2014-06

Examination of modeling languages to allow quantitative analysis for model-based systems engineerin

Nutting, Joseph W.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

EXAMINATION OF MODELING LANGUAGES TO ALLOW QUANTITATIVE ANALYSIS FOR MODEL-BASED SYSTEMS ENGINEERING

by

Joseph W. Nutting

June 2014

Thesis Advisor: Eugene Paulo
Second Reader: Paul Beery

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** June 2014	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE
EXAMINATION OF MODELING LANGUAGES TO ALLOW QUANTITATIVE ANALYSIS FOR MODEL-BASED SYSTEMS ENGINEERING

5. FUNDING NUMBERS

6. AUTHOR(S) Joseph W. Nutting

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES)
N/A

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB protocol number ____N/A____.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE
A

13. ABSTRACT (maximum 200 words)

Model-based systems engineering (MBSE) needs a formal language, one defined with explicit rules between its elements, in order to support the use of formal modeling in systems engineering. This thesis examines desirable features in the context of quantitative modeling for systems engineering modeling language. Object Management Group's UML and SysML and Vitech's System Definition Language are then analyzed in terms of these features.

The first important feature is the capability for interoperability between different MBSE tools combined with the ability to integrate the use of specialty tools to interact with and manipulate the system model. Flexibility is necessary in describing and defining entities in the system modeling language. This allows supporting project specific concerns in the system semantics, making MBSE tool support simpler. Finally, support for non-fixed value properties for entities, particularly random variables, is essential to representing system behavior.

Existing system modeling languages have shortcomings that should be addressed to improve the conduct of MBSE. Random variables are inconsistently supported. Behavior modeling allows intermingling event timelines for different entities, preventing automated analysis of possible event sequences. Finally, support of parametric modeling is not universal, and the semantics for the use of black box entities to represent external analysis is ad-hoc.

14. SUBJECT TERMS MBSE, UML, SysML, Modeling Languages, Systems Engineering	**15. NUMBER OF PAGES** 71
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UU

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

EXAMINATION OF MODELING LANGUAGES TO ALLOW QUANTITATIVE ANALYSIS FOR MODEL-BASED SYSTEMS ENGINEERING

Joseph W. Nutting
Lieutenant, United States Navy
B.S., University of Idaho, 2006

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SYSTEMS ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
June 2014

Author: Joseph W. Nutting

Approved by: Eugene Paulo
Thesis Advisor

Paul Beery
Second Reader

Cliff Whitcomb
Chair, Department of Systems Engineering

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Model-based systems engineering (MBSE) needs a formal language, one defined with explicit rules between its elements, in order to support the use of formal modeling in systems engineering. This thesis examines desirable features in the context of quantitative modeling for systems engineering modeling language. Object Management Group's UML and SysML and Vitech's System Definition Language are then analyzed in terms of these features.

The first important feature is the capability for interoperability between different MBSE tools combined with the ability to integrate the use of specialty tools to interact with and manipulate the system model. Flexibility is necessary in describing and defining entities in the system modeling language. This allows supporting project specific concerns in the system semantics, making MBSE tool support simpler. Finally, support for non-fixed value properties for entities, particularly random variables, is essential to representing system behavior.

Existing system modeling languages have shortcomings that should be addressed to improve the conduct of MBSE. Random variables are inconsistently supported. Behavior modeling allows intermingling event timelines for different entities, preventing automated analysis of possible event sequences. Finally, support of parametric modeling is not universal, and the semantics for the use of black box entities to represent external analysis is ad-hoc.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF FIGURES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

BOM	Base Object Model
BPMN	Business Process Model & Notation
DOD	Department of Defense
DoDAF	Department of Defense Architecture Framework
EFFBD	enhanced functional flow block diagram
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
INCOSE	International Council on Systems Engineering
ISO	International Standards Organization
MBSE	model-based systems engineering
MoDAF	Ministry of Defense Architecture Framework
MOF	Meta Object Facility
OMG	Object Management Group
SDL	Systems Definition Language
SE	systems engineering
SISO	Simulation Interoperability Standards Organization
SysML	Systems Modeling Language
UML	Unified Modeling Language
UPDM	Unified Profile for DoDAF/MoDAF

THIS PAGE INTENTIONALLY LEFT BLANK

EXECUTIVE SUMMARY

Model-based systems engineering (MBSE) is a reaction to the increasing complexity of modern systems. The International Council on Systems Engineering (INCOSE) defines MBSE as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases." The formal modeling allows the use of software tools to assist in maintaining the consistency throughout the systems engineering project. This increase in automation allows systems engineers to make fewer errors and spend a greater portion of their effort on quality engineering.

Formal modeling requires a formal language. A formal language, unlike a natural language, is one that operates with explicitly defined rules. This avoids ambiguity and provides consistency. A good language for systems engineering has to be sufficiently expressive that all relevant aspects of the system and its behavior can be described, but at the same time not require years to learn.

This thesis examines modeling languages for systems engineering from the viewpoint of integrating quantitative analysis into MBSE. Two secondary topics are examined to better understand what features are desirable in modeling language for systems engineering: 1) What does the process for creating a model for a quantitative analysis in systems engineering entail, and 2) what processes in systems engineering (as described by ISO standard 15288) are affected by language choices for quantitative modeling.

0is a process model that was created in order to examine what is involved in quantitative modeling for systems engineering. This model is consistent with the processes of the modeling and simulation community, and also with the stepwise refinement used in many MBSE methodologies (Robinson 2011; INCOSE 2008). Two key considerations resulted from this model: First, there needs to be an existing architecture in which to conduct the analysis. A simplistic view of a quantitative model is

that it transforms a set of inputs into a set of outputs. Without establishing an architecture beforehand, it is unlikely that the inputs for a quantitative analysis will align with the systems engineering effort. This creates desirability for the quantitative model to be integrated into the architecture, so that creating one also creates the other, and implicitly maintains consistency between the two views.

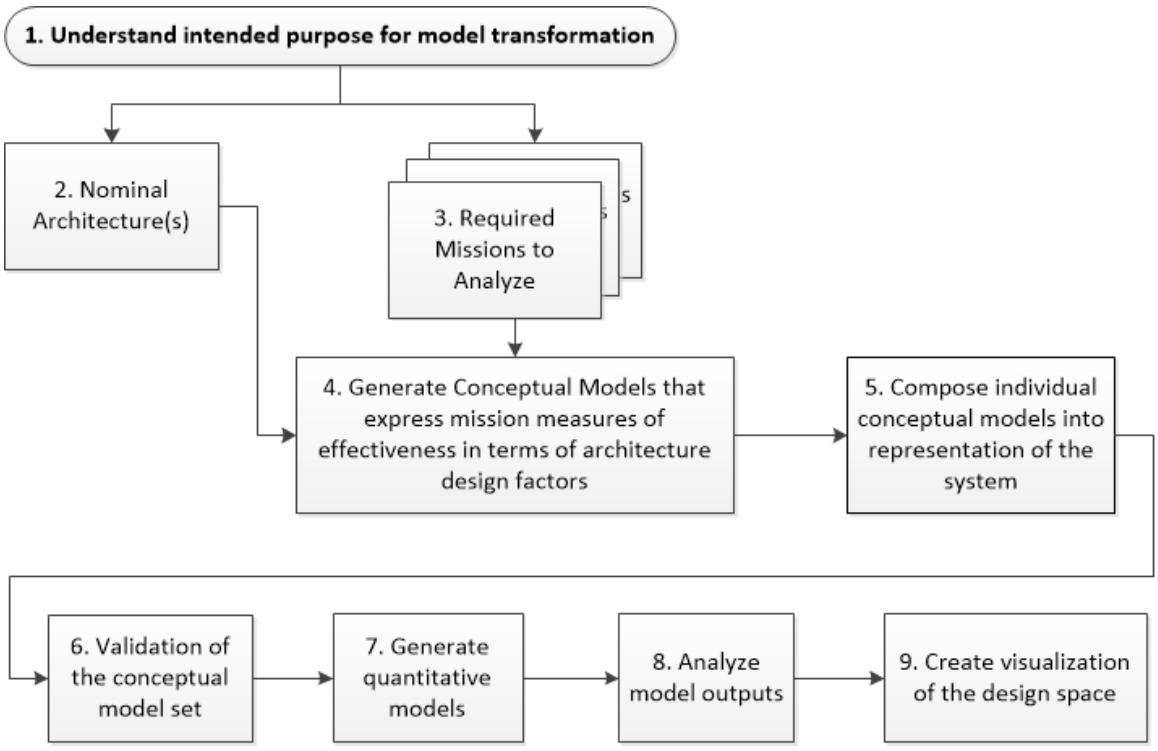

Figure 1 Quantitative Analysis Process Model

The second key consideration is in steps 5 and 6. The individual quantitative models need to be examined holistically, to understand how they relate to the system. A major concern is that interactions between differing areas of functionality may be omitted due to breaking down the modeling effort into more manageable pieces; explicitly considering whether the set of conceptual models is valid provides a deliberate opportunity to assess this concern. This consideration points to the desirability of incorporating quantitative analysis into the systems model to make this step easier to conduct.

The ISO standard 15288, Systems and Software Engineering—Systems Life Cycle Processes, has been adopted by INCOSE for categorizing and defining the differing processes where systems engineering is involved. Two of the processes where choice of modeling language for systems engineering and quantitative analysis are particularly important are infrastructure management and requirements engineering.

Infrastructure management has to take into consideration the capabilities and availabilities of software tools with which to conduct MBSE throughout the system life cycle. This implies that an open, or non-proprietary, language is preferable. Ideally, the language is standardized to reduce the possibility of a single software vendor making changes to the language, breaking interoperability with other software tools. Proprietary languages are at risk of the vendor exiting the market and forcing a decision on whether to maintain with increasingly obsolete tools or to undergo a costly rebuild of the system model in a different language. Additionally, the use of a proprietary language can complicate adapting custom tools between projects or through an industry.

The requirements engineering process benefits greatly from integration of quantitative modeling into the system model. The ability to view how a proposed requirement affects multiple aspects of the system simultaneously supports better decision making. Furthermore, having decision-makers examine these tradeoffs before setting final requirements allows the use of Richard Balling's "design by shopping" paradigm. This concept is to have decision-makers be able to examine the trade-space and understand what good designs (in terms of being Pareto optimal) look like before prioritizing the various requirements. This is a more natural decision making sequence similar to how individuals make decisions such as buying a new car or a house, and can lead to a better set of requirements from which to build the system.

Four important characteristics for a systems modeling language were identified: Support for interoperability and integration between multiple software tools, flexibility, support for non-fixed value parameters (such as random variables), and composability of executable architectures. Several systems modeling languages were analyzed based on these characteristics: Unified Modeling Language (UML), Systems Modeling Language (SysML), and Vitech's Schema Definition Language (SDL).

All three languages had shortcomings in supporting composability of executable architectures, due to the lack of semantics to clearly distinguish between events among different entities and internal events. They allow systems engineers to specify event sequences that are intermingled and miss possible interactions. There are approaches to resolve this limitation. The Monterey Phoenix project defines a formal language that clearly distinguishes which events are shared and which belong only to a single entity (Auguston et al. 2012). This enables automatic generation of all possible event sequences for analysis.

UML and SysML have good support for integration and interoperability with multiple tools. In addition to their specifications being open and available, both implement the XML metamodel interchange, a standard way to storing the model data to support interoperability between software tools. SysML and SDL have partial support for ISO 10303-233: Systems engineering data representation, a model interchange standard for systems engineering. It only specifies a limited subset of model views and is not meant to encompass the entire system engineering modeling effort.

Both SysML and UML are flexible. Their metamodel can be adapted and extended by users to accommodate specific needs. The concept of stereotypes and extensions allows the use of specialization type relationships that preserve the relationships of the general type of model entity.

SDL allows defining new model entities and relationships between them, providing some level of flexibility. It does not allow binding relationships between properties of different system, limiting the use of parametric relationships in the system model.

All three languages have poor support for non-fixed value properties. SysML has non-normative guidance on defining random variable datatypes but does not provide any guidance on how to handle random variables that are defined implicitly by some sort of parametric relationship between random variables. The UML standard provides no

guidance. Both languages can have ad-hoc support for random variables, but ad-hoc support by its nature is inconsistent, and for a base engineering concern such as random variables, it is insufficient.

SDL has support for certain properties related to behavior models to be defined as random variables. Systemic support for arbitrary properties to be defined as random variables does not exist. The CORE MBSE tool, similar to many SysML MBSE tools, has support for scripting that can be used to create this support, but without standardization, the ability to use this consistently between different tools and across projects is limited.

Future work is needed in three areas in languages for MBSE:

- Understanding and developing the appropriate semantics for the use of black box entities in the area of parametric modeling and analysis.
- Semantics and syntax for use of random variables in properties of model entities, including those defined by relationships between random variables.
- Modeling techniques for system behaviors that make it easier to avoid specifying expected event sequences versus possible event sequences.

LIST OF REFERENCES

Auguston, Mikahil, Clifford Whitcomb, and Kristin Giammarco. 2012. "A New Approach to System and Software Architecture Specifications Based on Behavior Models." http://hdl.handle.net/10945/14782.

INCOSE. 2008. *Survey of Model-Based Systems Engineering (MBSE) Methodologies Rev B*, by Jeff A. Estefan. INCOSE-TD-2007-003-02. http://www.incose.org/productspubs/pdf/techdata/mttc/mbse_methodology_survey_2008-0610_revb-jae2.pdf.

Robinson, Stewart. "Tutorial: Choosing What to Model - Conceptual Modeling for Simulation." Paper presented at the 2012 Winter Simulation Conference, Berlin, Germany, December 2012.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

> *Systems Engineering is an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle.*
>
> —INCOSE SE Handbook

The critical element of systems engineering (SE) is communication. Miscommunication between stakeholders leads to systems that do not address the right problems. Miscommunication between engineers leads to components that have the wrong functionality and do not integrate. Miscommunication between users leads to systems that do not fulfill their purpose. A systems engineer's most important task is to facilitate useful and unambiguous communication within and among groups of people.

Historically, systems were small enough that facilitating communications was not a dedicated role. Engineering diagrams and verbal discussions were sufficient to facilitate communications between stakeholders. As projects have become more complex, this is no longer adequate. People who never meet in person work on the same project. Engineers from different disciplines do not share common diagrams. Similar to governance, where bureaucracies are necessary to support consistent application of laws and regulations, some sort of formalization was needed to provide consistent quality for the engineering process.

The logical step was through standardized methodology and supporting documentation. Standards such as ISO 15288, "Systems and software engineering - System life cycle processes" provide a standardized set of processes such as requirements analysis, architectural design, and integration. The use of well-defined processes and frameworks enables all participants in the engineering process to understand the project's technical development and where to find information that is relevant to their work.

Even standardized documentation can only assist the people managing a project so much. As a system becomes bigger, identifying what elements may be affected by a

change is more difficult. The manpower required to ensure that all the documentation is consistent grows non-linearly with the amount of documentation. This growth has led to a need to move away from a document-based systems engineering process to one that is based on a coherent model. Initial approaches included hybrid methodologies such as use of central requirements repositories such as Dynamic Object-Oriented Requirements System, but the natural evolution is to some form of model-based systems engineering (MBSE) where the systems engineering efforts involve a single master model that contains all the information on the system.

A. MODEL-BASED SYSTEMS ENGINEERING

> *A model is "a representation of a real world process, device, or concept."*
>
> —IEEE Std 24765-2010

Strictly speaking, even a document providing a written description of a system is a model. It provides a representation of a system, and may even include diagrams. That does not mean it is MBSE. The International Council on Systems Engineering (INCOSE) defines MBSE in its *Systems Engineering Vision 2020* as

> Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases.

The two distinguishing features from traditional systems engineering are the "formalized application" of modeling and the implication that there is a master system model. These features cannot easily be separated; a master system model that is not supported by a formal structure will tend to give rise to parallel models that do not have enforced consistency. This is seen in traditional systems engineering, where although there may be a master document, secondary documents may be inconsistent and actually be authoritative on key details. Similarly, without a master system model the situation reduces to the same issues with documents where some human is manually evaluating differing models to ensure that their combined description of the system is not contradictory.

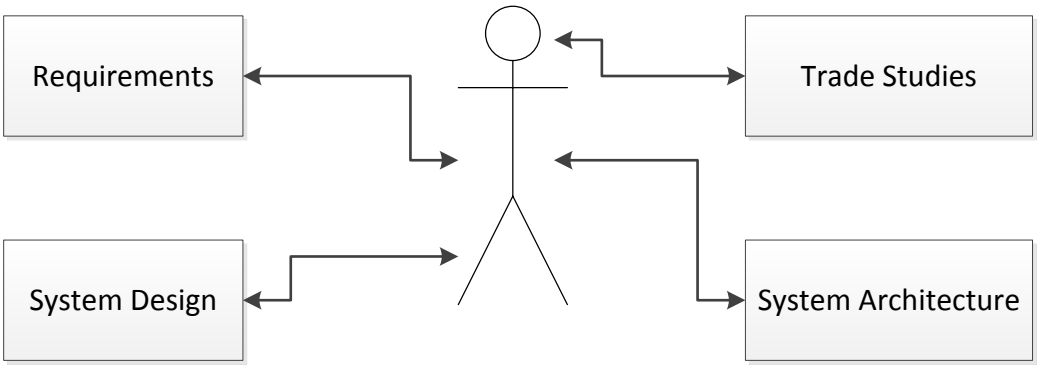

Figure 1. Traditional Systems Engineering. All Methods are Human Driven

Figure 1. provides a view of systems engineering focusing on the interaction between different products in the SE process. All the mediation between different products and processes is human driven. This makes consistency a primary concern for the systems engineer. Methods such as configuration control boards, standardized labeling, and tracking spreadsheets are used but these methods are people driven and have manual tools. Beyond these issues, making changes to any element of the system may involve updating multiple documents, creating more opportunities for errors.

One major objective of MBSE is to produce a coherent model of the system (Friendenthal, Moore and Steiner 2008, 17). The premise is that maintaining a logical and consistent model can be accomplished with software tools instead of through human effort. A potential view of how this would look to the systems engineer is shown in Figure 2. The MBSE tool supports linking the different products together and can both suppress the creation of inconsistencies in the first place, and automatically identify them for resolution as they occur. This enables the systems engineer to spend a greater fraction of time on performing competent engineering. Additionally, by being able to select intelligently portions of the system to view, the systems engineer can simultaneously have all the information available but restrict his focus to what is relevant.

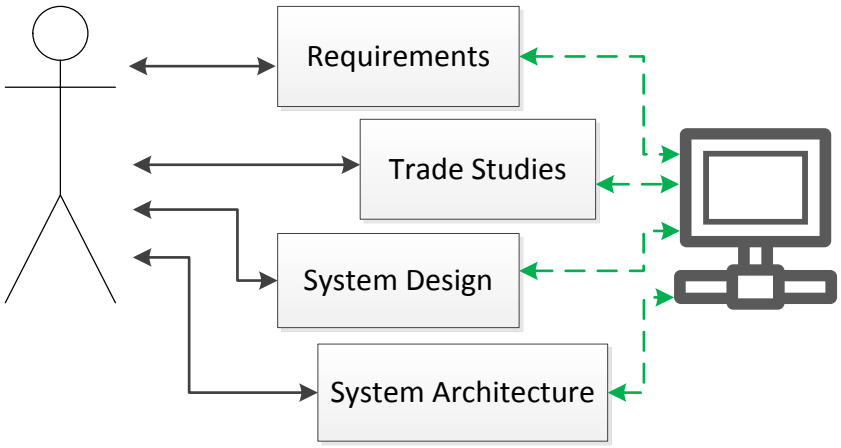

Figure 2. Model-based Systems Engineering: Software Tools Manage Coherency

An obvious consequence of the MBSE goal is a heavy reliance on software tools. If reading and manipulating a complete textual description of a system were practical, MBSE would be unnecessary. Software tools provide a means to focus on part of the system model to the exclusion of other elements.

At a level below the software tools is the modeling language that is used to describe the systems engineering effort. In order to efficiently process with software tools, it needs to be a formal language, which means that its structure and syntax is fully described by a set of rules. For example, programming languages and predicate logic are formal languages. If it is to be inclusive of all the system engineering products, it needs to be capable of more than just describing the components of the system design; it needs to be able to describe how the system operates and interacts with its environment.

The first generation of MBSE efforts primarily works with qualitative models. Qualitative models provide information such as “What requirements is this system designed to fulfill?” and “What functions does this system perform?” They are analogous to blueprints. However, qualitative models do not inherently provide information such as “How will the system, as designed, perform?” Models that contain this type of information are quantitative models. The incorporation of quantitative models into MBSE and the associated tools is critical in improving the ability for systems engineers to manage engineering.

B. RESEARCH QUESTIONS

1. Do existing modeling languages provide a "good enough" semantic structure for incorporation of quantitative modeling into MBSE?

An analogy to this concern is a general purpose programming language. There are several different programming paradigms such as procedural, object oriented, and functional. Attempting to use a paradigm with a language that is not compatible may be possible, but is inefficient and frustrating. This research question requires initial framing by the following research questions to better define "good enough."

2. Where should analytical modeling exist in the systems engineering domain?

This question addresses scope. There are two potential secondary research questions stemming from this question; first, where in the SE processes is analytical modeling involved? Second, is there a depth of detail where the analytical modeling is no longer a systems engineering concern but rather is a domain engineering concern? This question aids understanding how integral does the analytical model need to be to the systems engineering model of the system.

3. How should analytical modeling in MBSE be approached?

Creating a model is not a single-step process. The goal is to understand what products of the analytical modeling process should be represented within the systems engineering model. Examining this interaction will further understanding in what meaning, or semantics, needs to be communicated by a language used for MBSE.

C. SCOPE

While INCOSE backs SysML as the language of choice of systems engineering, SysML has limitations and software MBSE tool vendors are free to implement whatever language they choose that they believe best supports their users. Estefan's "Survey of Model-based Systems Engineering (MBSE) Methodologies" identified at least five modeling languages, although his work was not focused on languages but rather practices. Once niche languages that only intend to address subsections of systems

engineering are considered, there are too many to address in one document. This thesis will focus on three languages: Unified Modeling Language 2 (UML) and Systems Modeling Language (SysML) by the Object Management Group (OMG), and the Vitech System Definition Language (SDL) used by Vitech CORE for the following reasons:

SysML requires evaluation because it is supported by INCOSE, the most significant systems engineering professional body. Any consideration of languages for use by MBSE would be grossly lacking without analyzing this one. In order to understand the strengths and weaknesses of SysML, UML requires some level of examination. It is part of the foundation for SysML, so an examination of UML is important to better understand SysML. There are many variants such as the Unified Profile for DODAF/MODAF (UPDM) and Business Process Model & Notation (BPMN) that have origins in UML/SysML, but they will be omitted due to having similar structure.

Vitech SDL is included because the Naval Postgraduate School uses CORE within the systems engineering curriculum. It is not based on the same metamodel that forms the basis of SysML and UML, and was built for use primarily with a single MBSE tool, as opposed to OMG's languages. This led to some different structuring than SysML and UML, providing an alternate perspective on modeling languages for systems engineering.

D. RESEARCH METHODOLOGY

The research questions were examined from two viewpoints. The first viewpoint was from the modeling and simulation community. Modeling and simulation as a discipline has existed considerably longer than the MBSE effort, which did not become significant until the late 1990s. There is extensive experience in both working with quantitative models as well as model interoperability. Interoperability is a particular concern because the conjoinment of a description of a system architecture or design to a quantitative model is an exercise in interoperability.

The second viewpoint was the systems engineering community. A systems engineering language needs to reflect the needs of the systems engineering community,

which is diverse due to the interdisciplinary nature of SE. The INCOSE/OMG MBSE working groups were particularly valuable in understanding what issues were seen and approaches under consideration.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND

There are two major sources that have driven the development of MBSE: traditional systems engineering and software engineering. Software engineering had to deal with interfaces, version control, and modules, pushing for a non-document centric representation from the start. As software has become pervasive in complicated systems, the boundary between systems engineering and the software engineering domain has become less clear. This has led efforts to adapt techniques used to manage complex software into systems engineering.

In turn, there is a distinct divide in approaches to making MBSE more than just a representation of the system. There is the executable architecture view, where the system description can be directly used or translated use by a simulation tool for analysis. This appears to be linked to the software engineering tradition, where, in principle, if the software model is complete, then it should be able to be transformed into some sort of programming language code without human involvement. This is associated with model transformation research, although not exclusively.

At the opposite end of the spectrum is a parametric view. System parameters are used as inputs and/or outputs for an analytical model. This analytical model may itself be described within the system engineering model or it could be external to it. The abstraction provided by parametric models can assist in avoiding excessive modeling effort by focusing only on what is deemed important. Additionally, it fits better to certain types of analysis such as cost analysis that historically use parametric estimations.

This chapter is organized into two parts. The first part examines the executable architecture approach and the parametric approach in more detail. The second part discusses UML, SysML, and SDL in additional detail. This is necessary for analysis in Chapter V on the strengths and weaknesses of the languages in quantitative modeling.

A. APPROACHES TO QUANTITATIVE MBSE

1. Executable Architectures

Executable architectures promise to aid in understanding the dynamic behavior of a system with a minimum of additional work. There are two major branches of executable architecture related work on quantitative MBSE: those that aim to transform the system model into another model better suited for analysis, and those that aim to take existing behavioral representations, such as enhanced functional flow block diagrams (EFFBDs), within the system model and directly use it within a simulation.

a. Model Transformation

Model transformation has been a focus of OMG's efforts since they published their concept of model driven architecture (MDA) in 2000 (Soley 2000). OMG's goal is to be able to specify a software project at a high level of abstraction, such as in UML, and then be able to generate automatically the equivalent software code in a programming language capable of compiling into an executable program. Since SE is more than software, automatic generation of a product is not realistic. From the SE side, the goal is to specify the components of the system and see how they interact together.

Czarnecki and Helsen's paper, "Classification of Model Transformation Approaches," provides a good background on ways to approach model transformation as well as strengths and weaknesses of different approaches (Czarnecki 2003). Subsequently, the OMG Modeling and Simulation Interoperability challenge team has done significant work in this area (Peak and Zwemer 2011). They have defined a transformation from SysML to Modelica, an open source modeling language, and demonstrated the ability to use this to evaluate system performance and support engineering decisions (Paredis et al. 2010). In 2011, members published a paper detailing a more general approach to constructing a transformation between SysML and arbitrary modeling languages (Kerzhner, Jobe and Paredis 2011).

Challenges seen using model transformation is that there needs to be a defined relationship between elements in both languages. This makes use at high levels in the system architecture more difficult, and means a lot of initial effort to set up the libraries

in both languages needs to occur. An example of this for a single system component is shown in Figure 3. Without a well-defined system structure, model transformation seems impractical. It appears to be better suited for systems engineering once the architecture is established rather than for initial concept development.

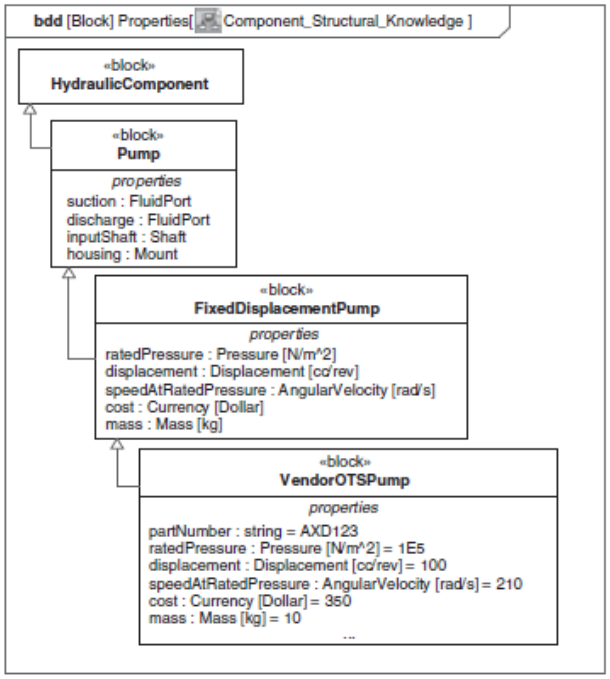

Figure 3. Detail Necessary for Model Transformation (from Kerzhner, Jobe, and Paredis 2011)

The benefits of model transformation are significant. Building the system model provides quantitative performance data on how well the system works, or if the design works at all. The concern that the performance model reflects a previous version of the system design goes away because as the system model changes, so does the performance model. Additionally, the segregation can improve model readability and provide granularity by allowing more refined modeling to be used later in the project as necessary without altering the system model. Finally, changing modeling languages, can allow analyzing concepts that are not easily expressed in the system modeling language, such as time-varying properties in SysML.

b. Executable Architectures

Executable architectures are more restricted than model transformations. System behavior models are used in a simulator to allow analysis of dynamic behavior such as system stalls from two activities that each wait for the other to complete before progressing. Executable architectures are a method that is not available in traditional SE due to the complete reliance on software tools. In current MBSE tools such as CORE and IBM Rhapsody, executable architectures rely partly on tool specific scripting languages to accommodate behavior that is more complex than specified by the behavior diagrams. This implies a weakness with current system modeling languages.

A difficulty with executable architectures is control mechanisms when the system has multiple independent entities that interact (Giammarco 2013). Many MBSE tools have limited parallelism in their simulators and the tools themselves cannot distinguish a diagram that has unintentional dependencies between the worldliness of different entities. This creates dependencies that do not exist in reality, causing the MBSE results not to match reality.

There has been work by the modeling and simulation community on this concern. The Simulation Interoperability Standards Organization (SISO) released the Base Object Model (BOM), meant to standardize data and event exchange between different members of a federated simulation (SISO 2006). While its primary focus is large simulations, it provides a standardized way to describe events in which an entity in a simulation participates, while encapsulating internal events within the activity. This formal separation resembles the ideas of the Monterey Phoenix project but from a different direction.

Additional work is needed in MBSE tools to achieve the potential of executable architectures. The number of possible sequences of events grows rapidly, and software assistance is necessary to identify sequences that fail by intention versus those that fail due to error in the system architecture and design.

2. Parametric Modeling

Parametric Modeling is the other means of quantitative analytical modeling. Properties of system elements form inputs to some sort of quantitative model. It can be as simple as a set of constraints evaluated to provide analysis of the system design. *Force = Mass * Acceleration* is an example of a parametric analysis. Alternatively, the quantitative model could be something as complex as the Department of Defense's STORM campaign simulation software. The distinguishing feature compared to executable architectures is that it is not necessarily linked to the system architecture. Implementations of parametric modeling within MBSE fall into two categories: those that are described internally to the system engineering model, and those that are external.

The first is explicit inclusion of the parametric relationship. An example from the SysML specification is shown in Figure 4. The fuel flow rate (ice.ft.FuelFlowRate) is calculated based off the fuel pressure (ice.fi.fuel.FuelPressure) and the fuel demand (ice.fi.FuelDemand) parameters. Software such as Paramagic from InterCax can interpret the parametric diagram and directly calculate a given attribute or send the equation to a more robust analysis tool such as MATLAB for a more detailed analysis. However, all the information about the relationship between the parameters is maintained within the system model. This assists in maintaining consistency, and if the relevant system entity that owns the parameter needs to change, the analysis can be updated to reflect this within the MBSE tool.

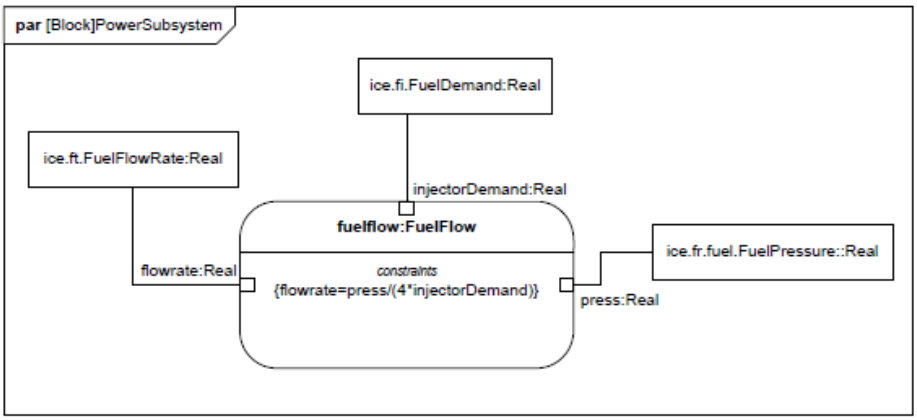

Figure 4. Example of Parametric Model (from OMG 2012)

The second category is when the quantitative model for the analysis is contained outside the system engineering model. It may be too complex for inclusion or be some already existing model that does not make sense to recreate. The simplest implementation is to read the system model and hardcode the appropriate parameters into the external model. To be more flexible, some sort of parametric diagram similar to the previous example may be used, where instead of including the constraint equation in the fuelflow constraint block as shown in Figure 4. , the fuelflow block acts as a placeholder for an external analysis. The external tool would examine the inputs (injectorDemand, press) and traverses the diagram until it reaches system attributes with values. This reduces the coupling between the two models, improving maintainability.

The strengths of the parametric approach lie in two areas. First, there are many tools capable of performing parametric analysis and dedicated software to support extracting and interpreting parametric models from SysML exists. Second, it allows the use of black box models where access to adequate information to create a proper model transformation is not available.

The weaknesses are that there are limits to what can be expressed parametrically. Important considerations such as system states are not easily expressible or may require duplicate work, increasing the costs to maintain consistency between the system model and the analytical model. As MBSE matures better solutions may be found.

B. SYSTEMS ENGINEERING LANGUAGES

1. Unified Modeling Language

a. History

The ultimate product from software engineering is the code that is executed by computing hardware. The designing and writing of the code has grown progressively more abstract as software has become more complex. Assembly languages were abstracted into procedural languages to allow more compact and understandable expressions. Procedural languages were then followed by object-oriented and functional languages that further abstracted the programming from the machine code result. Desires to further abstract the software engineering away from the actual programming led efforts

in the 1990s to use computer-based modeling to support software engineering (Weilkiens 2007, 144-145). Figure 5. provides an overview of this timeline leading to UML.

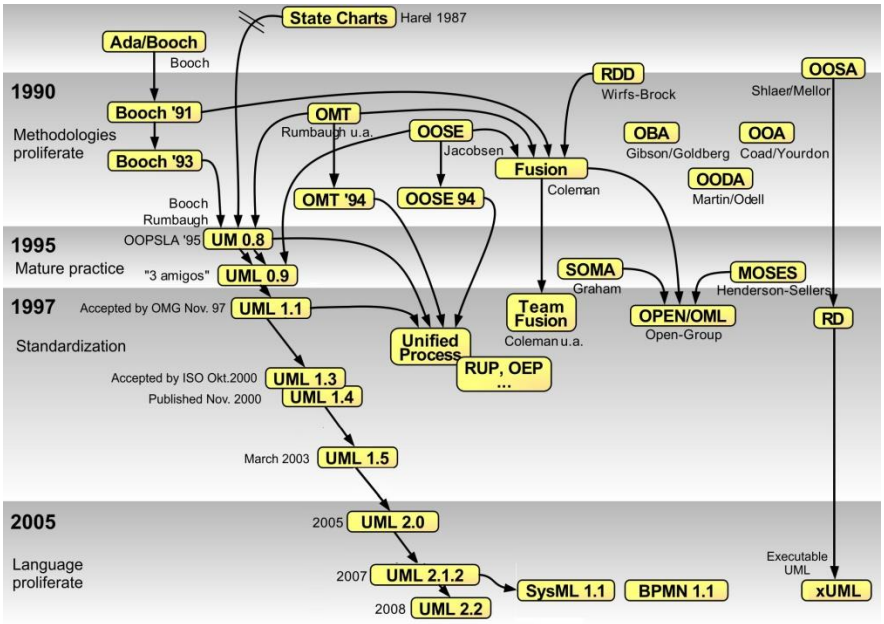

Figure 5. A History of Object Oriented Modeling Languages (from Zockoll et al. 2014)

This culminated in 1997 when the Object Management Group (OMG), a non-profit industry consortium, accepted the Unified Modeling Language (UML) specification as an industry standard. OMG was also responsible for having UML approved as an ISO standard (Weilkiens 2007; OMG 2000). This standardization process introduced stability and supported many software tool vendors, creating products that are capable of cross-tool support.

b. The Meta-Object Facility

UML is called the unified modeling language rather than the software modeling language because of its robust meta-model, the Meta Object Facility (MOF). The MOF provides the fundamental rules that UML follows, such as how associations between model elements are specified and how model elements themselves are described (OMG Meta Object Facility Core Specification 2013).

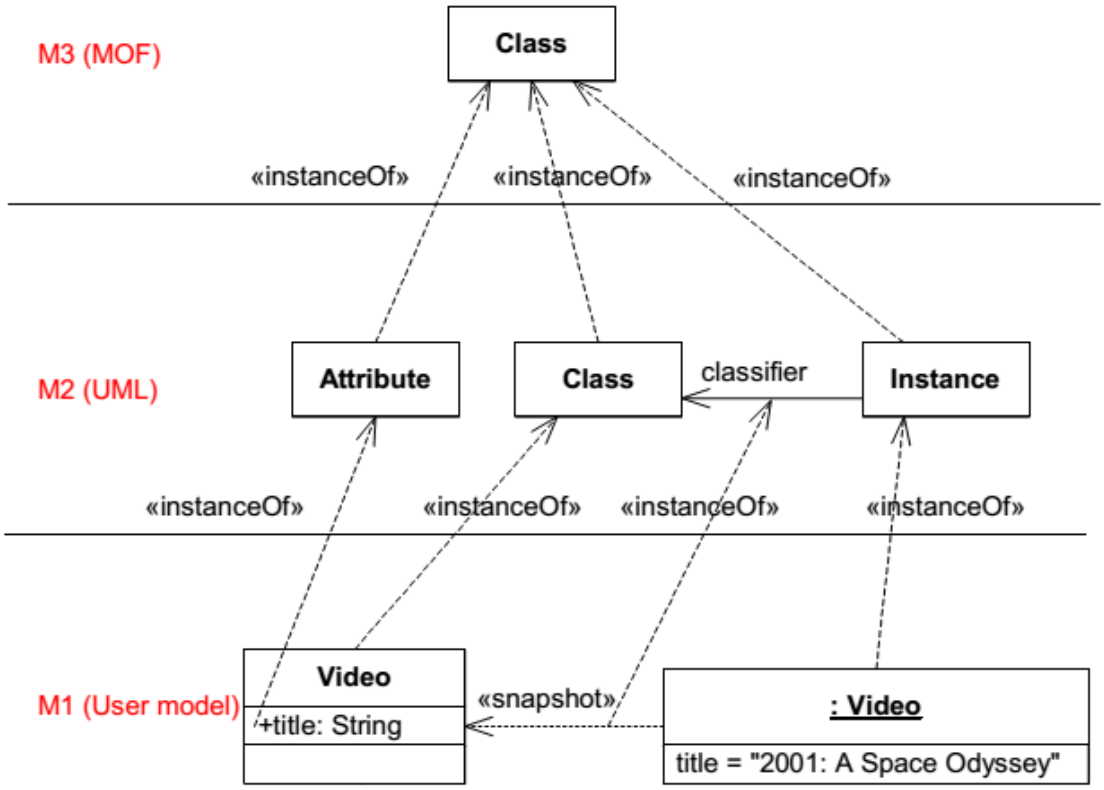

Figure 6. Example of Hierarchy between MOF and UML (from OMG 2011)

Figure 6. shows this relationship in practice. The different elements in the UML model are defined in terms of the base object in the MOF model, the Class. In turn, the user model is defined in terms of the UML model. The advantage of the MOF is that there is a formal way to extend UML, or other languages based on MOF, to meet specific needs. Software tools need only be able to interpret the MOF to allow reading the model without generating errors upon encountering on elements that were not previously known to the tool.

c. UML

UML is focused on representing the software engineering process without having to choose a specific methodology. This means that it does not specify how to perform the software engineering, but it does specify outputs and artifacts of the process and their

interrelationships. This includes not only the 13 UML diagrams in Figure 7. but also how different software elements are related, and how features such as software functions associated with a code module are expressed in terms of both syntax and semantics. This is critical in being able to have different tools operating with the same system representation.

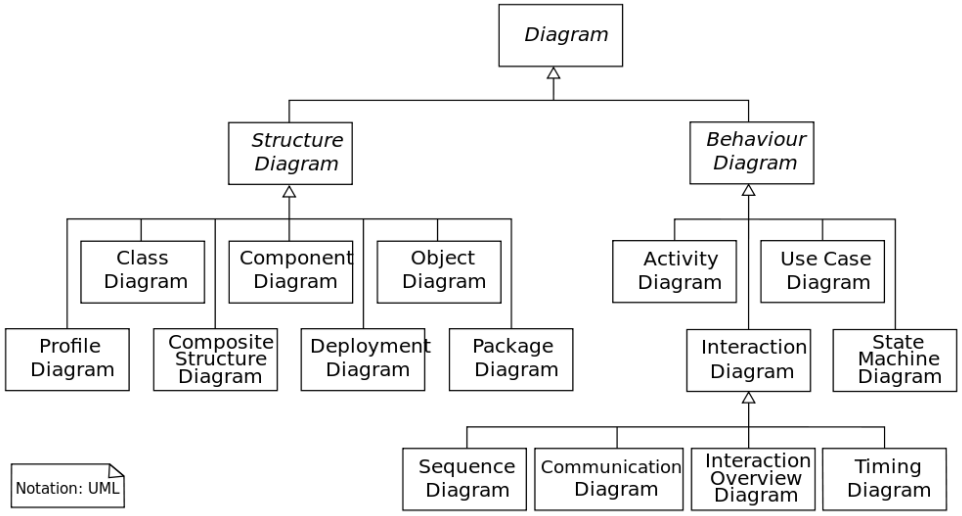

Figure 7. UML Diagrams (from Merson 2011)

A key limitation of UML from a systems engineering point of view is that there is no way to represent explicit requirements. UML is software focused and assumes that requirements will be defined implicitly by the behavior diagrams, particularly the Use Case Diagram (Weilkiens 2007). This particular inability hinders the use of UML in the context of systems engineering. While the MOF enables extensions to add new diagrams and relationships, having each vendor specifying their own extensions for such fundamental areas defeats much of the goals of a *unified* modeling language.

2. Systems Modeling Language

In 2001, INCOSE approached OMG about creating a standardized language for systems engineering based on UML (INCOSE 2003). OMG issued a request for proposal on creating "UML for Systems Engineering" in 2003. The work on creating a draft standard went through 2005, and SysML version 1.0 was not approved until 2007.

SysML had two underlying goals: The first was to be fully featured to support necessary systems engineering activities without prescribing a particular methodology. The second was to deliberately exclude unneeded UML features to simplify the language. The resulting diagrams in Figure 8. are the result. The interaction diagrams and the structural diagrams were modified, (green in Figure 8. and a new requirements diagram and parametric diagrams were added. Unlike UML, SysML supports explicit definition of requirements hierarchies as well as associating them with the system components and activities that enable the project to meet those requirements. This supports the traceability expectations in systems engineering that UML was not well structured to meet (Wielkiens 2007).

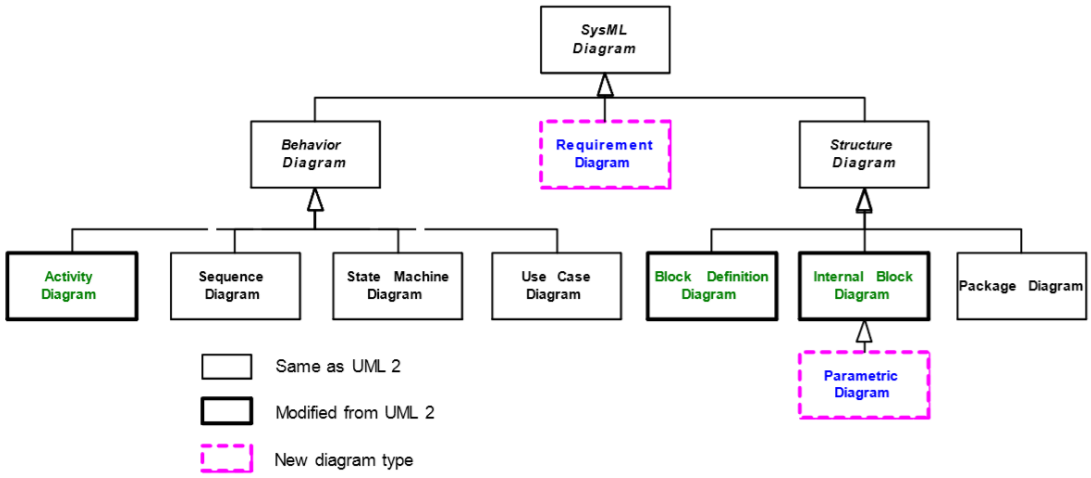

Figure 8. SysML Diagrams (from OMG 2012)

The Parametric Diagram provides a standardized construct to define quantitative relationships and then associate these relationships to particular attributes of system elements. This standardization provides the opportunity for non-customized software to interpret and analyze these relationships. What SysML does not do with the parametric diagrams is specify a particular language for how the mathematical relationships are specified, or even how to specify the language employed. This keeps with the OMG philosophy of not favoring a particular method, but can result in reduced intelligibility. For example, in Figure 4. the formula $flowrate = press / (4 * injectorDemand)$ could be

any one of a number of languages. For a simple case such as this, the difference is minimal, but for a complicated expression a formal syntax for specifying the language employed would be beneficial.

3. Vitech System Definition Language

Vitech CORE MBSE software uses their proprietary System Definition Language (SDL) for the system engineering model (Vitech 2013). Unlike UML and SysML, SDL is not based on the MOF. It is used to create a schema that defines potential relationships between different elements in the model. An example of an SDL schema is shown in Figure 9. Unlike SysML and UML, Vitech's SDL is tightly coupled to a particular MBSE tool, which provides some strengths and weaknesses.

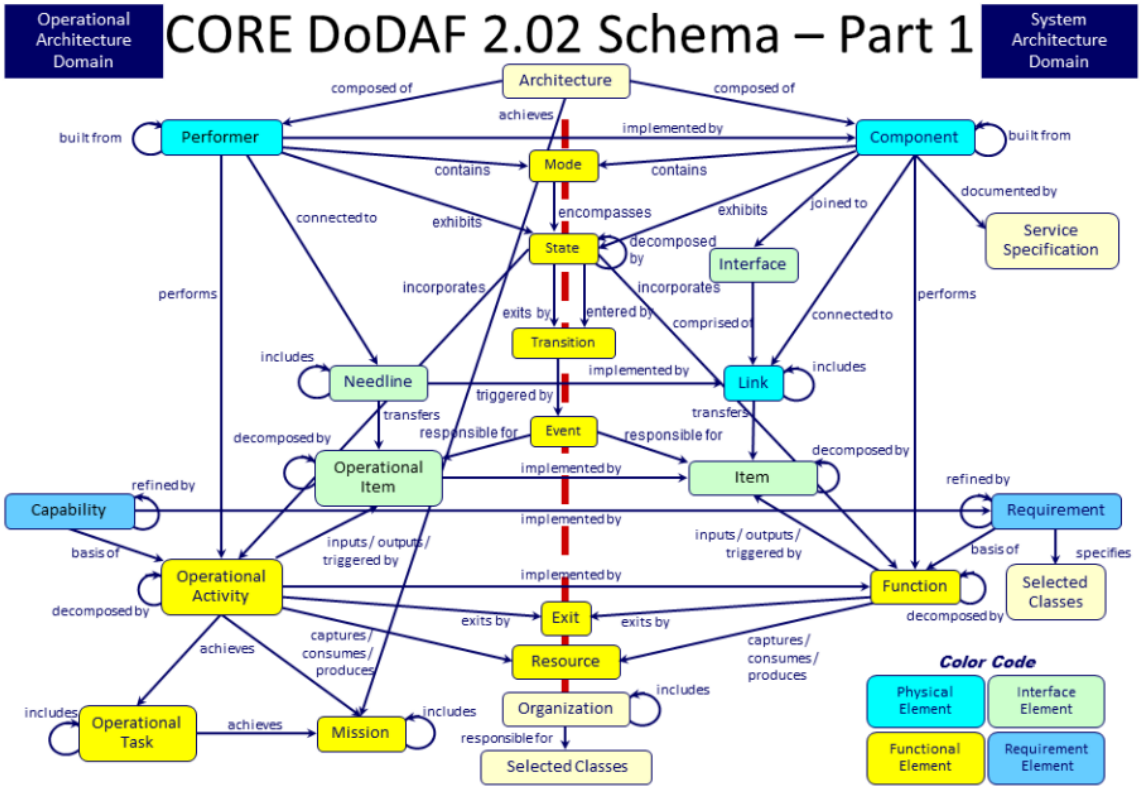

Figure 9. Sample SDL Schema (from Vitech 2013)

One consequence is that SDL hardcodes in attributes for system elements to support executable architectures in Vitech's COREsim. These are linked to particular

system elements in the schema, which imposes a restriction on what elements need to be defined within the schema in order to use the executable architecture feature of the tool.

SDL is used to support a wider variety of diagrams compared to those defined in SysML, but it is not clear whether this is a consequence of the CORE MBSE software tool or the language.

A drawback of SDL is that it does not expose a robust metamodel and is not XML metamodel interchange (XMI) compliant. This reduces the opportunity for interoperability compared to languages based on MOF. SDL is ISO 10303-233: Systems engineering data representation compliant. ISO 10303-233 is an SE data standard managed by the Standard for Exchange of Product data (STEP) subgroup of ISO, and defines a standard data format for certain types of SE information (Johnson 2006).

SDL does not support parametric modeling. Relationships between system elements exist at the level of the element. They cannot be bound to properties, which means that defining a new schema will not assist in this effort. This makes future extensions difficult without changing the structure of the language. In turn, third party tool support has to contend with the language structure is under Vitech control and not a neutral party.

III. PROCESS FOR QUANTITATIVE ANALYTICAL MODELING IN SYSTEMS ENGINEERING

Analysis is one of the fundamental tasks in systems engineering. Functional decompositions, behavior diagrams, and functional flow block diagrams are all analytical models. They provide a means to better understand how the system is expected to work. However, these methods are limited in that they do not provide assistance in essential questions for engineering a system, such as "how much" or "to what extent?" For example, consider designing some sort of search system for a military ship. Clearly, one of its primary functions could be "to detect." Without an understanding of how far it needs to detect, this provides minimal help in engineering. It is unlikely that the radar engineer feels that he has benefited from this analysis. However, if the systems engineering process also provided how far and a reference target, the radar engineer would have enough to start that part of the exploratory design process.

One of the common ways that quantitative modeling is brought into the systems engineering process is through trade-off analysis (Blanchard and Fabrycky 2011). Multiple concerns and objectives often lead to situations where improving performance for one objective is detrimental to another. An example is the classic tradeoff of cost, performance, and schedule. This nature of tradeoffs results in two parallel questions of "how much performance can I get for a reasonable cost" and "how much performance do I actually need," both of which are addressed by quantitative modeling.

A basic process for quantitative modeling in systems engineering is shown in Figure 10.

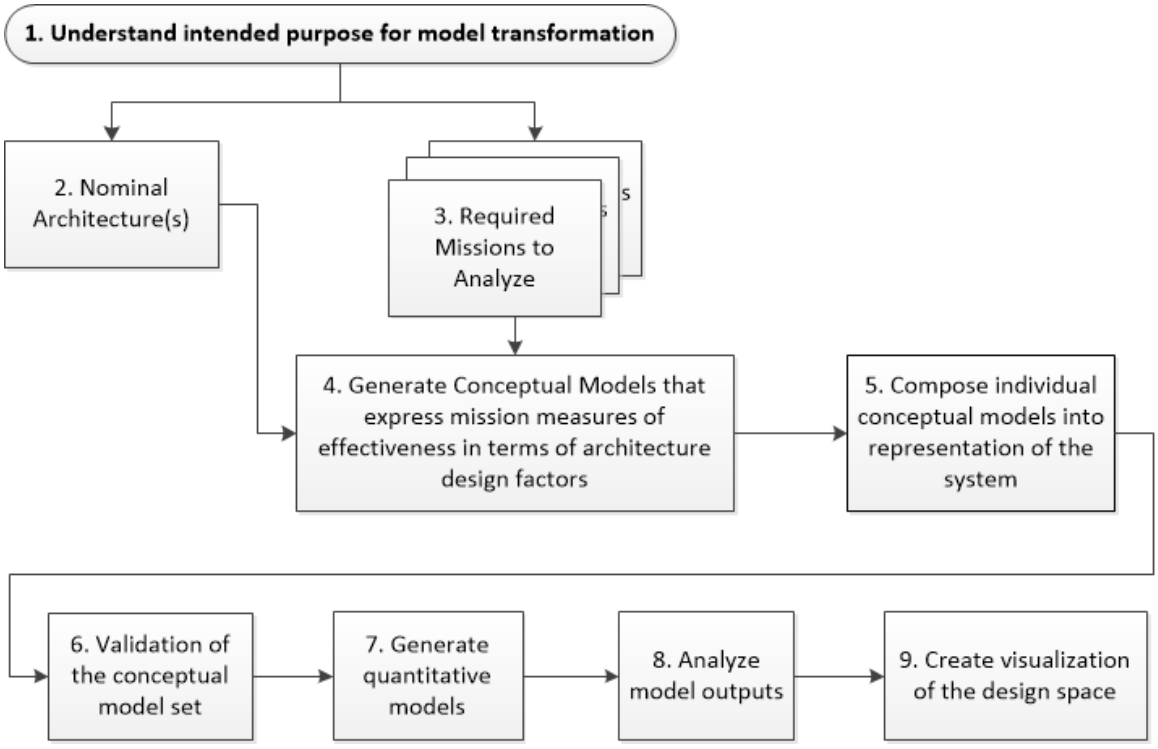

Figure 10. Quantitative Analysis Process Model

The first and most important step is to understand the purpose for the modeling effort. This can be thought of as what question the model is supposed to answer. Knowing the purpose keeps the modeling effort focused and assists in developing a model that is useful, rather than a model that is detailed for the sake of detail (Robinson 2012).

An architecture is required to build a model. It may be only a high level functional architecture meant for stochastic analysis, but it has to be available to make assumptions. Alternatively, it could be an architecture of the problem where the purpose is to better understand what an engineering solution needs to accomplish.

Block 3, required missions to analyze, is a breakdown of the distinct tasks or activities that need modeling in order to answer the question. This is used with the architectures to develop conceptual models. A conceptual model is shown in Figure 11. It provides an explicit view of what the quantitative model entails, from purpose to inputs, assumptions and outputs.

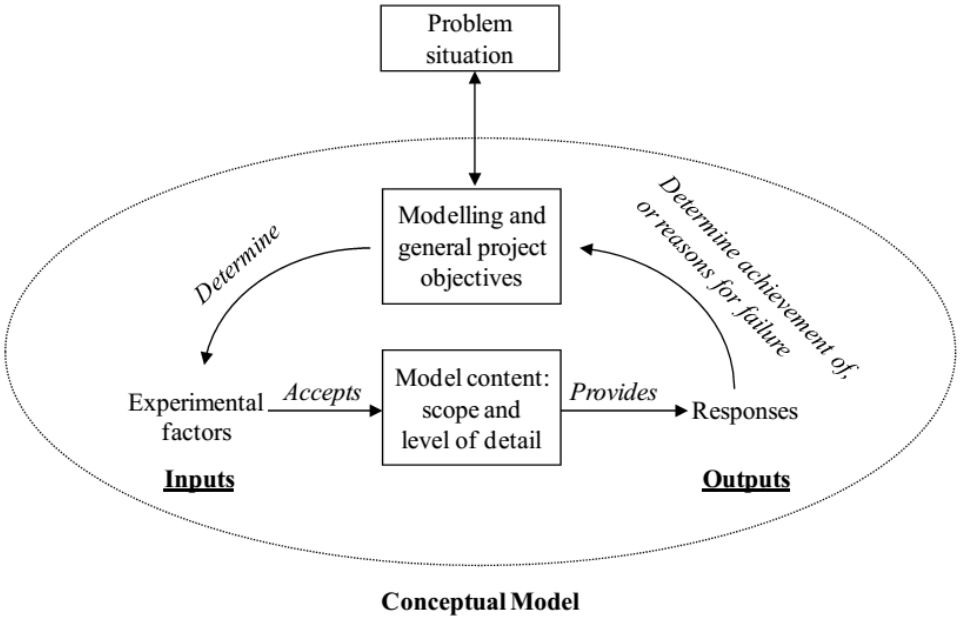

Figure 11. Breakdown of a Conceptual Model (from Robinson 2012)

The conceptual model provides an opportunity to confirm that the inputs and outputs align to the information available and to the information necessary to answer the question posed to the model. Many modeling efforts fail to be useful because they require information that cannot be obtained or produce results that cannot answer the particular question (John Hopkins University Applied Physics Laboratory 2010). Identifying these issues and resolving them before the programming and debugging efforts have even started is ideal.

Blocks 5 and 6 in Figure 10. deal with similar issues from a holistic view of the system. The major concern to be addressed is avoiding a composition fallacy. A composition fallacy results from not understanding the relationships between individual parts, and therefore assuming that if the parts perform satisfactorily, the whole must perform satisfactorily (Gula 2002). For example, if the system is an army tank, two missions or tasks might be to move and to shoot. If these are modeled separately, the results of the two models may not actually represent the desired performance of the tank. It may not be able to move and shoot simultaneously, or once it shoots, it can no longer

move. Since composition is an inferential rather than deductive process, the ability of software to recognize these mistakes is limited.

This is a point for engagement with subject matter experts. They can review the logic that the model is based on and concur it is a reasonable representation that is credible (Sargent 2011). Although experimental validation is the ideal means of validation, it may be impractical or the opportunity will occur so late in the system's lifecycle that its results will be less than useful. Expert review balances between practicality and utility. Accomplishing it with conceptual models, before programming occurs allows this review to focus on whether the model represents reality and not on "what does a particular module of code perform?"

The last elements of the process model are analysis and visualization. Visualization is distinguished because of the importance of communication. An example of why this is so important is the Challenger space shuttle catastrophe (Rogers 1986). Existing analysis supported that the launch conditions (specifically ambient temperature) should have resulted in an aborted launch but the communication of this concern failed. Similar possibilities for failure of decision-makers to understand an analysis are present on nearly every technical project. Neglecting how to display information exacerbates this concern.

IV. ANALYTICAL MODELING IN SYSTEMS ENGINEERING

Systems engineering encompasses both technical and managerial processes (INCOSE 2011). Consequently, viewing quantitative modeling from solely the viewpoint as a technical process does not capture the whole of its relationship with systems engineering. Managerial aspects including infrastructure management and communication between stakeholders are equally important. The ISO/IEC/IEEE standard 15288, Systems and software engineering—Systems Life Cycle Processes (ISO 15288) will be used as a reference standard. It is the SE process framework adopted by INCOSE for use with the *INCOSE Systems Engineering Handbook*.

A. SYSTEMS ENGINEERING PROCESSES

ISO 15288 divides system life cycle processes into the four categories shown in Figure 12. : Agreement Processes, Organizational Project-Enabling Processes, Project Processes, and Technical Processes. The processes in the leftmost column are focused on project management and have limited involvement in the systems engineering with the exception of infrastructure management, which is discussed in further detail in the next section. Similarly, the center column, project processes are primarily project management related and the only intersection with analytical modeling involvement is with configuration management, which includes the technical aspects of evaluating how changes to the system affects technical performance. The concerns arising from configuration management are similar to the concerns arising from requirements engineering and will not be addressed specifically to avoid redundancy.

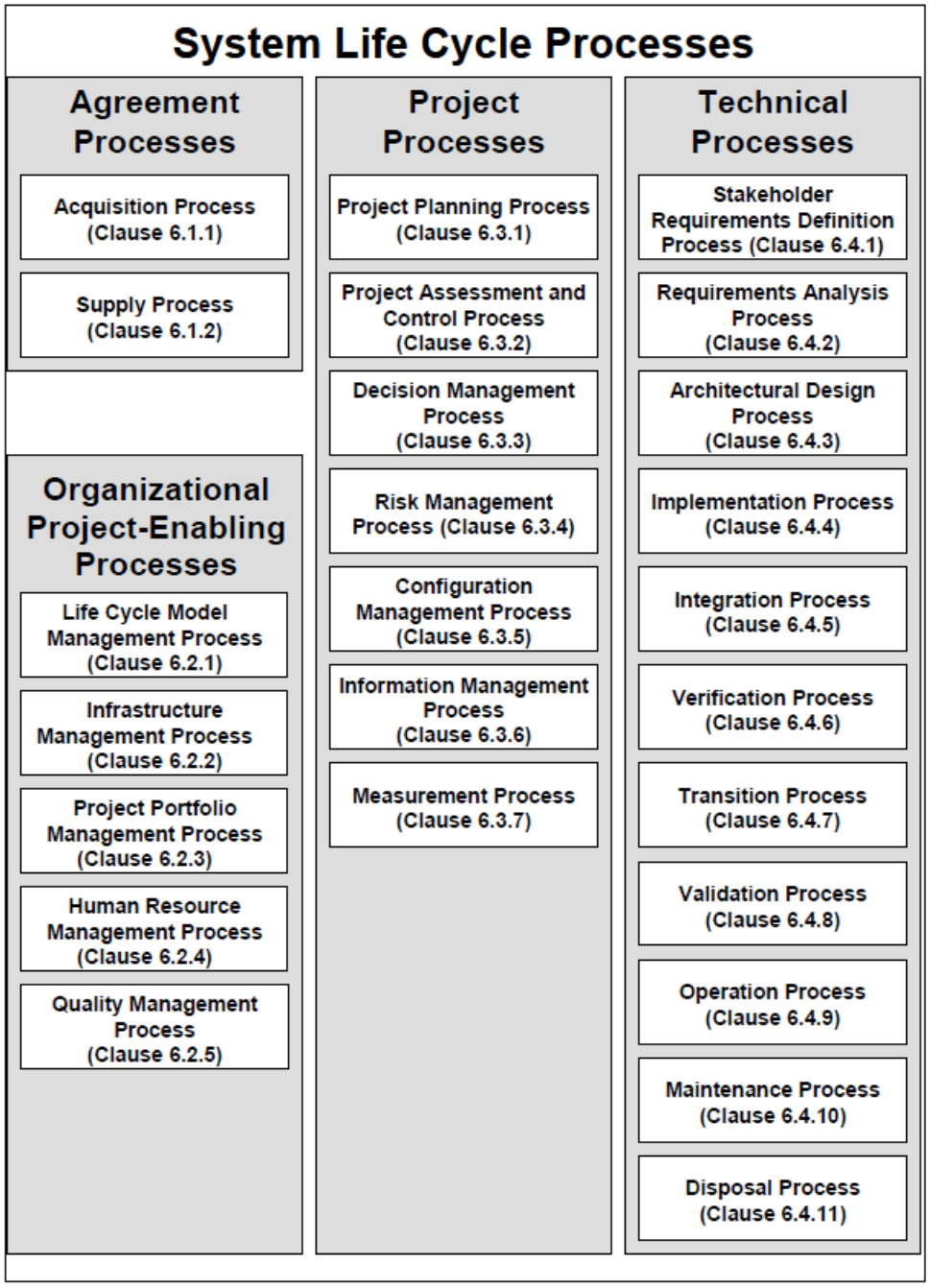

Figure 12. System Life Cycle Processes (from ISO 15288)

Technical processes are firmly related to systems engineering and have extensive involvement. Of the eleven processes identified in Figure 12. , there are two groupings that are particularly relevant for analytical modeling: requirements engineering, (which includes stakeholder requirements definition process, requirements analysis process, and

the architectural design process) and verification and validation. These two groups have distinct characteristics resulting from requirements engineering's focus towards informing the stakeholders so they can provide better inputs, while the verification and validation's focus is much more towards communication and understanding within the project.

B. INFRASTRUCTURE MANAGEMENT

Infrastructure management "provides the enabling infrastructure and services to projects to support organization and project objectives throughout the life cycle" (ISO 15288, 19). This is important because MBSE includes not only modeling methods but also the tools, and ultimately, the system model artifact itself. Given that without tools capable of manipulating a system model, there is no practical way to conduct MBSE. This is an important consideration. The primary concern is the availability (or even possibility) of different tools that are capable of manipulating the system model artifact. This concern exists in two dimensions: the breadth of different software tools and in the span of time a project may exist.

A breadth of tools is necessary because there is no possibility that all the possible analysis for a system will be capable of being conducted within a single tool or software package. For example, an MBSE effort towards building a CubeSat (a small satellite) integrated a variety of tools ranging from Microsoft Excel spreadsheets to Systems Tool Kit , in order to perform high fidelity analysis of orbital dynamics and the resulting effect on communication and power generation (Spangelo, Cultet and Anderson et al. 2013). Special, case-by-case support from the primary MBSE tool vendor should not be necessary to use specialized analysis within an MBSE environment.

This has two important implications. First, the system model should be available in a standardized format. Even if the primary MBSE tool has a proprietary internal representation, it should be able to generate a complete description of the system model in a standard format that does not rely on knowledge of a proprietary language to interpret. This allows both directly manipulating the system model with secondary tools and also performing more limited data exchange.

The second implication is that there is great value in adaptability of the system modeling language by the users, not only by the language developers. Important system attributes may not be known initially at the project start. The capability to add new properties to existing objects and to define new types of objects should exist in such a manner that the new relationships do not require specific support by MBSE tool vendor. This allows well-understood interaction between the system model and with other tools as well as retaining the capability to manage the new system attributes within the existing tool.

The issue with the span of time that the project life cycle encompasses is similar. A major system may exist for decades and require upgrades and support throughout its life. This duration has a major concern that the MBSE tool vendor may cease to exist before the SE project's life cycle is complete. A lessor concern is vendor lock, where a particular MBSE tool exists, is no longer preferable, but the cost of migrating to a competing tool is onerous. The potential duration strongly pushes towards an MBSE environment capable of writing the system model into a standardized language. This reduces the likelihood of MBSE tool obsolescence forcing a decision on whether to forego future tool improvements.

C. REQUIREMENTS ENGINEERING

> *Requirements Engineering is an interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software or service of interest. Requirements engineering is concerned with discovering, eliciting, developing, analyzing, determining verification methods, validating, communicating, documenting, and managing requirements*
>
> —ISO/IEC/IEEE 29148: Systems and software engineering Life Cycle processes—Requirements Engineering

This is one of the most important processes of systems engineering. Understanding stakeholder needs, one of the basic elements of requirements engineering, is difficult since they themselves may have a view that is incomplete or focused on one particular aspect. Without a good understanding of the stakeholder needs, there is little likelihood of being able to generate a high quality set of system requirements.

This overall process is both iterative and recursive. As shown in Figure 13. , when examining the system on one level, it is iterative. At first, the stakeholders know nothing beyond their initial thoughts. As more information is provided, their understanding of what they need improves, creating a set of requirements that will better solve their problems. This information flow in Figure 13. is shown as loops, but two-way dialogs may be more appropriate, since as the people in the process improve their understanding, they can ask better questions.

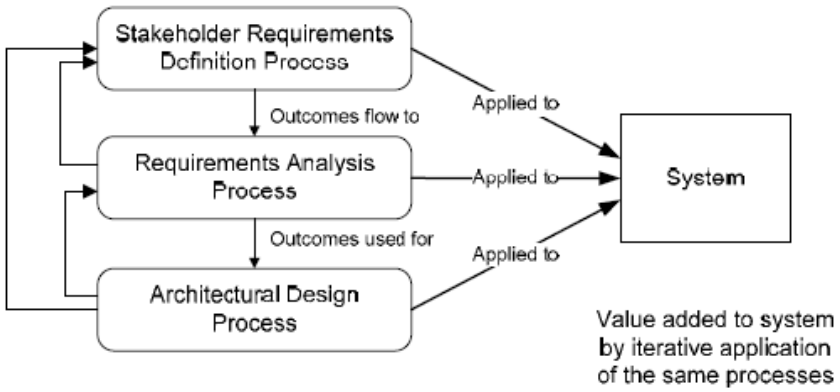

Figure 13. ISO/IEC/IEEE 29148 Iterative Process Model for Requirements Engineering

The recursive aspect is between different levels in the system's hierarchy of requirements. At the top level, the discussion is focused on definition of the system's requirements in terms of the problem's domain. As understanding of the required system is improved, progress on the requirements for key subsystems can be made. As the level of requirements moves away from the stakeholder's problem and towards the implied engineering requirements, the domain in which the requirements are described moves from the problem's domain into engineering domains. Understanding of the constraints and limitations of those key subsystems can lead to further refinement of the overall system requirements.

The outcome of requirements engineering is a "hierarchy of requirements that

- Enables an agreed understanding between stakeholders

- Is validated against real-world needs, can be implemented
- Provides a basis of verifying designs and accepting solutions" (ISO 29148, 8)

Current MBSE tools are highly capable of supporting the first point and the third point. Qualitative views of a system can illustrate the flow of logic involved in the assertion that the system will solve the stakeholder's problem. However, they provide limited specific information on how the system will meet the needs of the stakeholders.

Validation that the proposed solution can solve stakeholder's problem (within the limitations of a model) is a source of weakness that MBSE tool vendors attempt to handle via proprietary solutions. Many MBSE tool vendors such as Vitech, IBM, and NoMagic provide some level of support for executable architectures within their tool's environment. InterCax, another tool vendor, sells tools that assist in computation and manipulation of parametric diagrams within other SysML based MBSE tools.

1. Benefits of Integrating MBSE and Quantitative Modeling in Requirements Engineering

Quantitative modeling to support requirements engineering is not new. A common use is in tradeoff studies that seek to quantify the interrelationship between different factors. What MBSE provides is the opportunity to capture these tradeoffs within the system model instead of in stand-alone documents. This integration provides benefits along two avenues: First, this provides the opportunity to examine multiple tradeoffs simultaneously. This improves the ability of stakeholders to understand how different requirements are related, improving decision making. Second, it enables placing the rationale for technical decisions at a lower degree of abstraction within the system model.

a. Design by Shopping Paradigm

Richard Balling makes the point that the traditional analysis approach follows three steps: "1) formulate the design problem, 2) obtain/develop analysis models, and 3) execute an optimization algorithm." Since multiple criteria are common, some means of weighting the different criterion, such the analytical hierarchy process, is necessary to

create a score by which to rank alternatives (Ragsdale 2012, 737). Unfortunately, multi-criteria decision making processes are notorious for producing results that stakeholders do not prefer over manually generated designs (Balling 1999). A key problem with this traditional approach is that it requires stakeholders not only to know what they need, but also to understand the relative priorities of those needs.

An alternative approach is to take the set of needs and generate the tradespace of plausible designs. Allowing the stakeholder to examine these tradeoffs before prioritizing and weighting different requirements raises their knowledge of what requirements are naturally complementary, and which requirements are conflicting and require compromise (Balling 1999; Stump et al. 2002).

Balling's 1999 paper identified two areas of research needed to make this approach practical. First is the ability to create rich Pareto sets (the set of designs that are in some way superior to other possible designs). The second area is interactive means to display this information to decision-makers. Together these two elements enable decision-makers to see rational designs. Armed with this information, they can then set out an informed list of priorities and requirements to continue the systems engineering process.

The first area, creating rich Pareto sets, has had significant improvements by both new Design of Experiment (DOE) techniques such as Nearly Orthogonal Latin Hypercubes (NOLH) that have space filling properties as well as genetic algorithms that search for Pareto points (Cioppa and Lucas 2007, Balling 2006.) The second area, visualization, has also made progress. Tools such as JMP, MATLAB, and Maple are capable of displaying user manipulated contour plots to allow decision-makers to understand the potential solution space. For more complex situations where contour plots provide limited value (such as when the solution space is discontinuous or discrete), there has been research into visualizations of high-dimensional Pareto frontiers to allow decision-makers to explore the consequences of differing sets of priorities (Agrawal et al. 2004). Finally, software vendors such as InterCAX are bridging the gap between primary MBSE tools that maintain the system models and analysis software to allow integrating these efforts.

b. Requirements Analysis Process

The requirements analysis process is where the stakeholder requirements, which are largely in the problem domain, are translated "into a technical view of a required product that could deliver those services" (ISO 29148). An important element of this process is traceability, "the identification and documentation of derivation paths and allocation or flowdown paths of work products in the work product hierarchy" (ISO 24765). Traceability can benefit tremendously from the integration of quantitative modeling into the system model.

If requirements were fixed and not subject to change this would be less important. Traceability can be provided by a qualitative model to answer the question "What is the source of this particular performance requirement?" However, requirements do change, whether due to the stakeholders needs changing or because a planned technical specification is unattainable. The more important question becomes "Why was this particular value for the performance requirement selected, and how does changing it affect the overall system performance?"

By maintaining the analytical models with the MBSE model repository, both questions can be answered. This integration also enables the question "If I need to relax this technical specification, what other specifications can also be relaxed because the extra performance is superfluous?" An example of this is the interrelationships for a car between speed and range on one side and engine horsepower and fuel tank size on the other. If the engine horsepower becomes limited (say by the car's frame's maximum stress), a more efficient engine could be used, and correspondingly a smaller fuel tank is necessary to achieve the range requirement.

This results in a secondary benefit of analytical models in MBSE. By introducing relationships that provide more detail than "part of" or "satisfies," questions about rationales for performance specifications and exploration of the design space can be answered without necessarily returning to the original designers and stakeholders. This promotes more responsive engineering and better communication since everyone on the team can see the intent behind design choices.

V. CONSIDERATIONS ON MODELING LANGUAGES FOR SYSTEMS ENGINEERING

As mentioned in the introduction, a major objective for MBSE is to produce a coherent model of the system. A key element in being able to create a coherent model is through the use of a formal modeling language. A formal language is one that is constructed by a set of symbols and is constrained by a set of rules (Encyclopedia of Computer Science 2003). Having a known set of rules is crucial. Referring to Figure 2. in Chapter I, a strength of MBSE is having a set of tools support maintaining consistency in the SE process instead of requiring the systems engineer or other user to perform this function. The use of a formal language provides specific grammar and concept construction to allow software tools to achieve this purpose. This is in contrast to a natural language such as English, where rules of sentence structure and composition are not mandatory. Ambiguity is helpful for creative expression but is not at all desirable in engineering.

There are necessarily tradeoffs in constructing a language for supporting systems engineering. A language that is sufficiently complex to support every possible use is unlikely to be practical to use. On the other hand, a language that is narrowly confined to representing the system architecture may exclude the ability to quantify how different system elements interact and only show the qualitative aspects of which system elements interact in what order.

This chapter first examines what features would be desirable in a modeling language of systems engineering with a focus on those aspects particularly helpful for quantitative analysis. Second, several potential languages are examined on the basis of the features identified in the first part: UML, SysML, and Vitech Core.

A. DESIRABLE FEATURES IN A MODELING LANGUAGE FOR SYSTEMS ENGINEERING

1. Support for Multiple Tool Interoperability and Integration

The foremost feature for a modeling language for systems engineering is having a well-defined structure that is available to third-parties. Systems engineering is interdisciplinary, and by extension, quantitative analysis may draw from any branch of engineering as the project requires. Most engineering domains already have extensive software tools in order to assist in analysis and development. Existing tools should be able to interact with the system model instead of having a modeling language for systems engineering attempting to support the functionality of every possible tool.

Having a MBSE software tool expose an application programming interface (API) to mediate interaction with the system model is not sufficient. Tool vendors may exit the market, or may change the API and break the interaction. If the systems engineering project spans from system conception to disposal, the risk of having elements of the system engineering effort made unusable by software tool changes is significant. Even for projects with a limited timescale, there is the risk of outgrowing the primary MBSE tool, such as NoMagic's MagicDraw or Vitech's CORE, but being unable to change to a different one because the ability for secondary tools to interact with the system model is tied to the particular software tool rather than the modeling language.

A more specific language feature that would be desirable to support interoperability and integration of multiple tools is having specific rules or structures in the language for denoting the system engineering elements that are described in a different modeling language. This improves the semantics of black box constructs built such as in Chapter 2, Section A.2. A semantically meaningful construct would allow programmatically determining what data in the system model is primary and what is secondary data based on output from an external model. This improves traceability. Secondly, it enhances the ability of the MBSE tool to provide warning that changes may break an interaction with another tool. Consistent semantics and syntax for integrating external entities can make troubleshooting or reconfiguring external connections less reliant on any given individual's personal knowledge or experience. This will contribute

to users spending more time on performing engineering and less on understanding why a tool stopped working.

2. Flexibility

Another major important feature is flexibility, "the ease with which a system or component can be modified for us in applications or environments other than those for which it was specifically designed" (ISO/IEC/IEEE 24765, 144). For some projects it may be important to distinguish between the types of components used in the system, such as computers or hydraulic actuators. For others the system engineering is at a high enough level that this distinction is meaningless, but instead, cost estimates need to be allocated to individual components. Every project is different, and so is the important information.

The modeling language should be flexible enough to support assigning and tracking this information after the project starts and modeling the system has commenced. Furthermore, there needs to be rules on how additional information and new model entities will be presented in the modeling language. This permits the discovery of this new information by MBSE tools along with the ability to manipulate the new entities in a consistent manner.

Of less importance but still desirable is handling property values that are not fixed values. This can be due to the property in question (such as mass) being better represented by a random variable (i.e., the mass is uniformly distributed between 10 and 20 grams) or because the property is dynamic (i.e., the mass continuously decreases during operation as fuel is consumed). Quantitative analysis is very susceptible to "garbage-in, garbage-out," and restricting inputs to fixed values can create misunderstandings of system performance. Evaluating the performance of a system requires an understanding of the stochastic behavior of the processes in question (Sanchez 2000).

The modeling language specification should have normative guidance on how to represent these property values. This goes beyond merely being capable of representing random values. Normative guidance sets a standard, minimizing the assumptions and

proprietary behavior of modeling tools. This both reduces the likelihood of errors and supports interoperability and integration.

3. Composite Executable Architectures

Executable architectures have a strong tie to the software tools that implement their functionality. What is desirable but missing from the languages examined is language support for composited executable architectures. Activity diagrams follow a single world-line, or flow of events. Logical constructs such as “AND” or “OR” denote events that are expected to occur in parallel or the presence of a choice. It is easy to omit events that may occur when writing the potential activity flow due to the implications of the logic. A good example would be a pedestrian at a crosswalk as shown in Figure 14. : An activity diagram may show that if the crosswalk light is red, the pedestrian pushes the “press to walk” button, triggering a series of events that leads to the crosswalk light turning white, signaling the pedestrian to proceed. In reality, that pedestrian may press the “push to walk” button over and over, which goes untested and may result in undesirable system behavior such as the control circuit queuing many walk signals, delaying traffic, or failing altogether.

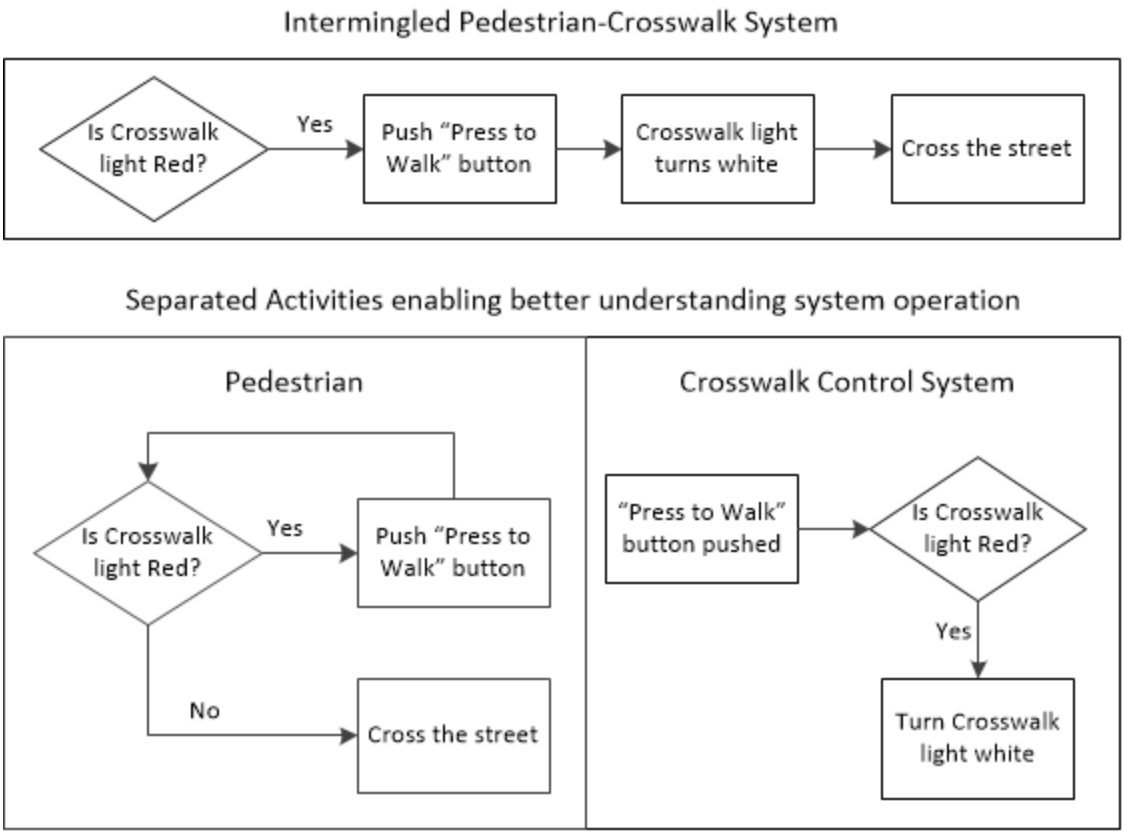

Figure 14. Separating Activity Sequences Better Describes System Operation

An issue with this is that the individual activities need to be composited back into a whole system in order to execute the event sequences. This can lead back to the intermingled example where real system behavior becomes lost. Two approaches to this problem are to represent the different activities as independent agents, and using a formal language to describe how events between activities are related and using an algorithm to build possible event sequences.

Simulation definition languages such as BOM and High Level Architecture (HLA) take the first approach and were designed to support discrete event simulation between multiple agents (SISO 2006, IEEE 2010). They are structured to decouple external events and internal events of federates (individual elements). The focus of those two languages is to support federated execution of simulation that is potentially distributed. Although not explicitly designed for systems engineering modeling and simulation, such an approach could work with MBSE. This separation reduces incorrect

simulation by not implicitly ruling out one agent such as the pedestrian repeating an event (i.e., button pushing) because a different agent, such as the crosswalk system, is processing an internal event. It requires either an external script to develop or additional programming within the agents to have the simulation express behaviors.

A separate approach is to formally denote events that are shared among different activities. An example is the Monterey Phoenix project, which uses a formal language to describe what events are shared (Auguston, Whitcomb, and Giammarco 2012). Explicitly labeling these interactions provides more insight into system behaviors by avoiding intermingling causal relationships between the activities sharing the same event. Another objective is to algorithmically generate a set of use cases that describe potential event sequences. This set of use cases can then be tested to verify that the system design does not fail during unanticipated sequences of events.

B. LANGUAGES

1. UML

a. Support for Interoperability and Integration

The UML standard is managed by the Object Management Group, an industry group. The UML specification is publicly available both from OMG and as international standards organization (ISO) specification 19505. Beyond this, the meta-modeling language used to define UML, MOF, is similarly available. A standardized file format, XMI, exists that all UML compliant tools are required to be able to output.

UML has limited support for quantitative analysis, and no formal way of specifying an external analysis. This traces back to its purpose in software design and an emphasis on managing program flow and information exchange between entities. Defining the semantics and syntax that an interface with an external analysis tool would use is possible, but it would be project and/or tool specific.

b. Flexibility

UML via the MOF has a high degree of flexibility through three related mechanisms available to users: stereotyping, extensions, and instancing (OMG 2012).

Stereotyping represents a specialization type of relationship, while extensions provide a means of specifying additional properties. Instancing handles different multiple elements of the same stereotype having different values assigned to their properties.

Stereotyping preserves the relationships available to the parent entity type. For example, if the base entity is a “class” object, the stereotype <<Input/Output>> of the entity type “class” is still a class. This provides two sets of benefits: first, it allows the MBSE tool to continue using the language rules for the “class” object to verify that <<Input/Output>> objects are used correctly. Secondly, it allows project specific rules and scripts to be created within the MBSE tool. For example, if the project is migrating to a different file system a developer can use the tool to locate all <<Input/Output>> objects to review if they need modifications to support the change.

Extensions address the need for different entities in the software project to represent different information. For a software class, each class provides different functions to the overall program, the number and type of which are not known at the start of the project. Having a standardized way to represent the distinct features for different stereotypes and/or instances makes tools simpler and more flexible.

Instancing reflects the interaction between stereotyping and extensions. A single stereotype (with several properties) may be used repeatedly within the project. Each location a distinct entity appears is an instance and can have its own set of values for the properties of the stereotype, in addition to any additional properties that need to be specified for that particular instance.

c. Non-fixed Value Properties

UML has neither rules nor normative guidance for properties without fixed values. The UML Datatype Diagram, shown in Figure 15. does support defining properties with a structured set of values. This allows the user to define a data type for a property with multiple primitive sub-properties. An example of how defining a random variable could be accomplished: *datatype: continuous random variable; sub-properties: distribution-type, mean, standard deviation.* The problem that this definition is ad-hoc,

may not cover all types of distributions, and may require a great deal of case-specific logic. Additionally, being user-defined, MBSE tool support for it would require further effort by the user to implement.

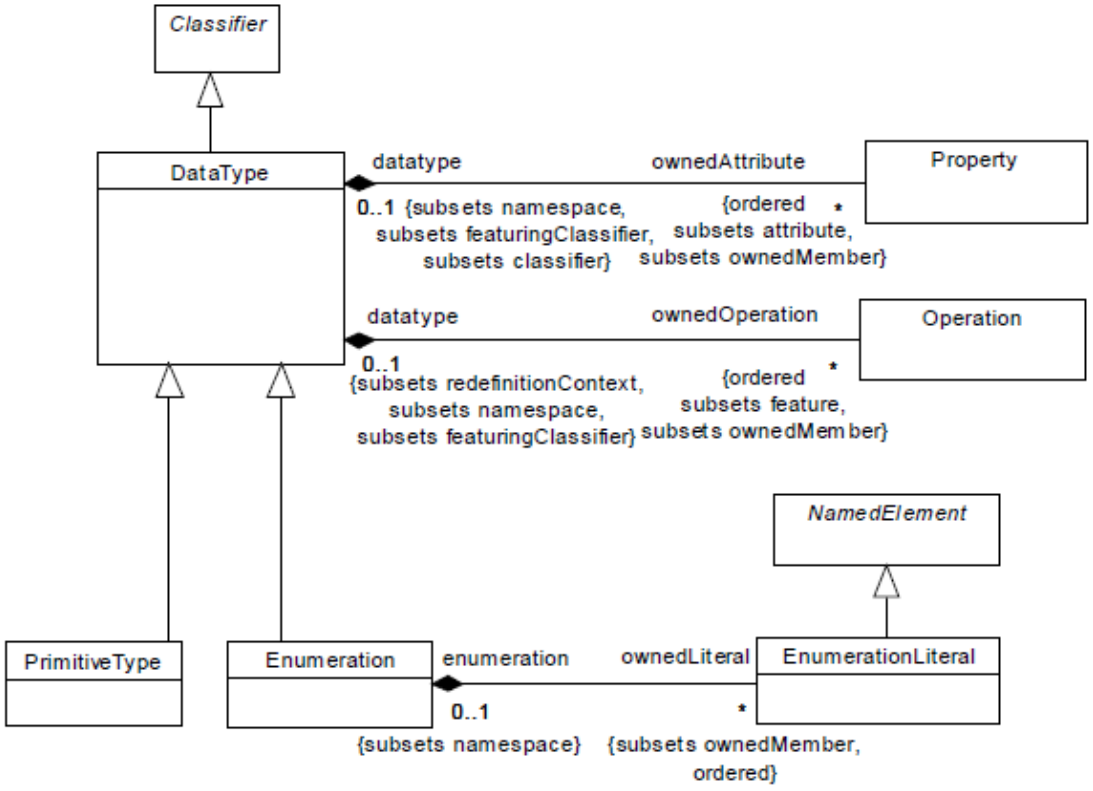

Figure 15. UML/SysML Datatype Diagram

UML has no support for parametric relationships, so that there is no support time dependence of properties is mooted.

2. SysML

a. Support for Interoperability and Integration

As covered in Chapter II, SysML is an outgrowth of UML, and it is also defined by the MOF. The complete SysML specification is available from OMG, and the process to standardize SysML as an ISO standard has commenced.

There is no formal way to represent a link to an external analysis. Through the use of parametric diagrams, a placeholder object can be created that shows a link between some system property and the placeholder. This provides a means to represent that the system property is derived, but showing that it is derived from an external source is left to the user to create a property on the placeholder object to retain information on the external analysis tool. This is less than ideal because it can change from project to project or even within a project among users.

There is an initiative by OMG to create a transformation specification between SysML and Modelica, a general purpose modeling language specialized towards system dynamics and that has an open language specification (Paredis, Bernard, and Burkhart 2010). On the SysML side, it creates a set of stereotypes to denote that the transformation engine will create the appropriate Modelica component for analysis.

This is a step forward and a step back. It provides a formal means for conducting analysis that cannot be directly represented in SysML. Furthermore, the analysis within Modelica has the potential to change as the system structure in SysML changes beyond manipulation of property values. This is a significant benefit by reducing duplication.

The step back is in two areas. First, this is only for Modelica. All other modeling languages and domain specific tools do not benefit from this work and remain in the state of ad-hoc notation for their use. Second, efforts in the modeling and simulation community on interoperability suggest that the current effort will have problems. Behavioral assumptions are particularly concerning (Gallant and Gaughan 2011.)

The Modelica transformation relies on creating the building blocks in Modelica, and specifying a particular syntax on the SysML side to allow the transformation tool to create the appropriate model on the Modelica side. The goal is to allow the Modelica model to update as the system model is updated in the MBSE tool, saving time and effort to be used on engineering. Encapsulated within the Modelica model are behavioral assumptions that are not visible from the SysML side. As the SysML model is constructed, those assumptions may be violated, and because there is no longer any need to write and verify the Modelica code, the output may no longer reflect the actual systems

performance. This may not be detected until a physical system is constructed. For the military application, this is particularly important because systems have human elements whose behaviors are situationally dependent. That is to say, human response may vary greatly with small changes to the situation or organizational structure and must be explicitly considered with changes from the original model.

b. Flexibility

SysML has the same strengths regarding flexibility as UML. Beyond this, for the purposes of quantitative analysis, it has a parametric diagram which includes "constraint" relationships. This relationship allows defining a link between properties of the same or different elements together.

c. Non-fixed Value Properties

SysML has limited progress compared to UML. In appendix D.6 of the SysML specification, there is non-normative guidance on modeling properties whose value is a distribution. Since this is not normative, there is no expectation that a user will follow it, and consequently there is less value for MBSE tool vendors to provide specific support for the approach recommended in the appendix. That it is considered at all is a sign that future revisions of the SysML specification may codify support in the future.

There is minimal support for time-dependent properties in the parametric diagrams. Constraints on property values may be written in terms of time-dependent differential equations, but in SysML parametric relationships do not specify directionality, or cause and effect. Without directionality, the ability to simulate the change in property values though time is limited because there neither side of equations is fixed for the other side to vary. The INCOSE space systems working group overcame this issue through a combination of external modeling tools and using a tool called PHX Modelcenter to support interoperability between different models and tools (Spangelo et al. 2013).

3. Vitech System Definition Language

a. Support for Interoperability and Integration

Vitech SDL creates a schema of the possible relationship between elements in the system model (Vitech 2013). The schema can be changed project to project, but the schema in use can be outputted through a report within the CORE tool. The file format used by CORE is not XMI compatible, but it is written in XML. Programmatically extracting information is possible with some effort even without a published format since XML is a structured text file and the schema is available. Because the XML structure does not follow a standard, there is some risk that future versions of CORE could change the structure, breaking the ability of other tools to manipulate the system model.

The system definition language does not have rules for external analysis. It also does not support parametric diagrams, preventing the use of black box elements to represent external analysis.

b. Non-fixed Value Properties

The System Definition Language allows some properties related to enhanced functional flow block diagrams (EFFBD), such as the time for an activity to complete, to be represented as random variables with a variety of distributions (Vitech 2013). Outside of these uses, there is no programmatic support for random variables. System elements can have user-defined properties and multiple properties could be used to represent a random distribution. This suffers the concerns related to ad-hoc formation discussed in more detail under UML.

There is some limited support for time-dependent properties, also in the context of EFFBDs. Resources can be managed, such as battery capacity, to represent the depletion over time. Outside of EEFBDs, there is no support for time-dependent properties.

THIS PAGE INTENTIONALLY LEFT BLANK

VI. CONCLUSIONS AND FUTURE RESEARCH

A. CONCLUSIONS

MBSE tools and languages are capable of performing quantitative modeling. Significant use of ad-hoc methods for routine situations such as handling random variables is required, which diminishes the formal modeling aspect of MBSE. Better-defined MBSE would improve the capacity for MBSE to assist systems engineers in designing better products by lowering the barrier for MBSE tools to enforce consistency across the system engineering process.

B. FUTURE RESEARCH

This thesis does not address how to eliminate the shortcomings identified in the systems modeling languages examined. The recommended future research is divided into two categories: parametric modeling and executable architectures.

1. Parametric Modeling

a. Semantics for Black Box Parametric Modeling

As discussed in Chapter 2, separate engineering analysis tools should be expected to exist for the foreseeable future. This implies that support for black boxes in parametric models, which may defined inputs and outputs but the details of the transformation are contained elsewhere, is desirable. The set of semantics to enable consistent use of black boxes for external analysis should be explored. Is it sufficient to merely identify a model entity as representing an external analysis, or is containing more information within the system model desirable?

b. Guidance on Modeling Random Variables

There is no normative guidance on treating property values as random variables. The concern is two-fold: First, how are explicitly defined random variables (such as one with an exponential distribution) handled? Second, how are random variables that are

implicitly defined by relationships to other random variables defined and manipulated? This is crucial to characterizing quantitative system behavior.

2. Executable Architectures

a. Recommendation on Way Forward in Modeling Event Timelines

MBSE languages and tools should support defining system behavior in such a way that the users do not need to detail every possible sequence of events. The SISO BOM and Monterey Phoenix are two similar but distinct approaches to this concern. How should MBSE proceed, and what language features are necessary?

LIST OF REFERENCES

Agrawal, G., K. Lewis, K. Chugh, C.-H. Huang, S. Parashar, and C.L. Bloebaum. "Intuitive Visualization of Pareto Frontier for Multi-Objective Optimization in n-Dimensional Performance Space." Paper presented at the 10th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, GA, September 2002.

Auguston, Mikahil, Clifford Whitcomb, and Kristin Giammarco. 2012. "A New Approach to System and Software Architecture Specifications Based on Behavior Models." http://hdl.handle.net/10945/14782.

Blanchard, Benjamin S., and Wolter J. Fabrycky. 2011. *Systems Engineering and Analysis.* 5th. Upper Saddle River, NY: Prentice Hall.

Czarnecki, Krzysztof, and Simon Helson. 2003. "Classification of Model Transformation Approaches." Paper presented at the the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture, Anaheim, CA, October .

INCOSE. 2008. *Survey of Model-Based Systems Engineering (MBSE) Methodologies Rev B,* by Jeff A. Estefan. INCOSE-TD-2007-003-02. http://www.incose.org/productspubs/pdf/techdata/mttc/mbse_methodology_survey_2008-0610_revb-jae2.pdf.

Encyclopedia of Computer Science. 2003. "Formal Languages." *Encyclopedia of Computer Science*. Hoboken: Wiley. http://search.credoreference.com/content/entry/encyccs/formal_languages/0.

Friedenthal, Sanford, Alan Moore, and Rick Steiner. 2008. *A Practical Guide to SysML.* Burlington, MA: Morgan Kaufmann Publishers.

Giammarco, Kristin, and Mikahil Auguston. "Well, You Didn't Say NOT to! A Formal Systems Engineering Approach to Teaching an Unruly Architecture Good Behavior." *Procedia Computer Science* 20, (2013): 277–282.

Gula, Robert J. 2002. *Nonsense: A Handbook of Logical Fallacies.* Mount Jackson, VA: Axios Press.

IEEE. 2010. *IEEE Std 1516.1-2010: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)—Federate Interface Specification.* New York: Institute of Electrical and Electronics Engineers.

ISO/IEC/IEEE. 2008. *ISO/IEC/IEEE Std 15288-2008: Systems and software engineering - System life cycle.* Piscataway, NY: Institute of Electrical and Electronics Engineers.

ISO/IEC/IEEE. 2010. *ISO/IEC/IEEE Std 24765-2010: Systems and software engineering - Vocabulary.* New York: Institute of Electrical and Electronic Engineers.

John Hopkins University Applied Physics Laboratory. 2010. *Best Practices for the Development of Models and Simulations*. Final Report NSAD-R-2010-037." Laurel, MD, John Hopkins University. Technical Report. http://www.msco.mil/documents/10-S-2_26_952%20-%20SIW10F%20-%20MS%20Development%20Best%20Practices%20Final%20Report%20-%20Diem%20-%2020100812%20-%20Dist%20A%20(3).pdf.

Johnson, Julian 2006. "How Does AP233 Support A Systems Engineering Process (e.g. ANSI / EIA-632)? An Update." Paper presented at the INCOSE UK Chapter Spring Conference, Swindon, UK, April 2006. http://www.incoseonline.org.uk/Documents/Events/SC06/SC2006_day2_slot7_johnson_julian.pdf.

Kerzhner, A. A., J. M. Jobe, and C. J. J. Paredis. 2011. "A Formal Framework for Capturing Knowledge to Transform Structural Models Into Analysis Models." *Journal of Simulation* 5: 202–216.

Merson, Paulo. 2011. "File:UML diagrams overview.svg." Last modified September 10. http://en.wikipedia.org/wiki/File:UML_diagrams_overview.svg.

Object Management Group. 2000. "Model Driven Architecture," by Robert Soley. Needham, MA. Technical Report. http://www.omg.org/mda/mda_files/model_driven_architecture.htm.

———. 2003. "INCOSE TECHNICAL BOARD LIAISON ADVISORY STATEMENTS: OMG/UML for Systems Engineering." Needham, MA. Technical Report. https://www.incose.org/practice/pdf/OMGLiaisonAdvisory_2003-0529.pdf.

———. 2011. "OMG Unified Modeling Language (OMG UML) Infrastructure Version 2.4.1." Needham, MA. Technical Report. http://www.omg.org/spec/UML/2.4.1/Infrastructure.

———. 2012. "OMG Systerms Modeling Language (OMG SysML) Version 1.3." Needham, MA. Technical Report. http://www.omg.org/spec/SysML/1.3.

———. 2013. "OMG Meta Object Facility (MOF) Core Specification version 2.4.1." Needham, MA. Technical Report. http://www.omg.org/spec/MOF/2.4.1.

Paredis, Chris, Yves Bernard, Roger M. Burkhart, Hans-Peter de Koning, Sanford Friedenthal, Peter Fritzson , Nicolas F. Rouquette and Wladimir Schama 2010. "An Overview of the SysML-Modelica Transformation Specification." Paper presented at the 2010 INCOSE International Symposium, Chicago, IL, July 2010. http://www.srl2.gatech.edu/btw/files/SysML-Modelica_overview_INCOSE2010.pdf.

Peak, Russel, and Dirk Zwemer. 2011. "Modeling & Simulation Interoperatibility (MSI) Challenge Team INCOSE MBSE Initiative." Paper presented at the INCOSE Internation Workshop 2011, Phoenix, AZ, January 2011. http://eislab.gatech.edu/pubs/seminars-etc/2011-01-incose-iw-mbse-msi-peak/draft/2011-01-incose-iw-mbse-challenge-msi-peak_v1.0_as_presented.ppt.

Robinson, Stewart. "Tutorial: Choosing What to Model - Conceptual Modeling for Simulation." Paper presented at the 2012 Winter Simulation Conference, Berlin, Germany, December 2012.

Rogers, William P. 1986. *Presidential Commission on the Space Shuttle Challenger Accident.* Washington DC: Government Publishing Office.

Sanchez, Susan. "Robust Design: Seeking the Best of All Possible Worlds." Paper presented at the 2000 Winter Simulation Conference, Orlando, FL, December 2000.

Sargent, Robert G. "Verification and Validation of Simulation Models." Paper presented at the the 2011 Winter Simulation Conference, Phoenix, AZ, December 2011.

Simulation Interoperability Standards Organization. 2006. "Base Object Model (BOM) Template Specification." Orlando, FL. Technical Report. http://www.sisostds.org/ProductsPublications/Standards/SISOStandards.aspx.

Stump, Gary, Timothy W. Stimpson, Mike Yukish, and Lori Bennett. "Multidimensional Visualization and its Application to a Design by Shopping Paradigm." Paper presented at the 9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Atlanta, GA, September 2002.

Vitech Corporation. 2013. "CORE 9 Architecture Definition Guide." Blacksburg, VA. Technical Report. http://www.vitechcorp.com/support/documentation/core/900/ArchitectureDefinitionGuideDoDAFv202-9a.pdf.

Wielkiens, Tim. 2007. *Systems Engineering with SysML/UML*. Burlington, MA: Morgan Kaufmann Publishers.

Zockoll, Guido, Axel Scheithauer, and Marcel Douwe Dekker. "File:OO Modeling language history.jpg." Last modified December 7, 2012. http://en.wikipedia.org/wiki/File:OO_Modeling_languages_history.jpg.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California