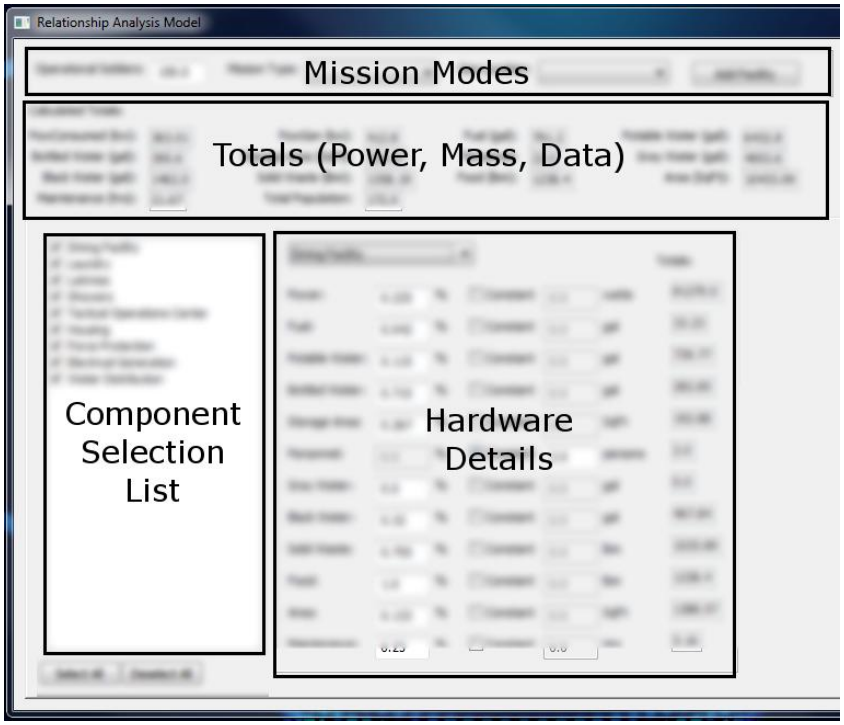

Figure 4.6. Theoretical user interface for satellite totals analysis.

5. CONCLUSIONS

The results of this work highlight methods to model and examine the interrelationships between elements of an operational base camp and the coupling that occurs between these elements. The diagrams show the sources and sinks of utilities. To manage the inherent complexity in these systems, much of the effort must be put forth early to develop appropriate requirements that will produce a sustainable and adaptable system. Creating a domain-specific profile for base camps appears to be beneficial because it allows for more customization. It would be essential for creating a library of design choices. Parameters could be added for mission type and geographical location, in addition to the minimum and maximum soldiers that facility supports. With those properties added, a program could be given the location of the base, the mission to be performed, and the anticipated number of soldiers needed for the mission. The program would then run through all of the facility choices and grab the ones that match the search criteria in a method that resembles searching a database.

One of the issues that arose with creating the model was making changes to the mathematical model and variables. For example, if there wasn't a gray water variable that was initially considered and it needed to be included, then it would have to be added individually to all of the facilities. There is no way to add the variable to one super class and have it added to all other facilities. It ends up becoming a very time consuming process to make changes that affect all facility types. One option around this issue is to the make a facility block and then all of the different facilities would be "parts" to that block. However, in this method, the values for the value properties would be the same for all facilities. So, all facilities would end up acting the same and requiring the same amount of utilities. The other option is using the profile created for differentiating the levels of facility types. Tag definitions representing the variables like electricity and water would be added to the facility stereotype. If a tag definition is added or removed from the stereotype, then it would change that for the facilities all at once. A possible issue with this approach would be a loss of functionality. Since the variables are no longer defined as value properties and value types, then there may be no way of making parametric diagrams with them. The mathematical relationships would no longer be able

to be modeled. However, because the mathematical relationships are set up independently in the analysis tool, this may be an acceptable loss of functionality.

The purpose for creating the satellite model is to help with system understanding among team members, make a central repository of information about the system that can be easily located, and help create presentation material for review presentations. As the model is generated, information not previously known or understood is learned. This may not be the same for anyone exploring the model. It was the act of creating the model that was most beneficial to broader system understanding. One observation was that there was always detailed information, but lacking with respect to some of the high level information. This may be attributed to engineers getting into the details too fast, and forgetting to think about the 'why' in the design process. With a diagram hierarchy like in MBSE, it guides a designer to create that high level information first to start the process of getting the details. However, this is not mandatory. The designer could just as easily skip over the high level information.

During the modeling process, the modes of operations and their respective activities were presented to the team. It allowed for an open discussion among all members of the team, and resulted in activity flow changes. A similar approach could be done for all aspects of the satellite: physical, behavior, requirements, and parametric models. This could help inform all members about other subsystems that affect their specific subsystems, but don't know many details about. Presenting some of the high level information would also remind the team about the rationale of what they are designing. A problem discovered when creating the model was to organize it in a document-tree structure. This is counterproductive in some respects. The idea is to start off at one top level diagram, and be able to get any and all information desired by clicking through the linked diagrams. When sections become separated too much, some duplication is performed and information is again lost in a large folder hierarchy.

The eventual goal for the base camp is to be able to import the model into a virtual engineering environment for detailed design and analysis. Facility models would be selected and placed in a 3-D environment that represents the actual terrain of a selected site. This environment will enable detailed analysis of the different utility systems for a given configuration. Third party software would be able to be integrated

into the virtual engineering tool for power grid analysis, water flow analysis, etc. The planner would end up with a site layout and detailed utility requirements. Scenarios could be developed where system failures occur so the planner could document how a configuration will continue to perform. The library repository style of MBSE means fully developed base camps could be reused. A planner could start with a previously designed camp, make some alterations, and run it through the analysis to check if it meets the requirements for the new mission. Currently operating camps could also be analyzed if any design changes are required due to a proposed change in mission or soldier population. The same approach could also be taken with the satellite model. The virtual engineering tool could utilize tools for structural, orbital, or thermal analysis.

APPENDIX A

CODE – BaseCampUI.py

BaseCampUI.py

```
import wx
from DataParser import GetData, GetEnvData
from Solver import Solver

Facility =
Env =
FacilityNames =
FacVariables = ['Power:','Fuel:','Potable Water:','Bottled Water:',
                'Storage Area:','Personnel:','Gray Water:','Black Water:',
                'Solid Waste:','Food:','Area:','Maintenance:','TotalPopulation:']
TotalVariables = ['PowConsumed (kw):','PowGen (kw):','Fuel (gal):','Potable Water (gal):','Bottled Water (gal):',
                'Storage Area (SqFt):','Personnel:','Gray Water (gal):','Black Water (gal):',
                'Solid Waste (lbm):','Food (lbm):','Area (SqFt):','Maintenance (hrs):', 'Total Population:']
EstVariables = ['PowAvail/Person/Day (watts):','PowUsage/Person/Day (watts):','FuelUsage/Person/Day (gal):','PotWatUsage/Person/Day (gal)',
            'BotWatUsage/Person/Day (gal):','Avg Storage (sqft)','GrayWatGen/Person/Day (gal)','BlkWatGen/Person/Day (gal)',
            'SolidWaste/Person/Day (lbm):','FoodConsumed/Person/Day (lbm)','AvgFacSqft (sqft)','AvgMaintenace (hrs)']
VariableUnits = ['watts','gal','gal','gal','SqFt','persons','gal','gal','lbm','lbm','SqFt','Hrs']

MissionTypes = ['Reconnaisance','Humanitarian Aid','Peace Keeping','Offensive Operations','Show of Force']
GeoTypes = ['Southeast Asia','South Africa','Mediterranean','Western Europe','Arctic','City']
current_selection = 0


#-----------------------------------------------------------------------|
#     Base Environment Values                                           |
#-----------------------------------------------------------------------|
class OverviewPanel(wx.Panel):
   def __init__(self, parent, id):
      wx.Panel.__init__(self, parent, id, style=wx.BORDER_DOUBLE)

      st1 = wx.StaticText(self, -1, 'Operational Soldiers:')
      st2 = wx.StaticText(self, -1, 'Mission Type:')
      st3 = wx.StaticText(self, -1, 'Geo Location:')
      self.SoldPop = wx.TextCtrl(self, -1, value=str(Env.SoldierPopulation), size=(50,-1), style = wx.TE_PROCESS_ENTER)
      self.SoldPop.Bind(wx.EVT_TEXT_ENTER, self.SetSoldPop)
      tc2 = wx.ComboBox(self,-1, size=(120,-1), choices=MissionTypes, style=wx.CB_READONLY)
      tc3 = wx.ComboBox(self, -1, size=(100,-1), choices=GeoTypes, style=wx.CB_READONLY)
      #btn = wx.Button(self, -1, 'Update', size=(100,-1))
      #btn.Bind(wx.EVT_BUTTON, self.onUpdateBtn)
      btn2 = wx.Button(self, -1, 'Add Facility', size=(100,-1))
      btn2.Bind(wx.EVT_BUTTON, self.onAddFacility)

      vbox = wx.BoxSizer(wx.VERTICAL)
      hbox = wx.BoxSizer(wx.HORIZONTAL)
```

```
        hbox.Add(st1, 0, wx.RIGHT, 5)
        hbox.Add(self.SoldPop, 0, wx.RIGHT, 20)
        hbox.Add(st2, 0, wx.RIGHT, 5)
        hbox.Add(tc2, 1, wx.RIGHT, 20)
        hbox.Add(st3, 0, wx.RIGHT, 5)
        hbox.Add(tc3, 1, wx.RIGHT, 20)
        #hbox.Add(btn, 0, wx.RIGHT, 20)
        hbox.Add(btn2,0, wx.RIGHT, 20)
        vbox.Add(hbox, 0, wx.ALL, 8)
        self.SetSizer(vbox)

    def SetSoldPop(self,event):
        item = event.GetEventObject()
        Env.SoldierPopulation = float(item.GetValue())
        self.GetParent().GetParent().Panel2.UpdateTotals()

    def onUpdateBtn(self,event):
        self.GetParent().GetParent().Panel2.UpdateTotals()

    def onAddFacility(self,event):
        pass

#-----------------------------------------------------------------------|
#     List of Available Facilities                                      |
#-----------------------------------------------------------------------|
class FacilityListPanel(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self, parent, id)

        vbox = wx.BoxSizer(wx.VERTICAL)

        self.FacCLB = wx.CheckListBox(self, -1, size=(200,380), choices = FacilityNames, style = wx.LB_NEEDED_SB)
        self.FacCLB.Bind(wx.EVT_CHECKLISTBOX, self.onFacilitySelection)

        hbox = wx.BoxSizer(wx.HORIZONTAL)
        btn1 = wx.Button(self, -1, 'Select All')
        btn1.Bind(wx.EVT_BUTTON, self.onSelectAll)
        btn2 = wx.Button(self, -1, 'Deselect All')
        btn2.Bind(wx.EVT_BUTTON, self.onDeselectAll)
        hbox.Add(btn1, 0, wx.TOP, 0)
        hbox.Add(btn2, 0, wx.TOP, 0)

        vbox.Add(self.FacCLB, 0, wx.ALL, 5)
        vbox.Add(hbox, 0, wx.TOP, 2)
        self.SetSizer(vbox)

    def onFacilitySelection(self,event):
```

```
        index = event.GetSelection()
        if self.FacCLB.IsChecked(index):
            Facility[index].Switch = 1
            self.UpdateTotals()
        else:
            Facility[index].Switch = 0
            self.UpdateTotals()

    def onSelectAll(self,event):
        for itemnum in range(len(Facility)):
            if self.FacCLB.IsChecked(itemnum):
                pass
            else:
                self.FacCLB.Check(itemnum, True)
                Facility[itemnum].Switch = 1
        self.UpdateTotals()

    def onDeselectAll(self,event):
        for itemnum in range(len(FacilityNames)):
            if self.FacCLB.IsChecked(itemnum):
                self.FacCLB.Check(itemnum, False)
                Facility[itemnum].Switch = 0
        self.UpdateTotals()

    def UpdateTotals(self):
        Solver(Facility, Env)
        self.GetParent().GetParent().GetParent().Panel5.PowTotal.SetLabel(str(Env.TotalPower))
        self.GetParent().GetParent().GetParent().Panel5.PowGen.SetLabel(str(Env.PowerGen))
        self.GetParent().GetParent().GetParent().Panel5.FuelTotal.SetLabel(str(Env.TotalFuel))
        self.GetParent().GetParent().GetParent().Panel5.PotWatTotal.SetLabel(str(Env.TotalPotWat))
        self.GetParent().GetParent().GetParent().Panel5.BotWatTotal.SetLabel(str(Env.TotalBotWat))
        self.GetParent().GetParent().GetParent().Panel5.StorageTotal.SetLabel(str(Env.TotalStorage))
        self.GetParent().GetParent().GetParent().Panel5.PersonnelTotal.SetLabel(str(Env.TotalPersonnel))
        self.GetParent().GetParent().GetParent().Panel5.GrayWatTotal.SetLabel(str(Env.TotalGrayWat))
        self.GetParent().GetParent().GetParent().Panel5.BlkWatTotal.SetLabel(str(Env.TotalBlkWat))
        self.GetParent().GetParent().GetParent().Panel5.SWasteTotal.SetLabel(str(Env.TotalSWaste))
        self.GetParent().GetParent().GetParent().Panel5.FoodTotal.SetLabel(str(Env.TotalFood))
        self.GetParent().GetParent().GetParent().Panel5.AreaTotal.SetLabel(str(Env.TotalArea))
        self.GetParent().GetParent().GetParent().Panel5.MtnTotal.SetLabel(str(Env.TotalMaintenance))
        self.GetParent().GetParent().GetParent().Panel5.PopTotal.SetLabel(str(Env.TotalPopulation))

#-----------------------------------------------------------------------|
#     Base Camp Totals Inputs                                           |
#-----------------------------------------------------------------------|
class EstTotalsPanel(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self,parent,id, style=wx.BORDER_DOUBLE)
```

```
    vbox = wx.BoxSizer(wx.VERTICAL)
    title = wx.StaticText(self, -1, 'Anticipated Totals:')
    self.SoldPowAvail = wx.TextCtrl(self, -1, str(Env.PowAvail), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[0])
    self.SoldPowUsage = wx.TextCtrl(self, -1, str(Env.PowUsage), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[1])
    self.SoldFuel = wx.TextCtrl(self, -1, str(Env.FuelUsage), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[2])
    self.SoldPotWater = wx.TextCtrl(self, -1, str(Env.PotWatUsage), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[3])
    self.SoldBotWater = wx.TextCtrl(self, -1, str(Env.BotWatUsage), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[4])
    self.AStorageArea = wx.TextCtrl(self, -1, str(Env.AvgStorage), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[5])
    self.SoldGrayWater = wx.TextCtrl(self, -1, str(Env.GrayWatGen), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[6])
    self.SoldBlkWater = wx.TextCtrl(self, -1, str(Env.BlkWatGen), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[7])
    self.SoldSWaste = wx.TextCtrl(self, -1, str(Env.SWasteGen), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[8])
    self.SoldFood = wx.TextCtrl(self, -1, str(Env.FoodConsumed), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[9])
    self.SoldFootprint = wx.TextCtrl(self, -1, str(Env.AvgFacSqFt), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[10])
    self.SoldMaint = wx.TextCtrl(self, -1, str(Env.AvgMaintenance), size=(50,-1), style = wx.TE_PROCESS_ENTER, name = EstVariables[11])
    inputs = [self.SoldPowAvail, self.SoldPowUsage, self.SoldFuel, self.SoldPotWater, self.SoldBotWater,
            self.AStorageArea, self.SoldGrayWater, self.SoldBlkWater, self.SoldSWaste,
            self.SoldFood, self.SoldFootprint, self.SoldMaint]
    grid = wx.FlexGridSizer(13, 2, 8, 5)

    for item in inputs:
        item.Bind(wx.EVT_TEXT_ENTER, self.UpdateEstValues)
        grid.Add(wx.StaticText(self, -1, EstVariables[inputs.index(item)]),0)
        grid.Add(item, 0)
    vbox.Add(title, 0, wx.BOTTOM, 5)
    vbox.Add(grid, 0, wx.ALL, 10)

    self.SetSizer(vbox)

def UpdateEstValues(self,event):
    item = event.GetEventObject()
    varName = item.GetName()

    if varName == EstVariables[0]:
        Env.PowAvail = float(item.GetValue())
    elif varName == EstVariables[1]:
        Env.PowUsage = float(item.GetValue())
    elif varName == EstVariables[2]:
        Env.FuelUsage = float(item.GetValue())
    elif varName == EstVariables[3]:
        Env.PotWatUsage = float(item.GetValue())
    elif varName == EstVariables[4]:
        Env.BotWatUsage = float(item.GetValue())
    elif varName == EstVariables[5]:
        Env.AvgStorage = float(item.GetValue())
    elif varName == EstVariables[6]:
        Env.GrayWatGen = float(item.GetValue())
```

```
        elif varName == EstVariables[7]:
            Env.BlkWatGen = float(item.GetValue())
        elif varName == EstVariables[8]:
            Env.SWasteGen = float(item.GetValue())
        elif varName == EstVariables[9]:
            Env.FoodConsumed = float(item.GetValue())
        elif varName == EstVariables[10]:
            Env.AvgFacSqFt = float(item.GetValue())
        elif varName == EstVariables[11]:
            Env.AvgMaintenance = float(item.GetValue())
        else:
            pass

        self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()



#-------------------------------------------------------------------------|
#     Facility Selected for Changes                                       |
#-------------------------------------------------------------------------|
class SelectionPanel(wx.Panel):
    def __init__(self,parent,id):
        wx.Panel.__init__(self,parent,id, style=wx.BORDER_DOUBLE)

        self.current_selection = 3000
        hbox = wx.BoxSizer(wx.HORIZONTAL)
        vbox = wx.BoxSizer(wx.VERTICAL)
        hbox_i = wx.BoxSizer(wx.HORIZONTAL)

        title = wx.BoxSizer(wx.HORIZONTAL)
        self.FacCB = wx.ComboBox(self, -1, value="Facility", size=(170,-1), choices=FacilityNames, style=wx.CB_READONLY)
        self.FacCB.Bind(wx.EVT_COMBOBOX, self.onSelectFacility)
        title.Add(self.FacCB)

        #Text Input for displaying/changing percentage values
        self.PowUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PowerUse")
        self.FuelUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FuelUse")
        self.PWUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PotWaterUse")
        self.BWUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="BotWaterUse")
        self.SAUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="StorageUse")
        self.PerUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PersonnelUse")
        self.GWWUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="GrayWaterUse")
        self.BWWUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="BlkWaterUse")
        self.SWUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="SolidWasteUse")
        self.FoodUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FoodUse")
        self.AreaUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FootprintUse")
        self.MtnUsage = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="MaintenanceUse")
```

```
self.inputs1 = [self.PowUsage, self.FuelUsage, self.PWUsage, self.BWUsage, self.SAUsage, self.PerUsage,
        self.GWWUsage, self.BWWUsage, self.SWUsage, self.FoodUsage, self.AreaUsage, self.MtnUsage]

grid = wx.FlexGridSizer(13, 3, 8, 5)
for item in self.inputs1:
    item.Bind(wx.EVT_TEXT_ENTER, self.UpdatePercValue)
    grid.Add(wx.StaticText(self, -1, FacVariables[self.inputs1.index(item)]))
    grid.Add(item)
    grid.Add(wx.StaticText(self, -1, '%'), wx.ALIGN_LEFT)

#Constants Checkboxes
self.PowConst = wx.CheckBox(self, -1, 'Constant', name="PowerUse")
self.FuelConst = wx.CheckBox(self, -1, 'Constant', name="FuelUse")
self.PWConst = wx.CheckBox(self, -1, 'Constant', name="PotWatUse")
self.BWConst = wx.CheckBox(self, -1, 'Constant', name="BotWatUse")
self.SAConst = wx.CheckBox(self, -1, 'Constant', name="StorageUse")
self.PerConst = wx.CheckBox(self, -1, 'Constant', name="PersonnelUse")
self.GWWConst = wx.CheckBox(self, -1, 'Constant', name="GrayWaterUse")
self.BWWConst = wx.CheckBox(self, -1, 'Constant', name="BlkWaterUse")
self.SWConst = wx.CheckBox(self, -1, 'Constant', name="SolidWasteUse")
self.FoodConst = wx.CheckBox(self, -1, 'Constant', name='FoodUse')
self.AreaConst = wx.CheckBox(self, -1, 'Constant', name="FootprintUse")
self.MtnConst = wx.CheckBox(self, -1, 'Constant', name="MaintenanceUse")
self.inputs2 = [self.PowConst, self.FuelConst, self.PWConst, self.BWConst, self.SAConst, self.PerConst,
        self.GWWConst, self.BWWConst, self.SWConst, self.FoodConst, self.AreaConst, self.MtnConst]
for item in self.inputs2:
    item.Bind(wx.EVT_CHECKBOX, self.OnSetConstant)

#Text inputs for displaying/changing constants values
self.PowConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PowerUse")
self.FuelConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FuelUse")
self.PWConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PotWatUse")
self.BWConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="BotWatUse")
self.SAConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="StorageUse")
self.PerConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="PersonnelUse")
self.GWWConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="GrayWaterUse")
self.BWWConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="BlkWaterUse")
self.SWConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="SolidWasteUse")
self.FoodConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FoodUse")
self.AreaConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="FootprintUse")
self.MtnConstVal = wx.TextCtrl(self, -1, '0.0', size=(50,-1), style = wx.TE_PROCESS_ENTER, name="MaintenanceUse")
self.inputs3 = [self.PowConstVal, self.FuelConstVal, self.PWConstVal, self.BWConstVal, self.SAConstVal, self.PerConstVal,
        self.GWWConstVal, self.BWWConstVal, self.SWConstVal, self.FoodConstVal, self.AreaConstVal, self.MtnConstVal]
grid1 = wx.FlexGridSizer(13,3,8,5)

for item in self.inputs3:
    item.Bind(wx.EVT_TEXT_ENTER, self.UpdateConstValue)
```

```
            grid1.Add(self.inputs2[self.inputs3.index(item)])
            item.Disable()
            grid1.Add(item)
            grid1.Add(wx.StaticText(self, -1, VariableUnits[self.inputs3.index(item)]))

        hbox_i.Add(grid, 0)
        hbox_i.Add(grid1, 0, wx.LEFT, 20)
        vbox.Add(title,0,wx.ALL,10)
        vbox.Add(hbox_i, 0, wx.ALL, 10)

        #Displaying individual facility totals
        title2 = wx.StaticText(self, -1, 'Totals: ')
        vbox2 = wx.BoxSizer(wx.VERTICAL)
        self.PowTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.FuelTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.PotWatTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.BotWatTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.StorageTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.PersonnelTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.GrayWatTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.BlkWatTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.SWasteTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.FoodTotal = wx.StaticText(self,-1,'0', style=wx.BORDER_DOUBLE)
        self.AreaTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        self.MtnTotal = wx.StaticText(self, -1, '0', style=wx.BORDER_DOUBLE)
        total_items = [self.PowTotal, self.FuelTotal, self.PotWatTotal, self.BotWatTotal, self.StorageTotal,
                    self.PersonnelTotal, self.GrayWatTotal, self.BlkWatTotal, self.SWasteTotal, self.FoodTotal,
                    self.AreaTotal, self.MtnTotal]
        grid2 = wx.GridSizer(13,1,10,0)
        for item in total_items:
            grid2.Add(item)

        vbox2.Add(title2, 0, wx.TOP, 20)
        vbox2.Add(grid2, 0, wx.TOP | wx.LEFT, 13)

        hbox.Add(vbox, 0)
        hbox.Add(vbox2, 0, wx.RIGHT, 35)
        self.SetSizer(hbox)

    def onSelectFacility(self,event):
        item = event.GetSelection()
        #item = item - 1
        self.current_selection = item
        #Change default values
        #Set true for constants
        count = 0
        if Facility[self.current_selection].constants[count] == 1:
```

```
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].PowerUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].PowerUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].FuelUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].FuelUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].PotWaterUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].PotWaterUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].BotWaterUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].BotWaterUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].StorageUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].StorageUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
```

```
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].PersonnelUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].PersonnelUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].GrayWaterUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].GrayWaterUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].BlkWaterUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].BlkWaterUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].SolidWasteUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].SolidWasteUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].FoodUse))
else:
    self.inputs2[count].SetValue(False)
    self.inputs1[count].ChangeValue(str(Facility[self.current_selection].FoodUse))
count = count + 1
if Facility[self.current_selection].constants[count] == 1:
    self.inputs1[count].Disable()
    self.inputs2[count].SetValue(True)
    self.inputs3[count].Enable()
    self.inputs3[count].ChangeValue(str(Facility[self.current_selection].FootprintUse))
```

```
    else:
        self.inputs2[count].SetValue(False)
        self.inputs1[count].ChangeValue(str(Facility[self.current_selection].FootprintUse))
    count = count + 1
    if Facility[self.current_selection].constants[count] == 1:
        self.inputs1[count].Disable()
        self.inputs2[count].SetValue(True)
        self.inputs3[count].Enable()
        self.inputs3[count].ChangeValue(str(Facility[self.current_selection].MaintenanceUse))
    else:
        self.inputs2[count].SetValue(False)
        self.inputs1[count].ChangeValue(str(Facility[self.current_selection].MaintenanceUse))
    count = count + 1

    #Display calculated total values for facility
    self.PowTotal.SetLabel(str(Facility[item].PowerReq))
    self.FuelTotal.SetLabel(str(Facility[item].FuelReq))
    self.PotWatTotal.SetLabel(str(Facility[item].PotWaterReq))
    self.BotWatTotal.SetLabel(str(Facility[item].BotWaterReq))
    self.StorageTotal.SetLabel(str(Facility[item].StorageReq))
    self.PersonnelTotal.SetLabel(str(Facility[item].PersonnelReq))
    self.GrayWatTotal.SetLabel(str(Facility[item].GrayWaterReq))
    self.BlkWatTotal.SetLabel(str(Facility[item].BlkWaterReq))
    self.SWasteTotal.SetLabel(str(Facility[item].SolidWasteReq))
    self.FoodTotal.SetLabel(str(Facility[item].FoodReq))
    self.AreaTotal.SetLabel(str(Facility[item].FootprintReq))
    self.MtnTotal.SetLabel(str(Facility[item].MaintenanceReq))


def UpdatePercValue(self,event):
    if self.current_selection == 3000:
        dlg = wx.MessageDialog(self, "No Facility has been selected. Please select a facility first.",'Error', wx.OK | wx.ICON_ERROR)
        dlg.ShowModal()
        dlg.Destroy()
    else:
        item = event.GetEventObject()
        varName = item.GetName()
        if varName == 'PowerUse':
            Facility[self.current_selection].PowerUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.PowTotal.SetLabel(str(Facility[self.current_selection].PowerReq))
        elif varName == 'FuelUse':
            Facility[self.current_selection].FuelUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.FuelTotal.SetLabel(str(Facility[self.current_selection].FuelReq))
        elif varName == 'PotWaterUse':
            Facility[self.current_selection].PotWaterUse = float(item.GetValue())
```

```
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.PotWatTotal.SetLabel(str(Facility[self.current_selection].PotWaterReq))
        elif varName == 'BotWaterUse':
            Facility[self.current_selection].BotWaterUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.BotWatTotal.SetLabel(str(Facility[self.current_selection].BotWaterReq))
        elif varName == 'StorageUse':
            Facility[self.current_selection].StorageUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.StorageTotal.SetLabel(str(Facility[self.current_selection].StorageReq))
        elif varName == 'PersonnelUse':
            Facility[self.current_selection].PersonnelUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.PersonnelTotal.SetLabel(str(Facility[self.current_selection].PersonnelReq))
        elif varName == 'GrayWaterUse':
            Facility[self.current_selection].GrayWaterUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.GrayWatTotal.SetLabel(str(Facility[self.current_selection].GrayWaterReq))
        elif varName == 'BlkWaterUse':
            Facility[self.current_selection].BlkWaterUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.BlkWatTotal.SetLabel(str(Facility[self.current_selection].BlkWaterReq))
        elif varName == 'SolidWasteUse':
            Facility[self.current_selection].SolidWasteUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.SWasteTotal.SetLabel(str(Facility[self.current_selection].SolidWasteReq))
        elif varName == 'FoodUse':
            Facility[self.current_selection].FoodUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.FoodTotal.SetLabel(str(Facility[self.current_selection].FoodReq))
        elif varName == 'FootprintUse':
            Facility[self.current_selection].FootprintUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.AreaTotal.SetLabel(str(Facility[self.current_selection].FootprintReq))
        elif varName == 'MaintenanceUse':
            Facility[self.current_selection].MaintenanceUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.MtnTotal.SetLabel(str(Facility[self.current_selection].MaintenanceReq))
        else:
            pass

    def UpdateConstValue(self,event):
        if self.current_selection == 3000:
            dlg = wx.MessageDialog(self, "No Facility has been selected. Please select a facility first.",'Error', wx.OK | wx.ICON_ERROR)
            dlg.ShowModal()
            dlg.Destroy()
        else:
```

```
item = event.GetEventObject()
varName = item.GetName()
if varName == 'PowerUse':
    Facility[self.current_selection].PowerUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.PowTotal.SetLabel(str(Facility[self.current_selection].PowerReq))
elif varName == 'FuelUse':
    Facility[self.current_selection].FuelUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.FuelTotal.SetLabel(str(Facility[self.current_selection].FuelReq))
elif varName == 'PotWaterUse':
    Facility[self.current_selection].PotWaterUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.PotWatTotal.SetLabel(str(Facility[self.current_selection].PotWaterReq))
elif varName == 'BotWaterUse':
    Facility[self.current_selection].BotWaterUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.BotWatTotal.SetLabel(str(Facility[self.current_selection].BotWaterReq))
elif varName == 'StorageUse':
    Facility[self.current_selection].StorageUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.StorageTotal.SetLabel(str(Facility[self.current_selection].StorageReq))
elif varName == 'PersonnelUse':
    Facility[self.current_selection].PersonnelUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.PersonnelTotal.SetLabel(str(Facility[self.current_selection].PersonnelReq))
elif varName == 'GrayWaterUse':
    Facility[self.current_selection].GrayWaterUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.GrayWatTotal.SetLabel(str(Facility[self.current_selection].GrayWaterReq))
elif varName == 'BlkWaterUse':
    Facility[self.current_selection].BlkWaterUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.BlkWatTotal.SetLabel(str(Facility[self.current_selection].BlkWaterReq))
elif varName == 'SolidWasteUse':
    Facility[self.current_selection].SolidWasteUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.SWasteTotal.SetLabel(str(Facility[self.current_selection].SolidWasteReq))
elif varName == 'FoodUse':
    Facility[self.current_selection].FoodUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.FoodTotal.SetLabel(str(Facility[self.current_selection].FoodReq))
elif varName == 'FootprintUse':
    Facility[self.current_selection].FootprintUse = float(item.GetValue())
    self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
    self.AreaTotal.SetLabel(str(Facility[self.current_selection].FootprintReq))
elif varName == 'MaintenanceUse':
```

```
            Facility[self.current_selection].MaintenanceUse = float(item.GetValue())
            self.GetParent().GetParent().GetParent().Panel2.UpdateTotals()
            self.MtnTotal.SetLabel(str(Facility[self.current_selection].MaintenanceReq))
        else:
            pass

    def OnSetConstant(self,event):
        if self.current_selection == 3000:
            dlg = wx.MessageDialog(self, "No Facility has been selected. Please select a facility first.",'Error', wx.OK | wx.ICON_ERROR)
            dlg.ShowModal()
            dlg.Destroy()
        else:
            item = event.GetEventObject()
            varName = item.GetName()
            variables = ['PowerUse','FuelUse','PotWaterUse','BotWaterUse','StorageUse','PersonnelUse',
                    'GrayWaterUse','BlkWaterUse','SolidWasteUse','FootprintUse','MaintenanceUse']
            if item.IsChecked():
                Facility[self.current_selection].constants[variables.index(varName)] = 1
                self.inputs1[variables.index(varName)].Disable()
                self.inputs3[variables.index(varName)].Enable()
            else:
                Facility[self.current_selection].constants[variables.index(varName)] = 0
                self.inputs1[variables.index(varName)].Enable()
                self.inputs3[variables.index(varName)].Disable()


#-------------------------------------------------------------------------|
#     Calculated Totals                                                   |
#-------------------------------------------------------------------------|
class CalcTotalsPanel(wx.Panel):
    def __init__(self, parent, id):
        wx.Panel.__init__(self, parent, id)

        vbox = wx.BoxSizer(wx.VERTICAL)
        title = wx.StaticText(self, -1, 'Calculated Totals:')
        self.PowTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.PowGen = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.FuelTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.PotWatTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.BotWatTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.StorageTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.PersonnelTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.GrayWatTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.BlkWatTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.SWasteTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.FoodTotal = wx.StaticText(self,-1,'0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.AreaTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
```

```
        self.MtnTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        self.PopTotal = wx.StaticText(self, -1, '0', size=(75,-1), style=wx.BORDER_DOUBLE)
        total_items = [self.PowTotal, self.PowGen, self.FuelTotal, self.PotWatTotal, self.BotWatTotal, self.StorageTotal,
                       self.PersonnelTotal, self.GrayWatTotal, self.BlkWatTotal, self.SWasteTotal, self.FoodTotal,
                       self.AreaTotal, self.MtnTotal, self.PopTotal]
        grid = wx.FlexGridSizer(4, 8, 0, 10)
        for item in total_items:
          grid.Add(wx.StaticText(self, -1, TotalVariables[total_items.index(item)]), 0, wx.ALIGN_RIGHT)
          grid.Add(item, 0)
        vbox.Add(title,0,wx.ALL,5)
        vbox.Add(grid, 0, wx.ALL, 5)

        self.SetSizer(vbox)

#-----------------------------------------------------------------------|
#     Main Window                                                       |
#-----------------------------------------------------------------------|
class Main(wx.Frame):
    def __init__(self,parent,id,title):
        wx.Frame.__init__(self,parent,id, title, size=(975,650))


        self.GetPopulation()
        self.UpdateEnv()

        Interface = wx.Panel(self, -1)

        vbox = wx.BoxSizer(wx.VERTICAL)

        #Overview Information
        self.Panel1 = OverviewPanel(Interface, -1)

        # Input Content
        Content = wx.Panel(Interface, -1, style=wx.BORDER_RAISED)

        hbox = wx.BoxSizer(wx.HORIZONTAL)

        self.Panel2 = FacilityListPanel(Content, -1)
        self.Panel3 = EstTotalsPanel(Content,-1)
        self.Panel4 = SelectionPanel(Content, -1)

        hbox.Add(self.Panel2, 0, wx.ALL, 10)
        hbox.Add(self.Panel3, 0, wx.ALL, 10)
        hbox.Add(self.Panel4, 0, wx.ALL, 10)
        Content.SetSizer(hbox)

        # Totals
```

```
        self.Panel5 = CalcTotalsPanel(Interface, -1)

        vbox.Add(self.Panel1, 0, wx.TOP | wx.RIGHT | wx.LEFT, 10)
        vbox.Add(self.Panel5, 0, wx.LEFT, 10)
        vbox.Add(Content, 1, wx.RIGHT | wx.LEFT | wx.BOTTOM, 10)


        Interface.SetSizer(vbox)

        self.Show(True)
        #self.Centre()

    def GetPopulation(self):
        dia = wx.TextEntryDialog(self,'Please Enter Operational Soldier Population','Operational Soldier Population','0',style=wx.OK)
        if dia.ShowModal()== wx.ID_OK:
            self.PopChoice = float(dia.GetValue())
        dia.Destroy()
        self.LoadData()

    def LoadData(self):
        if self.PopChoice >= 100 and self.PopChoice <= 200:
            self.xml_file = 'InputValues150.xml'
        elif self.PopChoice > 200 and self.PopChoice <= 400:
            self.xml_file = 'InputValues300.xml'
        elif self.PopChoice > 450 and self.PopChoice <= 2000:
            self.xml_file = 'InputValues600.xml'
        #elif self.PopChoice > 1000 and self.PopChoice <=2000:
        #    self.xml_file = 'InputValues1500.xml'
        else:
            dlg = wx.MessageDialog(self, "Soldier Population provided was out of the capabilities of this program. Please provide a value between 100
and 2000.",'Error', wx.OK | wx.ICON_ERROR)
            dlg.ShowModal()
            dlg.Destroy()
            self.GetPopulation()


    def UpdateEnv(self):
        Facility.pop()
        FacilityNames.pop()
        GetData(self.xml_file, Facility)
        EnvTemp = GetEnvData(self.xml_file)
        for t in range(len(Facility)):
            FacilityNames.append(Facility[t].Name)

        Env.AvgFacSqFt = EnvTemp.AvgFacSqFt
        Env.AvgMaintenance = EnvTemp.AvgMaintenance
        Env.AvgStorage = EnvTemp.AvgStorage
```

```
        Env.BlkWatGen = EnvTemp.BlkWatGen
        Env.BotWatUsage = EnvTemp.BotWatUsage
        Env.FoodConsumed = EnvTemp.FoodConsumed
        Env.FuelUsage = EnvTemp.FuelUsage
        Env.GrayWatGen = EnvTemp.GrayWatGen
        Env.PotWatUsage = EnvTemp.PotWatUsage
        Env.PowAvail = EnvTemp.PowAvail
        Env.PowUsage = EnvTemp.PowUsage
        Env.SWasteGen = EnvTemp.SWasteGen
        Env.SoldierPopulation = float(self.PopChoice)
        if self.xml_file == 'InputValues300.xml':
            Env.PersonnelFactor = 0.25
        if self.xml_file == 'InputValues600.xml':
            Env.PersonnelFactor = .44

GetData('BaseEmpty.xml', Facility)
Env = GetEnvData('BaseEmpty.xml')
#Env.SoldierPopulation = SoldPop
for t in range(len(Facility)):
    FacilityNames.append(Facility[t].Name)
app = wx.App()
Main(None, -1, 'Relationship Analysis Model')
app.MainLoop()
```

APPENDIX B

CODE – DataParser.py

DataParser.py

```
from xml.etree import ElementTree as ET

infile = 'InputValue150.xml'

Facility = []
Env = []
BaseVar = []
FacilityVar = []

class System:
    def __init__(self, param):
        self.Name = param[0]
        self.PowerUse = float(param[1])
        self.FuelUse = float(param[2])
        self.PotWaterUse = float(param[3])
        self.BotWaterUse = float(param[4])
        self.StorageUse = float(param[5])
        self.PersonnelUse = float(param[6])
        self.GrayWaterUse = float(param[7])
        self.BlkWaterUse = float(param[8])
        self.SolidWasteUse = float(param[9])
        self.FoodUse = float(param[10])
        self.FootprintUse = float(param[11])
        self.MaintenanceUse = float(param[12])

        self.PowerReq = float(param[13])
        self.FuelReq = float(param[14])
        self.PotWaterReq = float(param[15])
        self.BotWaterReq = float(param[16])
        self.StorageReq = float(param[17])
        self.PersonnelReq = float(param[18])
        self.GrayWaterReq = float(param[19])
        self.BlkWaterReq = float(param[20])
        self.SolidWasteReq = float(param[21])
        self.FoodReq = float(param[22])
        self.FootprintReq = float(param[23])
        self.MaintenanceReq = float(param[24])

        self.constants = [0]*12
```

```
        self.Variables = [self.PowerUse, self.FuelUse, self.PotWaterUse, self.BotWaterUse,
                          self.StorageUse, self.PersonnelUse, self.GrayWaterUse,
                          self.BlkWaterUse, self.SolidWasteUse, self.FoodUse, self.FootprintUse, self.MaintenanceUse]

        self.A1 = [-1,0,0,0,0,0,0,0,0,0,0,0] #Power Equation
        self.A2 = [0,-1,0,0,0,0,0,0,0,0,0,0] #Fuel Equation
        self.A3 = [0,0,-1,0,0,0,0,0,0,0,0,0] #Potable Water Equation
        self.A4 = [0,0,0,-1,0,0,0,0,0,0,0,0] #Bottled Water Equation
        self.A5 = [0,0,0,0,-1,0,0,0,0,0,0,0] #Storage Equation
        self.A6 = [0,0,0,0,0,-1,0,0,0,0,0,0] #Personnel Equation
        self.A7 = [0,0,0,0,0,0,-1,0,0,0,0,0] #Gray Water Equation
        self.A8 = [0,0,0,0,0,0,0,-1,0,0,0,0] #Black Water Equation
        self.A9 = [0,0,0,0,0,0,0,0,-1,0,0,0] #Solid Waste Equation
        self.A10 = [0,0,0,0,0,0,0,0,0,-1,0,0] #Food Consumed Equation
        self.A11 = [0,0,0,0,0,0,0,0,0,0,-1,0] #Footprint Equation
        self.A12 = [0,0,0,0,0,0,0,0,0,0,0,-1] #Maintenance Equation
        self.A = [self.A1, self.A2, self.A3, self.A4, self.A5, self.A6, self.A7, self.A8, self.A9, self.A10, self.A11, self.A12]
##      self.A = []
##      for item in items:
##          self.A.append(item)
        self.B = [0]*len(self.Variables)



        self.Switch = 0

    def SetConstants(self, const):
        for x in const:
            self.constants[31] = 1

class GetData:
    def __init__(self,XMLFile, Fac):

        tree = ET.parse(XMLFile)

        #Get Facility Values
        element = tree.find('Facility')
        x = tree.findall('Facility')

        for index in x:
            List = []
            const = []
            for node in index:
```

```
            check = node.get('IsConstant')
            if check == '1':
               const.append(len(List)-1)
            List.append(node.text)
            FacilityVar.append(node.tag)
         Fac.append(System(List))
         Fac[x.index(index)].SetConstants(const)

class GetEnvData:
   def __init__(self,XMLFile):
      tree = ET.parse(XMLFile)
      root = tree.getroot()
      env = root.keys()
      env.sort()
      List = []
      for node in env:
         List.append(root.get(node))
      param = List

      self.AvgFacSqFt = float(param[0])
      self.AvgMaintenance = float(param[1])
      self.AvgStorage = float(param[2])
      self.BlkWatGen = float(param[3])
      self.BotWatUsage = float(param[4])
      self.FoodConsumed = float(param[5])
      self.FuelUsage = float(param[6])
      self.GrayWatGen = float(param[7])
      self.PotWatUsage = float(param[8])
      self.PowAvail = float(param[9])
      self.PowUsage = float(param[10])
      self.SWasteGen = float(param[11])
      self.SoldierPopulation = 0
      self.PersonnelFactor = 0

      self.TotalPower = 0
      self.TotalFuel = 0
      self.TotalPotWat = 0
      self.TotalBotWat = 0
      self.TotalStorage = 0
      self.TotalPersonnel = 0
      self.TotalGrayWat = 0
      self.TotalBlkWat = 0
```

```
        self.TotalSWaste = 0
        self.TotalFood = 0
        self.TotalArea = 0
        self.TotalMaintenance = 0
        self.TotalPopulation = 0
        self.PowerGen = 0

        self.Variables = [self.PowUsage, self.FuelUsage, self.PotWatUsage, self.BotWatUsage,
                          self.AvgStorage, 1, self.GrayWatGen, self.BlkWatGen, self.SWasteGen,
                          self.FoodConsumed, self.AvgFacSqFt, self.AvgMaintenance]

class UpdateData:
    pass
```

APPENDIX C

CODE – Solver.py

Solver.py

```
from numpy import *
from scipy import *
from xml.etree import ElementTree as ET

class Solver:
    def __init__(self, Fac, Env):

        EnvVariables = [Env.PowUsage, Env.FuelUsage, Env.PotWatUsage, Env.BotWatUsage,
                        Env.AvgStorage, Env.PersonnelFactor, Env.GrayWatGen, Env.BlkWatGen, Env.SWasteGen,
                        Env.FoodConsumed, Env.AvgFacSqFt, Env.AvgMaintenance, Env.SoldierPopulation]

        self.count = 0
        for index in range(len(Fac)):
            if Fac[index].Switch == 1:
                self.count = self.count + 1
        self.CreateBlankMatrix()
        Solution = self.PopulateMatrix(Fac, EnvVariables)
        UpdateLibrary(Fac,Env,Solution)
        WriteNewFile(Fac,Env)

    def CreateBlankMatrix(self):
        self.TableA = []
        self.TableB = []
        self.n = 12+self.count*12
        self.TableB = [0]*self.n
        for i in range(self.n):
            self.TableA.append([0]*self.n)

    def PopulateMatrix(self,Fac,EnvVariables):

        l = [0,1,2,3,4,5,6,7,8,9,10,11]
        for i in range(len(l)):
            for j in range(i,self.n,len(l)):
                if i == j:
                    self.TableA[i][j] = -1
                else:
                    self.TableA[i][j] = 1
        self.TableB[5] = EnvVariables[-1]*(-1)

        FacCount = 0
```

```
        for FacIndex in range(len(Fac)):
            if Fac[FacIndex].Switch == 1:

                Variables = [Fac[FacIndex].PowerUse, Fac[FacIndex].FuelUse, Fac[FacIndex].PotWaterUse,
                        Fac[FacIndex].BotWaterUse, Fac[FacIndex].StorageUse, Fac[FacIndex].PersonnelUse,
                        Fac[FacIndex].GrayWaterUse, Fac[FacIndex].BlkWaterUse, Fac[FacIndex].SolidWasteUse,
                        Fac[FacIndex].FoodUse, Fac[FacIndex].FootprintUse, Fac[FacIndex].MaintenanceUse]

                FacCount = FacCount + 1
                startI = len(I)*(FacCount)
                h = 0
                for i in range(startI, (startI + len(I)), 1):
                    if Fac[FacIndex].constants[h] == 1:
                        self.TableA[i][i] = 1
                        self.TableB[i] = Variables[h]
                    else:
                        self.TableA[i][i] = -1
                        self.TableA[i][5] = Variables[h]*EnvVariables[h]
                    h = h + 1

        self.C = linalg.solve(self.TableA, self.TableB)



        #print "Table A"
        #for index in range(self.n):
        #    print self.TableA[index]
        #print "Table B"
        #print self.TableB
        #print self.C

        return self.C

class UpdateLibrary:
    def __init__(self,Fac,Env,Sol):
        Env.TotalPower = round(Sol[0]/1000, 2)
        Env.TotalFuel = round(Sol[1], 2)
        Env.TotalPotWat = round(Sol[2], 2)
        Env.TotalBotWat = round(Sol[3], 2)
        Env.TotalStorage = round(Sol[4], 2)
        Env.TotalPersonnel = round(Sol[5] - Env.SoldierPopulation, 2)
        Env.TotalGrayWat = round(Sol[6], 2)
        Env.TotalBlkWat = round(Sol[7], 2)
```

```
        Env.TotalSWaste = round(Sol[8], 2)
        Env.TotalFood = round(Sol[9], 2)
        Env.TotalArea = round(Sol[10], 2)
        Env.TotalMaintenance = round(Sol[11], 2)
        Env.TotalPopulation = round(Sol[5], 2)
        Env.PowerGen = round((Env.PowAvail*Env.TotalPopulation)/1000, 2)

        pos = 11
        for i in range(len(Fac)):
            if Fac[i].Switch == 1:
                Fac[i].PowerReq = round(Sol[pos+1],2)
                Fac[i].FuelReq = round(Sol[pos+2],2)
                Fac[i].PotWaterReq = round(Sol[pos+3],2)
                Fac[i].BotWaterReq = round(Sol[pos+4],2)
                Fac[i].StorageReq = round(Sol[pos+5],2)
                Fac[i].PersonnelReq = round(Sol[pos+6],2)
                Fac[i].GrayWaterReq = round(Sol[pos+7],2)
                Fac[i].BlkWaterReq = round(Sol[pos+8],2)
                Fac[i].SolidWasteReq = round(Sol[pos+9],2)
                Fac[i].FoodReq = round(Sol[pos+10],2)
                Fac[i].FootprintReq = round(Sol[pos+11],2)
                Fac[i].MaintenanceReq = round(Sol[pos+12],2)
                pos = pos + 12

class WriteNewFile:
    def __init__(self,Fac,Env):

        root = ET.Element("Base")

        #Write Initial Assumptions
        elements = ['SoldierPopulation','PowAvail','PowUsage','FuelUsage','PotWatUsage',
                'BotWatUsage','AvgStorage','GrayWatGen','BlkWatGen','SWasteGen',
                'FoodConsumed','AvgFacSqFt','AvgMaint']
        element_values = [Env.SoldierPopulation, Env.PowAvail, Env.PowUsage, Env.FuelUsage,
                    Env.PotWatUsage, Env.BotWatUsage, Env.AvgStorage, Env.GrayWatGen,
                    Env.BlkWatGen, Env.SWasteGen, Env.FoodConsumed,Env.AvgFacSqFt,
                    Env.AvgMaintenance]
        for i in range(len(elements)):
            root.set(elements[i], str(element_values[i]))

        #Write Total Calculated Values
        elements = ['TotalPower','TotalFuel','TotalPotWat','TotalBotWat','TotalStorage',
```

```
        'TotalPersonnel','TotalGrayWat','TotalBlkWat','TotalSWaste','TotalFood',
        'TotalArea','TotalMaintenance','TotalPopulation']
element_values = [Env.TotalPower, Env.TotalFuel, Env.TotalPotWat, Env.TotalBotWat,
            Env.TotalStorage, Env.TotalPersonnel, Env.TotalGrayWat, Env.TotalBlkWat,
            Env.TotalSWaste, Env.TotalFood, Env.TotalArea, Env.TotalMaintenance,
            Env.TotalPopulation]
for i in range(len(elements)):
   BaseTotal = ET.SubElement(root, elements[i])
   BaseTotal.text = str(element_values[i])

#Write Facilities
elements = ['Name','PowerUse','PotWaterUse','BotWaterUse','StorageUse',
        'PersonnelUse','GrayWaterUse','BlkWaterUse','SolidWasteUse',
        'FoodUse','FootprintUse','MaintenanceUse','PowerReq','FuelReq',
        'PotWaterReq','BotWaterReq','StorageReq','PersonnelReq',
        'GrayWaterReq','BlkWaterReq','SolidWasteReq','FoodReq',
        'FootprintReq','MaintenanceReq']

for i in range(len(Fac)):
   FacVar = []
   Facility = ET.SubElement(root, "Facility")
   element_values = [Fac[i].Name, Fac[i].PowerUse, Fac[i].FuelUse, Fac[i].PotWaterUse,
            Fac[i].BotWaterUse, Fac[i].StorageUse, Fac[i].PersonnelUse,
            Fac[i].GrayWaterUse, Fac[i].BlkWaterUse, Fac[i].SolidWasteUse,
            Fac[i].FoodUse, Fac[i].FootprintUse, Fac[i].MaintenanceUse,
            Fac[i].PowerReq, Fac[i].FuelReq, Fac[i].PotWaterReq,
            Fac[i].BotWaterReq, Fac[i].StorageReq, Fac[i].PersonnelReq,
            Fac[i].GrayWaterReq, Fac[i].BlkWaterReq, Fac[i].SolidWasteReq,
            Fac[i].FoodReq, Fac[i].FootprintReq, Fac[i].MaintenanceReq]

   for x in range(len(elements)):
      FacVar = ET.SubElement(Facility, elements[31])
      FacVar.text = str(element_values[31])
      if x < 8:
         FacVar.set("IsConstant", str(Fac[i].constants[31]))


#Write out to file
tree = ET.ElementTree(root)
tree.write("InputValues-User.xml")
```

BIBLIOGRAPHY

[1] S. Friedenthal, A. Moore, and R. Steiner, *A Practical Guide to SysML: The Systems Modeling Language*. Boston: Morgan Kaufmann OMG Press, 2009.

[2] A. W. Wymore, *Model-based systems engineering: an introduction to the mathematical theory of discrete systems and to the tricotyledon theory of system design.* Boca Raton: CRC Press, 1993.

[3] Noblis, "Sustainable Forward Operating Bases," Strategic Environmental Research and Development Program (SERDP)2010.

[4] A. Kande, "Integration of model-based systems engineering and virtual engineering tools for detailed design," Masters, Engineering Management and Systems Engineering, Missouri University of Science & Technology, 2010.

[5] OMG, "OMG MOF 2 XMI Mapping Specification," ed, August 2011.

[6] T. Trainor, D. Brazil, and T. Lindberg, "Building Knowledge from Organizational Experience: Approaches and Lessons Learned from US Army Base Camp Workshops," *Engineering Managment Journal,* vol. 20, pp. 37-45, 2008.

[7] *Systems Engineering Vision 2020,* T. Operations, September 2007.

[8] J. A. Estefan, "Survey of Model-Based Systems Engineering (MBSE) Methodologies," International Council on Systems Engineering (INCOSE): the Model-Based Systems Engineering (MBSE) InitiativeJune 2008.

[9] OMG. (Dec 13). *OMG Systems Modeling Language*. Available: www.omgsysml.org

[10] V. V. Patel, J. D. McGregor, and S. Goasguen, "SysML-based domain-specific executable workflows," presented at the 2010 4th Annual IEEE Systems Conference, 2010.

[11] C. J. J. Paredis, Y. Bernard, R. M. Burkhart, H.-P. d. Koning, S. Friedenthal, P. Fritzson, N. F. Rouquette, and W. Schamai, "An Overview of the SysML-Modelica Transformation Specification," in *the 2010 INCOSE International Symposium, Proceedings of*, Chicago, IL, 2010.

[12] C. A. Jones and F. L. Drake, *Python & XML*: O'Reilly & Associates, Inc., 2002.

[13] R. Karban, et al., "Exploring Model based Engineering for Large Telescopes - Getting Started with Descriptive Models.," presented at the Proceedings of SPIE - the International Society for Optical Engineering., 2008.

[14] A. Solyler, and S. Sala-Diakanda, "A Model-Based Systems Engineering Approach to Capturing Disaster Management Systems.," in *2010 IEE International Systems Conference Proceedings*, 2010.

[15] OMG. (Dec 14). *MBSE Wiki*. Available: http://www.omgwiki.org/MBSE/doku.php

[16] J. A. Haiar, J. C. Lewis, and H. G. Tiedeman, "Model-Based Engineering for Platform System Design," in *25th Digital Avionics Systems Conference, 2006 IEEE/AIAA*, 2006, pp. 1-12.

[17] *Real Property Master Planning for Army Installations*, D. o. t. Army, 2005.

[18] A. Soyler and S. Sala-Diakanda, "A model-based systems engineering approach to capturing disaster management systems," in *Systems Conference, 2010 4th Annual IEEE*, 2010, pp. 283-287.

[19] J. Williams, "Contingency Operations and the use of GeoBEST," in *Esri International User Conference*, San Diego, CA, 2002.

[20] *Construction and Base Camp Development in the USCENTCOM Area of Responsibility*, USCENTCOM, Dec 2007.

[21] *Base Camp Facility Standards for Contingency Operations*, USAREUR, Feb 2004.

[22] M. Nikolaidou, V. Dalakas, L. Mitsi, G. D. Kapos, and D. Anagnostopoulos, "A SysML Profile for Classical DEVS Simulators," in *Software Engineering Advances, 2008. ICSEA '08. The Third International Conference on*, 2008, pp. 445-450.

[23] Y. Ben Maissa and S. Mouline, "A SysML profile for wireless sensor networks modeling," in *I/V Communications and Mobile Network (ISVC), 2010 5th International Symposium*, 2010, pp. 1-4.

[24] "OMG UML 2.3 Specification," ed: Object Management Group, May 2010.

[25] D. Mertz. (2003, Dec 14). *XML Matters: Process XML in Python with ElementTree*. Available: http://www.ibm.com/developerworks/library/x-matters28/

[26] F. Lundh. (July 2007, Dec 14). *Elements and Element Trees*. Available: http://effbot.org/zone/element.htm

[27] *Nanosat-7 User's Guide: University Nanosat-7 Program*, A. F. R. Laboratory, February 2011.

[28] A. Stewart, "A Guide to the Establishment of a University Satellite Program," Masters, Mechanical and Aerospace Department, Missouri University of Science and Technology, 2007.

[29] S. Miller, "Managment of a University Satellite Program with Focus on a Refrigerant-Based Propulsion System.," Masters, Mechanical and Aerospace Department, Missouri University of Science and Technology, May 2010.

[30] B. Cole, Chris Delp, and Kenny Donahue, "Piloting Model Based Engineering Techniques for Spacecraft Concepts in Early Formation," presented at the INCOSE IS 2010, 2010.

[31] J. Gross, A. Reichwein, and S. Rudolph, "An Executable Unified Product Model Based on UML to Support Satellite Design," in *AIAA Space 2009 Conference & Exposition*, Pasadena, CA, 2009.

VITA

Dustin Scott Nottage was born in Boynton Beach, FL on November 5, 1985. For High School, he attended Trinity High School in Euless, TX where he was part of the inaugural class for the International Baccalaureate Program at the school. He earned a B.S. in Aerospace Engineering in May 2009 from Missouri University of Science and Technology. After graduating, he decided to pursue a Master's degree starting in the Fall of 2009. Dustin completed his M.S. in Systems Engineering in May 2012 from Missouri University of Science and Technology. During his time at Missouri University of Science and Technology, Dustin participated on the 2005 Solar House Design Team and the Satellite Design Team.