Mathematical Framework for Early System Design Validation Using Multidisciplinary System Models

Bradley Jared Larson
Brigham Young University - Provo

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Mechanical Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/3000

This Dissertation is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
ABSTRACT

Mathematical Framework for Early System Design Validation Using Multidisciplinary System Models

Brad J. Larson
Department of Mechanical Engineering, BYU
Doctor of Philosophy

A significant challenge in the design of multidisciplinary systems (e.g., airplanes, robots, cell phones) is to predict the effects of design decisions at the time these decisions are being made early in the design process. These predictions are used to choose among design options and to validate design decisions. System behavioral models, which predict a system’s response to stimulus, provide an analytical method for evaluating a system’s behavior. Because multidisciplinary systems contain many different types of components that have diverse interactions, system behavioral models are difficult to develop early in system design and are challenging to maintain as designs are refined. This research develops methods to create, verify, and maintain multidisciplinary system models developed from models that are already part of system design. First, this research introduces a system model formulation that enables virtually any existing engineering model to become part of a large, trusted population of component models from which system behavioral models can be developed. Second, it creates a new algorithm to efficiently quantify the feasible domain over which the system model can be used. Finally, it quantifies system model accuracy early in system design before system measurements are available so that system models can be used to validate system design decisions. The results of this research are enabling system designers to evaluate the effects of design decisions early in system design, improving the predictability of the system design process, and enabling exploration of system designs that differ greatly from existing solutions.

Keywords: system design, multidisciplinary design optimization, nonlinear systems, model composition
ACKNOWLEDGMENTS

I am grateful for my wife and children for letting their husband and father pursue a dream.
# TABLE OF CONTENTS

List of Tables .............................................................................................................. iv

List of Figures ............................................................................................................... v

NOMENCLATURE ......................................................................................................... vi

Chapter 1  Introduction ................................................................................................. 1
  1.1 Current System Modeling Research ......................................................................... 4
  1.2 Scope and Objectives .............................................................................................. 6

Chapter 2  Mathematical Formulation of System Models using Existing Engineering Models ....................................................................................................... 8
  2.1 System Modeling Requirements ............................................................................... 8
    2.1.1 System Model Requirements .............................................................................. 9
    2.1.2 Component Model Requirements ...................................................................... 9
    2.1.3 Requirements Summary ................................................................................... 12
    2.1.4 Needed Developments ...................................................................................... 12
  2.2 System Model Mathematical Formulation ................................................................ 13
    2.2.1 Component Model Definition .......................................................................... 14
    2.2.2 System Model Definition .................................................................................. 24
    2.2.3 Solution Requirements ..................................................................................... 28
    2.2.4 Modeling in System Design .............................................................................. 30
  2.3 Solar Powered UAV System Model .......................................................................... 31
    2.3.1 UAV Component Models ................................................................................... 31
    2.3.2 UAV System Model .......................................................................................... 36
  2.4 Summary ................................................................................................................ 42

Chapter 3  System Model Feasibility ........................................................................... 43
  3.1 System Model Evaluation Failures ........................................................................... 44
  3.2 Feasible Domain Exploration ................................................................................... 46
    3.2.1 Defining a Searchable Space ............................................................................. 47
  3.3 UAV System Model ................................................................................................ 48
  3.4 Model Feasibility Design Space Exploration ............................................................ 51
    3.4.1 Search Method Evaluation .............................................................................. 56
    3.4.2 Feasible Solution Set Boundary Classification ............................................... 58
    3.4.3 Increasing Feasibility using System Model Feasibility Exploration ................. 59
  3.5 Summary ................................................................................................................ 60

Chapter 4  System Model Accuracy ............................................................................. 61
  4.1 System Model Accuracy .......................................................................................... 63
    4.1.1 System Model Error Sources .......................................................................... 63
  4.2 Quantifying System Model Accuracy ......................................................................... 64
LIST OF TABLES

2.1 Component Model Evaluation Sequence ........................................ 16
2.2 Differential Algebraic Equations (DAE) Evaluation Sequence .................. 18
2.3 Discrete event System Specification (DEVS) Evaluation Sequence ............... 20
2.4 Turing Machine Evaluation Sequence ........................................... 22
3.1 Component and System Model Failures ........................................... 46
### LIST OF FIGURES

2.1 Example of a System Model Directed Graph $G$ ........................................... 25
2.2 Solar Powered Unmanned Aerial Vehicle (UAV) Propulsion System Graph .... 32
2.3 Solar Powered UAV Measured vs. Computed Voltage, Current, and Thrust .... 40

3.1 Graph of a Simple Solar Powered Unmanned Aerial Vehicle Propulsion System . 49
3.2 Overview of Proposed Design Space Exploration Algorithm ............................. 52
3.3 Feasibility Exploration of a UAV System Model ........................................... 57
3.4 Comparison of Random, Mean-Squared Error, and the Proposed Feasibility Design Space Exploration ................................................................. 58

4.1 System Model $S$, Physical System Behavior $S_u$, Variational System Model $S_v$, and Error Bounds $S \pm \varepsilon$ Response to Input $u$ ......................................................... 65
4.2 Comparison of the (a) System Model $S$ and (b) Variational System Model $S_v$ Graph Structure .......................................................... 68
4.3 Variational Model Vector Ranges Satisfying Equation 4.28 ............................. 74
4.4 Solar Cells Model with Max Error Window Overlaid on Experimental Data .... 78
4.5 Measured Data vs. Battery Model of Battery Voltage During Discharge ....... 79
4.6 Measured vs. Modeled Motor Speed and Current Over a Step in Supply Voltage ... 81
4.7 Propeller Thrust Model with Max Error Window Overlaid on Experimental Data .... 82
4.8 Max Error Results Graph Showing Measured Data and Model Prediction with Worst-Case Error Bounds ......................................................... 83
NOMENCLATURE

The following represent key variables used throughout this dissertation.

\[ S \] System model
\[ S_v \] Variational system model
\[ U \] Domain or set of all possible inputs to the system model
\[ u \] Sequence of system model inputs
\[ Y \] Range or set of all possible outputs from the system model
\[ y \] Sequence of system model outputs
\[ C_j \] \( j \)th component model within a system model
\[ D_j \] \( j \)th discipline-specific model
\[ f_j(x_j, u_j) \] Progression function of the \( j \)th component model
\[ g_j(x_j, u_j) \] Output function of the \( j \)th component model
\[ F(S(u)) \] Feasibility function
\[ \varphi \] Input configuration parameters
\[ I(\varphi) \] Input function
\[ F_\varphi \] Feasible solution set
\[ R \] System model feasibility ratio
\[ \tilde{F} \] Feasibility prediction matrix
\[ \Phi \] Feasibility error matrix
\[ P \] Resample priority matrix
\[ \varepsilon \] System model error vector
CHAPTER 1.  INTRODUCTION

System design is the process of creating and integrating various components (which are frequently heterogeneous and dynamic) to satisfy system objectives [1]. To be effective and efficient at system design, a design team must predict with sufficient accuracy the effects of design decisions on system behavior at the time these decisions are being made. This is usually well before the physical system is available to verify these design decisions. Additional difficulties arise because multidisciplinary systems contain many different types of components that have diverse interactions. System behavioral models, which provide an analytical method of predicting a system’s behavior, are often not developed as part of system design [2]. Even when developed, system behavioral models are rarely available early enough in system design to guide foundational design decisions [3]. This is because they are difficult to create and are challenging to maintain as designs are refined. Rather, component-level models, designer expertise, and iterative physical prototyping are frequently used to validate system design decisions [4]. Consequently, the length of time between making system design decisions and verifying the results of these decisions can be long and result in an unpredictable system design process [5].

The separation between design decisions and design verification that is currently part of system design adds significant risk to the system design process [6]. Frequently, a system design that does not achieve its requirements continues to be developed, without discovering its limitations, until the full system integration and evaluation is performed at the end of system design [7]. At this point, a major system redesign could be necessary to meet system objectives. This was illustrated in the F-35 (Joint Strike Fighter) development, which is one of the leading system design projects to date. In 2003, two years after the prototype X-35 had flown (the F-35 experimental prototype), design reviews discovered that the F-35’s design weight compromised performance requirements. Removing the extra weight resulted in an 18 month production delay.
and added significantly to its design costs [5]. The separation between design decisions and design verification, even in a premier system design process, enabled this error to persist undetected.

This dissertation develops and illustrates a mathematical framework enabling system design decisions to be validated early in system design. It accomplishes this by addressing 3 fundamental obstacles of using compositional system models early in system design. The first obstacle is developing a population of component models from which system models can be built. Therefore, we create and verify a mathematical formulation for creating system models using existing engineering models that are currently available within system design [8]. The second obstacle is ensuring that these system models can continue to compute feasible results in the continually changing system design environment [9]. We introduce a new theory and method of identifying the system model feasible domain. Using this, we develop several approaches to evaluate and improve the feasible domain, thus ensuring reliable system model evaluation even as underlying component and system models change during system design [10]. Finally, to validate system design decisions, we must be able to quantify system model accuracy before the actual system behavior is known [11]. Current methods rely on system measurements, empirical evaluation, or differential error estimation and are insufficient to quantify the accuracy of system models early in system design [12]. To overcome this obstacle, we develop a theory to bound system model accuracy based on known component model errors and verify this theory analytically and numerically [13]. Together, this dissertation introduces a mathematical framework enabling multidisciplinary system design decisions to be validated at the point they are made. This enables the system design process to reliably create new systems that achieve their design objectives. It also enables new optimal system designs to differ greatly from previous designs, a practice that is currently avoided because of the unpredictable system design process used today.

We will refer to three levels of models: system models, component models, and discipline specific models. A discipline specific model is a model that describes a portion of a system using a modeling tool designed for that purpose (i.e., domain specific modeling language). Some examples of discipline specific models are electrical models in PSpice, CAD models in Pro/ENGINEER®, control system models in Simulink®, and data analysis in Excel. A discipline specific model becomes a component model when it is able to be combined by composition with other models. Model composition is the process of combining component models by evaluating one model, pass-
ing its results to another component model, and then evaluating this second component model [14]. Model composition of \( y_1 = C_1(u_1) \) and \( y_2 = C_2(u_2) \) can be written \( y_2 = C_2(C_1(u_1)) \), where component model \( C_1 \) is evaluated first, its output \( y_1 \) is transferred to \( u_2 \), and then \( C_2 \) is evaluated [15]. In this dissertation, a **system model** is a model that combines component models by composition to describe a system’s behavior [16]. Additional levels such as sub-system and sub-component can be achieved by making a system model a component model within a higher level system model.

Though simple in concept, the complexities of modeling multidisciplinary systems are significant. Developing system models from existing discipline specific models places additional demands at both the component modeling and system modeling levels. For example, all component models must conform to a common control and data interface so they can be used by a system model. This is complicated by the fact that discipline specific models are created and maintained independently to model different processes. Additionally, computational requirements imposed by the system model result in the redevelopment and reverification of non-conforming discipline specific models. This redesign eliminates many of the benefits of reusing existing discipline specific models. Along with these challenges, the resulting system models must also reliably compute results over the desired simulation domain (set of inputs into the system model) and must model system behavior with sufficient accuracy to validate system design decisions.

Despite these complexities, creating system models from existing discipline specific models provides several important benefits: (1) existing discipline specific models become a large population of trusted component models (i.e., models with some level of validation, experience with their valid domain, and acceptable accuracy) from which system models can be quickly developed; (2) during system design, these component models can be combined to provide immediate and ongoing analysis of the impact of design decisions on system behavior; and (3) system models are able to directly benefit from the increasing power and capabilities of discipline specific modeling tools.

The benefit of using system models to validate design decisions has been demonstrated by digital logic’s rapid increase in capabilities and performance [17]. Digital logic (e.g., a computer’s microprocessor) is created from a few types of well-defined components [18]. Based on models of these components, digital logic design automation tools enable system design and modeling of
extremely complex digital logic circuits. This has resulted in the ability to reliably design new, more capable systems.

Unfortunately, these benefits cannot be directly attained by multidisciplinary system design. In contrast to the few types of components in digital logic design, multidisciplinary systems contain many different types of components that interact in many different ways. This diversity has impeded advancements similar to those in digital logic for multidisciplinary systems. The many different types of components and their varied interactions complicate creating component models, describing component interactions, and developing system models [19]. As such, multidisciplinary systems require a more general method of creating and managing components and their interactions than is needed in digital logic design.

From our study, the primary impediment in multidisciplinary system design is the designer’s inability to accurately predict system behavior early in system design due to the diversity of component behaviors and component interactions. This difficulty has increased as the digital revolution, scientific advances, and demands for higher performance have increased system complexity. Simultaneously, the increased power of discipline specific modeling tools and fabrication techniques has increased component complexity [20]. Our ability to predict and design multidisciplinary systems has not increased correspondingly as has occurred in digital logic design [2]. This has resulted in increased design time, cost uncertainty, and design failures when developing new multidisciplinary systems [5]. The objective of this research is to enable designers to predict system behavior early and throughout the system design process. This enables quantitative analysis of different design concepts, understanding of design trade-offs, the ability to explore emergent behavior within a system, and ongoing verification of a design’s implementation as the system design progresses.

1.1 Current System Modeling Research

This section evaluates existing system modeling research and methods. It identifies model composition as a method that enables different types of discipline specific models to be combined to predict system behavior. Research applicable to specific topics such as model development, feasibility and accuracy is addressed in their respective chapters.
A number of leading multidisciplinary system design organizations have recently published initiatives aimed at improving the design of new systems in light of their growing complexity [21]. Initiatives have been announced by DARPA [22], NASA [23], Lockheed Martin [24], and Boeing [25]. A unifying characteristic of these initiatives is that each seeks to design and model new systems based on reusable components (e.g., mechanics, propulsion, electronics). Some of their motivations include improving the reliability and speed of system design, enabling various concepts to be compared early in system design, and ongoing verification during a design’s implementation [6].

Currently, a very common method of determining a new system’s behavior is by developing and testing physical prototypes [4,26]. When verifying system design decisions using physical prototypes, a new system design is initially based on studies of previous systems, analysis of component models, and designer expertise. If this new system is unable to accomplish its design objectives, the prototype system is studied to determine the causes of the shortfall and the system design cycle restarts [27]. This results in significant development risk because of the many design decisions that can only be verified much later when system prototypes are available [24]. Actual system behavior can diverge unchecked from the desired system behavior between system prototypes, causing numerous design iterations and long development cycles [28]. Iterative prototyping also limits system design exploration due to the effort needed to develop and evaluate physical system prototypes.

New system behavior can also be predicted using system models. Compositional system modeling assumes that heterogeneous, discipline specific models can be combined into system models [29]. Rather than translating component models into a common modeling language, model composition enables heterogeneous component models to interact within a system model through model composition [15, 30]. Compositional modeling has been implemented in commercial design tools such as iSight and Simulink [31, 32]. It is used frequently in multidisciplinary design optimization research [33–35].

Because compositional system models are able to integrate component models implemented in different design tools, they are able to model systems composed of diverse components without losing accuracy due to model conversion [21,36]. Model composition can also be achieved without requiring elusive unified modeling standards [37,38].
Due to the long-term as well as renewed interest in compositional system modeling, many aspects of this approach have been well established. Some of these include: (1) system behavior can be segmented and modeled using lower level component models [39–42], (2) model composition allows models of various types and time domains to be coupled [16, 43, 44], (3) component models implemented in various design tools can be coupled into compositional models [33, 35, 45], (4) various methods of classifying and sharing the diverse data transferred between component models have been developed [38, 46], and (5) dynamic analysis of compositional system models can predict system behavior [47].

Despite this ongoing interest in compositional system modeling, multidisciplinary system design frequently does not benefit from this method [3]. Several obstacles hinder its use to validate design decisions. One obstacle is that compositional system modeling presupposes an available population of component models from which to develop system models. This population usually does not exist. Without this, a compositional system model demands more effort to develop than a stand-alone system model. Secondly, because of the dynamic nature of system design, system models must be adapted frequently to continue to represent the design. In some industries, this results in a lag between the design and analysis. In others, it prevents system modeling completely. Finally, without validation, a system model can be used to explore general trends in system behavior, but not to validate design decisions. System model validation typically relies on system measurements that are not available until the end of the system design process. To use system models to validate design decisions, methods to quantify system model accuracy that do not require system measurements are needed.

1.2 Scope and Objectives

The objective of this research is to enable system designers to predict system behavior early and throughout the system design process. This enables design verification early in system design when designers have the most freedom to select alternative design concepts and design implementations. Later in system design, it allows design trade-offs to be evaluated and various implementations to be compared. It also enables ongoing design implementations to be monitored and evaluated for their impact on system objectives.
To accomplish these objectives, this research:

1. develops a mathematical formulation for creating system models from existing discipline specific models,

2. verifies that this formulation is sufficiently broad to cover most engineering models,

3. identifies solution requirements for these system models,

4. verifies that this formulation is able to correctly model engineering systems,

5. identifies primary failure modes for this system model formulation,

6. defines feasibility as a measure of these failure modes,

7. establishes a general method of defining a searchable domain for system models,

8. creates and illustrates a design exploration algorithm able to quantify an arbitrarily complex feasible domain of a system model,

9. identifies methods to improve the feasible domain of a system model using this algorithm,

10. establishes a mathematical foundation to quantify system model accuracy early in system design before system measurements are available, and

11. uses system model accuracy to validate design decisions early in system design.

This work is structured as follows. Chapter 2 develops an approach to formulate system models from existing discipline specific models. Chapter 3 describes a new design space exploration algorithm enabling reliable system model execution during system design. Chapter 4 presents a new method for quantifying system model accuracy early in system design, enabling design validation before system measurements are available. Chapter 5 summarizes the contributions of this research and identifies future research directions based on this work.
CHAPTER 2. MATHEMATICAL FORMULATION OF SYSTEM MODELS USING EXISTING ENGINEERING MODELS

Building on current system modeling research, this chapter develops an approach for creating multidisciplinary behavioral system models by composition of existing discipline specific models. Specifically, this chapter (1) identifies requirements from published solutions for creating compositional system models from discipline specific models, (2) establishes a mathematical definition for system models, component models, and discipline specific models conforming to these requirements, (3) demonstrates how to combine these component models into compositional system models, and (4) illustrates this approach using a solar powered unmanned aerial vehicle (UAV).

2.1 System Modeling Requirements

System design research has long been interested in modeling system behavior by integrating different models from various disciplines [48]. Many specialties, including system engineering [1], control systems [49], multidisciplinary design optimization [35], software development [50], embedded systems [43], and modeling and simulation [20] have contributed to the body of knowledge about compositional system modeling. From this broad array of research, this section identifies fundamental requirements for compositional system modeling by investigating the following questions:

1. What must a system model accomplish?

2. What is required for a discipline specific model to be usable within a system model?

This section concludes by articulating needed developments to be able to create system models using existing discipline specific models.
2.1.1 System Model Requirements

A system model predicts a system’s response to stimulus [51]. This includes how a system responds not only to a distinct input, but more usefully, how a system responds to a sequence of inputs (a simulation) [52]. In addition, a system model’s prediction must be sufficiently accurate to be useful [53]. Therefore, a system model must approximate the actual system’s response to input sequences.

To accomplish these requirements, the system model components and their interactions must approximate the system components and interactions. A system, by definition, is built from distinct physical components [54]. Similarly, a compositional system model is built from distinct component models. In addition, component model interactions must approximate with sufficient accuracy the actual system’s component interactions.

2.1.2 Component Model Requirements

This section identifies required component model attributes from compositional modeling research and implementations. These attributes are categorized into the categories: functionality, interface, and user interactions. Functionality describes the types and properties of component models. Interface discusses component model interactions. User interactions evaluates a component model’s relationship with component and system model creators.

Component Model Functionality

The functionality of a component model must encompass the different types of discipline specific models it could represent. These include empirical data, mathematical functions, stationary statistical processes, or dynamic systems [55]. Dynamic systems include differential algebraic equations, infinite-dimensional dynamic systems [56], dynamic-stochastic systems, discrete time systems, discrete event systems [39], and hybrid systems [57].

All of these model types must compute outputs based on inputs [58]. In addition, dynamic systems have model state, state initialization, and state progression [59]. Dynamic systems might also require simulation time as an input as well as the ability to request simulation evaluations
at specific future times as an output [44]. Because of these requirements, component models are evaluated with the following sequence [60]:

1. Initialize model state, followed by repeatedly performing steps 2-5.
2. Receive inputs.
3. Update state.
4. Compute outputs.
5. Return outputs.

Finally, multiple instances of the same component model, each having a unique state, should be able to exist within a system model [61].

Component Model Interface

The second category in our investigation of component models is component model interface. A component model’s interface enables model evaluation and composition [62]. Model inputs and outputs are organized into logical groups called ports [63, 64]. Some modeling environments support bidirectional ports both for communication [46] and to facilitate model development by allowing the system model to dynamically determine component model causality (i.e., determine port direction) [63]. Requiring a specific port direction, however, simplifies model composition and does not limit the range of system model behavior [60]. However, it does require an appropriate set of component models [65]. Also, because we are interested in the composition of existing discipline specific models, the causality between discipline specific models and system models is fixed. Changing causality involves re-deriving the discipline specific model, rather than using the existing model. The assumption of fixed causality for external models is even made in declarative modeling languages that automatically determine model causality such as Modelica [66].

The component model interface must also address communication methods and port data. In modeling physical system behavior, the most basic communication mechanism between components is synchronous data transfer [60]. To this, buffered and asynchronous communication
schemes can be added using additional component models [30]. Many implementations recommend that communication should be coordinated by the system model [67–69]. Finally, model composition requires an exact match of the data transferred (units, type, interpretation) for the composition to be correct [58].

**Component Model Users**

Component models have various users that interact with the model in different ways. Here, we consider component model creators and system model creators. These users impose creation, use, and verification requirements.

Creating component and system models requires component models to conform to a specific interface that provides access to model data and control over model evaluation. Because discipline specific modeling tools are created independently without considering compositional modeling, satisfying requirements identified in this section can be accomplished using a software interface that maps the modeling tools’ capabilities to the component model requirements [37]. This interface also facilitates changes such as modifying existing component models, exchanging one component model with another within a system model, and creating subsystems that combine lower level component models. [58].

When creating system models, it is often necessary to combine different types of models (e.g., an event-based voltage controller regulating a continuous current source). Combining different types of models has received a great deal of attention in literature. The compositional modeling environments investigated have successfully combined continuous time, discrete time, discrete event, and other types of component models based on the requirements outlined in this section [70]. Therefore, we assume that these rules are sufficient to enable engineering models to coexist during simulation. This assumption will be tested in Section 2.3.

Component models should be validated before being included into system models. This validation is performed by component model creators but used by system model creators [71]. Therefore, it is necessary to communicate the level of component model validation appropriate for the system model. This can be accomplished by defining a set of component model domains appropriate for various levels of validation (e.g., theoretical domain, various levels of accuracy, domain of proven designs) [72].
2.1.3 Requirements Summary

Fundamental attributes for component and system models from the compositional system modeling literature and implementations identified in Sections 2.1.1 and 2.1.2 upon which we will build are:

1. System model approximate system behavior.
2. Component model approximate component behavior.
3. Component model interactions approximate interactions within the system.
4. Component model must be able to represent the behavior found in system design.
5. Component model evaluation follows the sequence identified in Section 2.1.2.
6. Each component model instance maintains its own state.
7. Composition data transfer is unidirectional.
8. Composition data transfer is synchronous.
9. Composition requires an exact match of the data transferred.
10. Communication is coordinated by the system model.
11. Component model domains define various levels of validation.
12. An interface maps the component model to the discipline specific model.

Sections 2.2 builds upon these attributes in order to reuse existing engineering models to predict multidiciplinary system behavior.

2.1.4 Needed Developments

Although feasible methods of compositional system modeling have been demonstrated in the literature, it is still not a routine part of system design [3]. Several obstacles prevent compositional system models from being used early enough in system design to validate design decisions.
One obstacle is that compositional system modeling presupposes an available population of component models. In practice, this population usually does not exist. Without this, a compositional system model demands more effort to develop than a stand-alone system model because of the requirements listed in Section 2.1.3. Secondly, because system design is a dynamic and iterative process, system models must be able to execute and produce results even as underlying component models are updated during system design. Finally, to validate design decisions using system models, system model accuracy must be quantified even before actual system behavior is known.

This chapter addresses the first obstacle: developing a population of trusted component models. It shows how existing discipline specific models become component models within a system model to predict system behavior. Chapter 3 addresses quantifying and improving reliable system model execution, and Chapter 4 proposes a new method to quantify system model accuracy.

Section 2.2 develops a mathematical formulation of compositional system modeling from which necessary attributes of system model solutions are identified. Section 2.3 demonstrates this formulation by developing a system model for a solar powered unmanned aerial vehicle propulsion system.

### 2.2 System Model Mathematical Formulation

Based on the compositional system modeling requirements identified in Section 2.1, this section proposes an approach for developing system models from existing discipline specific models. This enables existing models to be combined to model system behavior and is especially beneficial for early and ongoing system design analysis. This involves first establishing a mathematical definition of component models from discipline specific models in Section 2.2.1. This definition is shown to be broad enough to cover most types of models used in engineering design while still being a bounded and sufficiently simple definition that enables system model analysis. Using this component model definition, Section 2.2.2 develops a system model formulation and an approach for solving these models. Section 2.2.3 determines system model solution requirements based on these definitions.
2.2.1 Component Model Definition

A component model is a model that describes the behavior of one portion of a system and is able to be combined with other component models by composition into a system model. Based on the component model requirements summarized in Section 2.1.2, this section establishes a mathematical definition for component models developed from existing discipline specific models. This definition has three parts: (1) the component model definition, (2) the discipline specific model definition, and (3) a mapping between the discipline specific model and component model definitions. Then this section establishes that this component model definition is sufficiently broad to cover most types of models used in engineering design. Finally, it describes some specific limitations of this component model definition.

Discipline specific models generally contain four parts: model input, model output, model definition, and model evaluation. To develop component models from existing discipline specific models, changes cannot be made to the discipline specific model’s definition or evaluation because these changes would require reformulation and revalidation. System models, however, require specific component model inputs and outputs for successful composition within a system model. If these inputs and outputs can be exposed to a system model without requiring discipline specific model reformulation and revalidation, then we are able to reuse this discipline specific model.

Accordingly, component models are defined as the input-output relation

\[ y_j = C_j(u_j) \text{ where } \{ u_j \in U_j, y_j \in Y_j \}, \quad (2.1) \]

where

- \( U_j \) set of inputs \( \{ u_j : u_j \in U_j \} \)
- \( Y_j \) set of outputs \( \{ y_j : y_j \in Y_j \} \).

By this definition, a component model receives a set of inputs \( u_j \) from which it computes a set of outputs \( y_j \). The index \( j \) in Equation 2.1 indicates that it is one of a set of component models within a system model. The only demands placed on discipline specific models from this definition is that they: (1) are able to receive inputs from a system model, (2) compute outputs based on these inputs, and (3) return these outputs to a system model.
While Equation 2.1 provides the desired protection of the discipline specific model definition and evaluation, it gives little insight into component model behavior. Therefore, in addition to the component model definition in Equation 2.1, we add the discipline specific model definition
\[ D_j = (U_j, X_j, Y_j, f_j, h_j, x_{j,0}), \]  
(2.2)

where

- \( U_j \) set of inputs \( \{u_j : u_j \in U_j\} \)
- \( X_j \) set of states \( \{x_j : x_j \in X_j\} \)
- \( Y_j \) set of outputs \( \{y_j : y_j \in Y_j\} \)
- \( f_j \) progression function \( f_j : U_j \times X_j \rightarrow X_j \)
- \( h_j \) output function \( h_j : U_j \times X_j \rightarrow Y_j \)
- \( x_{j,0} \) initial state \( \{x_{j,0} : x_{j,0} \in X_j\} \).

The parentheses in Equation 2.2 denote a tuple, which is an ordered collection of possibly heterogeneous elements. Equation 2.2 adds to Equation 2.1 that, within a component model, a discipline specific model \( D_j \) is a representation of a general dynamic system. Equation 2.2 maps from the input set \( U_j \), through the set of states \( X_j \), to the model output set \( Y_j \). These mappings are accomplished by the progression function \( f_j \) and output function \( h_j \). The progression function \( f_j \) maps from one location to another within the state space \( X_j \) based on the current input \( u_j \) and state \( x_j \). This is written in mathematical notation as \( f_j : U_j \times X_j \rightarrow X_j \) (Equation 2.2). The notation \( U_j \times X_j \) is the Cartesian product of the sets \( U_j \) and \( X_j \), which means the input to \( f_j \) can be any combination of \( u_j \in U_j \) and \( x_j \in X_j \). The arrow (\( \rightarrow \)) denotes the function’s mapping of these inputs to its output \( x_j \in X_j \). The output function \( h_j \) maps from the current input set \( U_j \) and set of states \( X_j \) to the output set \( Y_j \) (\( h_j : U_j \times X_j \rightarrow Y_j \) in Equation 2.2). \( x_{j,0} \) is the initial state of \( x_j \).

As the system model evaluates a component model (Equation 2.1), the component model, in turn, evaluates a discipline specific model (Equation 2.2). The five component model evaluation steps (identified in Section 2.1.2) are interpreted as described in Table 2.1 by component models to evaluate discipline specific models.

The discipline specific model definition (Equation 2.2) has limited the types of models that can be used as component models to dynamic systems. Because of this, we must now establish that this component model definition is sufficiently broad to represent the range of discipline specific
Table 2.1: Component Model Evaluation Sequence

| Initialize:  | \( x_j = x_{j,0} \) |
| Receive Inputs: | set discipline specific model inputs from component model inputs |
| Update State:  | evaluate \( f_j \) |
| Compute Outputs: | evaluate \( h_j \) |
| Return Outputs: | set component model outputs from discipline specific model outputs |

models used in system design. To do this, the following sections map three general classes of engineering models (i.e., differential-algebraic equations, discrete event system specification, and computer computation) into this component model definition. These mappings show that these model types are able to be represented by the component model definition in Equation 2.1, Equation 2.2, and Table 2.1. As a result, analysis based on this component model definition applies directly to these types of discipline specific models. Also, models falling into these categories can be used as component models without requiring additional analysis.

These three model classes cover many but not all types of discipline specific models used in system design. For example, stochastic models and nonlinear programming can also be represented by Equation 2.2 but are not covered by these three classes. The process used in this section demonstrates how additional classes of discipline specific models can be mapped to this component model definition. As such, Equation 2.2 provides a simple, common representation of diverse engineering models that enables mathematical analysis of component and system behavior.

**Differential-Algebraic Equations (DAE)**

Differential-algebraic equations (DAE)

\[
F(t, x, \dot{x}, z, u) = 0
\] (2.3)
where

\[ t \text{ time variable } \{ t : t \in T \subset \mathbb{R}^+ \} \]

\[ x \text{ differential variables } \{ x : x \in X \subset \mathbb{R}^n \} \]

\[ \dot{x} \text{ time derivative of } x \]

\[ z \text{ algebraic variables } \{ z : z \in Z \subset \mathbb{R}^p \} \]

\[ u \text{ input variables } \{ u : u \in U \subset \mathbb{R}^m \}, \]

are a general form of constrained nonlinear system of differential equations. DAE represent many types of systems such as mechanical systems, chemical processes, and thermodynamic systems [73, 74]. DAE are a superset of strictly algebraic and strictly differential equations, which are abundant in discipline specific modeling.

Our objective in this section is to show that DAE are a form of component model by mapping the DAE definition in Equation 2.3 into the component model definitions in Section 2.2.1. This mapping enables analysis performed using the component model definition to apply directly to DAEs.

To do this, let the discipline specific model state \( X_j \) from Equation 2.2 be the Cartesian product of DAE sets \( X, Z, \) and \( T \) written

\[ X_j = X \times Z \times T_j. \tag{2.4} \]

This means that the discipline specific model state \( x_j \) is a vector created by appending DAE vectors \( x, z, \) and the component model simulation time \( t_j. \)

Also let discipline specific model input \( U_j \) be the Cartesian product of the DAE input set \( U \) and system simulation time \( T \): \n
\[ U_j = U \times T. \tag{2.5} \]

The solution of the DAE equation \( F \) determines the new DAE state \( x_{j,i+1} \) based on the current states \( x_{j,i}, \) inputs \( u_j, \) and system simulation time \( T. \) Therefore, the progression function

\[ f_j : X \times Z \times U \times T_j \times T \rightarrow X_j \tag{2.6} \]
is the solution to the DAE equation (Equation 2.3). Various methods to solve the DAE equation are available [75], [76]. Combining these equations into the discipline specific model formulation in Equation 2.2 results in

\[ D_{DAE \ j} = (U \times T, X \times Z, f_j, h_j, x_j, 0). \] (2.7)

Based on Equation 2.7, we next establish the DAE evaluation sequence, mapping from Equation 2.7 to the component model evaluation sequence (Table 2.1). The previous system state is shown as \( x_{j,i} \), and the new state is \( x_{j,i+1} \). The index \( i \) is the simulation step index.

Table 2.2: Differential Algebraic Equations (DAE) Evaluation Sequence

| Initialize: \( x_{j,0} \) | \( = (x_0, z_0, t_0) \) |
| Receive Inputs: \( u_j \) | \( = (u, t) \) |
| Update State: \( x_{j,i+1} \) | \( = f_j(x, z, u_j, t_j, t) \) |
| Compute Outputs: \( y_j \) | \( = h_j(x_{j,i}, u_j) \) |
| Return Outputs: \( y_j \) | \( = y_j \) |

While Table 2.2 implies numeric DAE solutions, if the discipline specific modeling tool solves the DAE analytically, then the DAE can be reduced to an algebraic equation before evaluating the system model and thus reduce model evaluation time.

This section establishes that differential algebraic equations, which also include differential equations and algebraic equations, are represented by the component model definition.

**Discrete Event System Specification (DEVS)**

The discrete event system specification (DEVS) is an efficient method of representing event-based systems such as finite-state machines, digital logic, and computer controlled systems. DEVS has also been extended to discrete time, continuous time, and hybrid dynamic systems [44]. Equation 2.8 shows the classic DEVS formulation for the model \( M_{DEVS} \) [61]. A significant concept DEVS introduces is the time advance function \( t_a \). This allows DEVS to specify important times to evaluate the model. This can speed simulations by skipping evaluation steps where the DEVS model does not change. DEVS also introduces an internal transition function \( \delta_{int} \) to represent
internal transitions to new states.

\[ M_{\text{DEVS}} = (U, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a) \]  (2.8)

where

- \( U \) set of inputs \( \{ u : u \in U \} \)
- \( X \) set of states \( \{ x : x \in X \} \)
- \( Y \) set of outputs \( \{ y : y \in Y \} \)
- \( \delta_{\text{int}} \) internal transition function \( \delta_{\text{int}} : X \rightarrow X \)
- \( \delta_{\text{ext}} \) external transition function \( \delta_{\text{ext}} : U \times T \times X \rightarrow X \)
- \( \lambda \) output function \( X \rightarrow Y \)
- \( t_a \) time advance function \( t_a : X \rightarrow T \subseteq \mathbb{R}^+ \)

Mapping DEVS to the discipline specific model specification in Equation 2.2 involves two steps. The first is mapping the DEVS internal and external transition functions (\( \delta_{\text{int}} \) and \( \delta_{\text{ext}} \)) to the component model progression function

\[ f_j = (\delta_{\text{int}}, \delta_{\text{ext}}). \]  (2.9)

Second, the DEVS output function \( \lambda \) and time advance function \( t_a \) are mapped into the component model output function

\[ h_j = (\lambda, t_a). \]  (2.10)

Although the output value from the time advance function \( t_a \) is not specifically mentioned in the DEVS formulation as an output, it is included as an output because Section 2.1.2 requires data transfer be performed by the system model. Similarly, simulation time \( T \) also becomes a component model input in Equation 2.11.

The DEVS discipline specific model formulation

\[ D_{\text{DEVS}} = (U \times T, X, Y, \delta_{\text{int}}, \delta_{\text{ext}}, \lambda, t_a, x_0) \]  (2.11)
is mapped to Equation 2.1 based on the evaluation sequence in Table 2.3.

### Table 2.3: Discrete event System Specification (DEVS) Evaluation Sequence

<table>
<thead>
<tr>
<th>Operation</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize:</td>
<td>$x_0 = x_{j,0}$</td>
</tr>
<tr>
<td>Receive Inputs:</td>
<td>$(u, t) = u_j$</td>
</tr>
<tr>
<td>Update State:</td>
<td>$x_i = \begin{pmatrix} \delta_{int}(x_{i-1}) \ \delta_{ext}(u, t, x_{i-1}) \end{pmatrix}$</td>
</tr>
<tr>
<td>Compute Outputs:</td>
<td>$y = \begin{pmatrix} \lambda(x_{i-1}) \ t_a(x_{i-1}, t) \end{pmatrix}$</td>
</tr>
<tr>
<td>Return Outputs:</td>
<td>$y_j = y$</td>
</tr>
</tbody>
</table>

Equation 2.11 and Table 2.3 establish that DEVS is a special case of the component model definition. Analysis performed based on the component model definition, therefore, applies directly to DEVS.

**Computer Computation**

Because software’s role in defining multidisciplinary system behavior is increasingly important [5], discipline specific models must be able to represent computer computation. A Turing machine is a general model of computation (not a physical computational machine) accepted as being able to represent any computer program [77]. Therefore, this section maps the Turing machine model of computation, and by extension, any computer program, to the component model formulation in Equations 2.1 and 2.2.

A Turing machine

\[
M_{TM} = (K, \Gamma, b, \Sigma, \delta, s, H) \tag{2.12}
\]
where

\[ K \quad \text{finite set of states } \{k : k \in K\} \]
\[ \Gamma \quad \text{tape alphabet } \{\gamma : \gamma \in \Gamma\} \]
\[ b \quad \text{blank symbol } \{b : b \in \Gamma\} \]
\[ \Sigma \quad \text{input alphabet (excludes blank)} \]
\[ \{\sigma : \sigma \in \Sigma \subseteq \Gamma - \{b\}\} \]
\[ s \quad \text{initial state } \{s : s \in K\} \]
\[ H \quad \text{set of halting states } \{H : H \subseteq K\} \]
\[ \delta \quad \text{transition function} \]
\[ \delta : (K - H) \times \Gamma \rightarrow K \times \Gamma \times \{L, R\} \]
\[ L \quad \text{left tape shift} \]
\[ R \quad \text{right tape shift} \]

has a set of instructions \( \Sigma \) written on a movable tape that it can read, write, and execute one instruction at a time. Execution continues until a halting state \( H \) is reached. The final values written on the tape are the machine’s output.

To equate the Turing machine definition in Equation 2.12 to the discipline specific model definition in Equation 2.2, the Turing machine initial state \( s \) is component model initial state \( x_{j,0} \) (Equation 2.13). The input alphabet \( \Sigma \) is the component model input \( U_j \) (Equation 2.14). The transition function \( \delta \) is the component model progression function \( f_j \) (Equation 2.15).

\[ x_{j,0} = s \quad (2.13) \]
\[ U_j = \Sigma \quad (2.14) \]
\[ f_j = \delta \quad (2.15) \]

Component model state \( X_j \) combines both Turing state \( K \) and tape \( \Gamma \) as

\[ X_j = K \times \Gamma. \quad (2.16) \]

We retain the component model output function \( h_j \), which maps from the final contents of the tape to the output set \( Y_j \) when the Turing machine enters a halting state \( k \in H \) shown as \( \Gamma_{\text{final}} \) in
Equation 2.17.

\[ h_j := \Gamma_{\text{final}} \to Y_j \]  

(2.17)

Combining Equations 2.13–2.17, a Turing machine discipline specific model is

\[ D_{\text{Turing } j} = (\Sigma, K \times \Gamma, Y_j, \delta, h_j, s). \]  

(2.18)

Table 2.4 maps the Turing discipline specific model in Equation 2.18 to the component model definition in Equation 2.1. The Turing machine evaluation differs from DAEs (Table 2.2) and DEVS (Table 2.3) in that it continues to compute instructions and iterate within the “Update State” step until a halting state is reached. If a halting state is never reached, Turing machine outputs are never available. While the other model types can also fail to compute a result, the Turing machine acknowledges this behavior explicitly.

<table>
<thead>
<tr>
<th>Table 2.4: Turing Machine Evaluation Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize: ( s = x_{j,0} )</td>
</tr>
<tr>
<td>Receive Inputs: ( \sigma = u_j )</td>
</tr>
<tr>
<td>Update State: do ( \delta(k \times \gamma) ) while ( k \notin H )</td>
</tr>
<tr>
<td>Compute Outputs: ( y_m = h_j(\gamma_{\text{final}}) )</td>
</tr>
<tr>
<td>Return Outputs: ( y_j = y_j )</td>
</tr>
</tbody>
</table>

While the “Update State” computation in Table 2.4 does differ from the other model types previously discussed, the evaluation “do \( \delta(k \times \gamma) \) while \( k \notin H \)” does satisfy the progression function mapping defined in Equation 2.2: \( f_j : U_j \times X_j \to X_j \). It maps from a set of inputs and states to a new state as required.

An important property of computer computation models is that they should be treated as time-varying (i.e., as non-autonomous) systems. The behavior of the transition function \( \delta \) is dependent on the input alphabet \( \sigma \), which can change during the simulation.

Because Turing machines are able to represent any computer program, we have established in this section that the component model definition in Section 2.2.1 is able to represent any computer program.
Component Model Implementation

We have shown that the component model definition introduced in Section 2.2.1 covers a wide range of systems. The tuple formulation enables some aspects of the component models to be explicitly specified (e.g., state, progression function), while allowing their implementations to vary according to the method. As a result, any discipline specific model based on DAEs, DEVS, or computer programs can be used without modification as a component model by providing an appropriate set of inputs and outputs.

Component Model Limitations

This component model definition does have some intentional limitations intended to facilitate developing system models from existing models for engineering design. One limitation of this component model definition is seen in the difference between the component model (Equation 2.1) and the discipline specific model (Equation 2.2). Because the component initial state \( x_{j,0} \) and current model state \( x_j \) are not required to be accessible outside the component model, the system model cannot directly set a discipline specific model’s state. Techniques that require full control over component model state would be unsuccessful because of this component model definition. For example, microelectronics’ “Design for Test” relies on setting a system to a specific state and executing a test sequence at that state [78]. Although this is an important technique, requiring a system model to be able to completely and directly read and write a component model’s state would significantly limit the component models that could be used within a system model. Many discipline specific models purposely hide model state (e.g., object oriented software and proprietary modeling software). Therefore, more discipline specific models can be used as component models because only an input/output interface is required.

A second limitation of this component model definition is the requirement for distinct component input/output interfaces. Some systems, such as neurons in animals, are able to dynamically adapt their interface [79]. Dynamic interfaces would be limited by this rigid component model interface description. Dynamic component interfaces, however, are currently uncommon for man-made components and therefore are not addressed here.
A final limitation is component model causality: outputs are computed based on inputs and model states. This adds constraints to declarative modeling tools such as Modelica by fixing the inputs and outputs between system and component levels. This also confines some economic and human behavior models by limiting the available knowledge to a fixed set of inputs and states.

2.2.2 System Model Definition

Using system model requirements in Section 2.1.1 and the component model definition in Section 2.2.1, we now develop a general mathematical definition for compositional system models, enabling system model evaluation and analysis. If a specific system modeling environment such as Simulink or Modelica is being targeted, then their application specific requirements would also need to be added to this formulation.

A system model \( S \) (Equation 2.19) predicts the response to stimulus of an actual system \( S_a \) (Equation 2.20) [52]. System models are a partial representation of system behavior. A system model \( S \) will typically not consider all inputs to or outputs from the actual system \( S_a \). Therefore, system model inputs \( u \) and outputs \( y \) are subsets of the actual system inputs \( u_a \) and outputs \( y_a \).

\[
y = S(u) \quad \{u : u \in U \subseteq U_a\}, \{y : y \in Y \subseteq Y_a\} \tag{2.19}
\]

\[
y_a = S_a(u_a) \quad \{u_a : u_a \in U_a\}, \{y_a : y_a \in Y_a\} \tag{2.20}
\]

A system model’s inputs \( u \) and outputs \( y \) are assumed to be dynamic sequences (static being a special case) as shown in Equations 2.21 and 2.22 [71]. The input sequence \( u \) progresses by an independent variable appropriate for the system model (e.g., time increment for continuous-time simulations) [80].

\[
u = [u_1, u_2, \cdots, u_l] \quad \{u_i : u_i \in U\} \tag{2.21}
\]

\[
y = [y_1, y_2, \cdots, y_m] \quad \{y_i : y_i \in Y\} \tag{2.22}
\]

With this definition of the system model interface (Equations 2.19, 2.21, and 2.22), we next formulate the system model structure. Model composition defines a relationship between component models: it is a one-way data transfer from the output of one model to the input of another [34]. Using graph theory terminology to represent compositional system models, composition (shown
as an arrow in Figure 2.1) is a doubly-connected directed edge between two component models or vertices (shown as boxes in Figure 2.1) as described by [81]. System model inputs \( u \) and outputs \( y \) are signally-connected edges. Following this pattern, a compositional system model \( S \) can be represented as a directed graph \( G \), as shown in Figure 2.1 [82].

In addition to a visual representation, graph theory also provides mathematical notation and analysis tools for system models. Equation 2.23 states that a system model \( S \) is a graph \( G \) composed of vertices \( V \) and directed edges \( E \) [83]. \( V \) is the set of all component models (Equation 2.24). Graph edges \( E \) represent data transfer within the system model. The set of directed edges \( E \) (Equation 2.25) includes system model inputs \( u \), outputs \( y \), and model composition \( E(Y,U) \) (or all doubly-connected edges in the graph).

\[
S \equiv G = (V, E) \tag{2.23}
\]

\[
V \equiv \{C_j \; \forall \; j \in G\} \tag{2.24}
\]

\[
E \equiv \{u, y, E(Y,U)\} \tag{2.25}
\]
An important feature in system model graphs are cycles where signals can loop within the graph as is the case with Figure 2.1. Depending on the component models in the cycle, they may result in algebraic loops in the system model. To define cycles, we first define a path \( P \) (Equation 2.26) as a set of unique, connected component models.

\[
P \equiv (V, \vec{E})
\]  

(2.26)

where:

\[
V = \{C_0, C_1, \ldots, C_k\}
\]

\[
\vec{E} = \{\vec{E} (0, 1), \vec{E} (1, 2), \vec{E} (k - 1, x_k) , \ldots\}
\]  

(2.27)

A cycle

\[
Y \equiv P + \vec{E} (k, 0)
\]  

(2.28)

is a path that connects back to itself.

To solve system models, the system model graph shown in Figure 2.1 is written as a system of algebraic equations in Equation 2.29. Each vertex, or component model \( C_j \), computes the output \( y_j \) from the input \( u_j \). Each directed edge (composition) equates inputs and outputs. Solving the system model is as simple as solving the system of equations shown in Equation 2.29.

\[
y_1 = C_1 (u_{1,1}, u_{1,2})
\]

\[
y_2 = C_2 (u_2)
\]

\[
y_3 = C_3 (u_3)
\]

\[
u_{1,1} = u
\]

\[
u_{1,2} = y_3
\]

\[
u_2 = y_1
\]

\[
u_3 = y_2
\]

\[
y = y_2
\]  

(2.29)
By substitution, Equation 2.29 becomes Equation 2.30. Because of the system model graph cycle in Figure 2.1, this set of equations can be written as the implicit equation

$$y = C_2(C_1(u, C_3(y))).$$  \hspace{1cm} (2.30)

If the implicit equation converges, the solution can be found by holding the system model input $u$ constant and repeatedly evaluating the graph cycle until the output $y$ converges. Optimized ordering of graph evaluations are discussed by Shaja [84].

Although graph theory representations of systems are common, the preceding system model formulation has restricted component models to represent existing discipline specific models (Equation 2.2). The requirement introduced here – that directed edges represent exclusively model composition – does not limit the ability to model engineering systems. Any transfer (e.g., mass, energy, information) between system components can be represented as composition [85]. Additionally, bidirectional communication such as modeling energy transfer between component models can be achieved using two directed edges in opposite directions. Therefore, system model graphs are able to represent general system structure and behavior [86].

In addition to algebraic loops, which have been illustrated in Equation 2.29, another concern is constraints (also referred to as structural singularities) in the system model [75]. For example, at an electrical circuit node connecting a battery and a capacitor, each of which defines the node voltage, both are constrained to be at the same voltage. This system model formulation, however, requires structural singularities to be resolved before composition is possible. To illustrate this, if we write the composition equations from Equation 2.29 as matrix multiplication, they form an identity matrix

$$
\begin{bmatrix}
  u_{1,1} \\
  u_{1,2} \\
  u_2 \\
  u_3 \\
  y
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0 & 0 \\
  0 & 0 & 1 & 0 & 0 \\
  0 & 0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  u \\
  y_3 \\
  y_1 \\
  y_2 \\
  u_3
\end{bmatrix}
$$

\hspace{1cm} (2.31)
because each equation is simply an assignment. Therefore, structural singularities must be resolved before developing the system model.

Some techniques to resolve structural singularities include grouping constraints into a component model (e.g., developing a SPICE model containing both the capacitors and battery), reformulating component models based on the Pantelides algorithm [75], or adding dynamic constraint convergence to the system model illustrated in Section 2.3.

Some limits to this graph theory representation of system structure include modeling very large, repeated structures or modeling processes without distinct components. For very large repeated structures, such as DRAM memory, this graph representation would be inefficient compared with other higher level abstractions [18]. Systems without distinct components, such as the behavior within fluidized bed reactors (e.g., used in coal combustion), are also not represented well as a graph because of indistinct and continually changing boundaries [87]. While few engineering systems fall into these two categories, both of these limitations can be resolved by adjusting the boundary between component and system model.

2.2.3 Solution Requirements

Sections 2.2.1 and 2.2.2 provide a mathematical formulation for system models by composition of existing discipline specific models. This section gives a critical evaluation of this formulation to determine conditions when the system model is able to compute a solution.

Based on the system and component model definitions above, several requirements emerge that must be satisfied to effectively solve the system model. These are:

1. All component models must compute a solution.
2. Inputs must be within component model domains.
3. The system model solution must converge.

Composition theorems have established the first solution requirement [88].

The second requirement demands that all component model inputs must be from the component model’s domain in order for its solution to be valid. This requirement is from the component model definition in Equation 2.1.
The need for the third requirement – system model solutions must converge – is illustrated by Equation 2.30. Cycles in the system model graph result in implicit system model equations.

Fixed point theory, which studies the convergence properties and solutions of systems in the form

\[ s = F(s), \]  

(2.32)

provides a basis and various methods to solve implicit system models. Here, we briefly summarize some major fixed point theorems and discuss their application to solving implicit system models.

In order to apply fixed point theory to system models, we must first formulate the system model in Equation 2.19 in terms of the fixed point problem in Equation 2.32. To do this, the component models \( C_j \) are decomposed into the discipline specific model functions \( f_j \) and \( h_j \) from Equation 2.2. The resulting function \( F \) in Equation 2.32 is then the set of discipline specific model transition functions \( f_j \) from Equation 2.2. Also, \( s \) in Equation 2.32 is the Cartesian product of the discipline specific model states and inputs in the form \( s = x_k \times h_j(x_i, u_i) \). The output function \( h_j(x_i, u_i) \) transforms the state and input of one model to the input of another model. The system model input \( u \) and simulation time are held constant. As such, the system model solution becomes a fixed point problem at each step of the system model evaluation.

To solve the fixed point problem, the Banach contraction principle establishes conditions for the existence and uniqueness of fixed point solutions that can be found by iteration. This theorem states: let \( S, s \in S \) be a Banach space (complete, normed vector space), let \( F \) be the mapping \( F : S \rightarrow S \), and suppose that

\[ ||F(s_1) - F(s_2)|| \leq \rho ||s_1 - s_2||, \forall s_1, s_2 \in S, \]  

(2.33)

and \( 0 \leq \rho < 1 \)

then there exists a unique \( s^* \) satisfying \( s^* = F(s^*) \). Also, \( s^* \) can be found by successive approximation starting from any arbitrary \( x \in S \) [40]. This means that if component model states and inputs are complete, normed vector spaces and Equation 2.33 are satisfied, then the system model will converge to a unique solution by repeatedly executing the model and holding the system model inputs and simulation time constant.
If the inputs and states of component models are not Banach spaces, then the Knaster-Tarski and Tarski-Tantorovitch theorems establish the existence of fixed points and convergence by successive approximation for spaces $S$, defined as partially ordered sets [89].

In practice, these theorems are used as conceptual supports for attempting successive approximations to find system model solutions. If convergence fails, these theorems may be used to analyze the failure.

In addition to iterative convergence, methods of reformulating models to eliminate implicit system model solutions may be very helpful to reduce simulation evaluation time [75]. Because our focus is on reusing existing models, these techniques are not discussed here.

2.2.4 Modeling in System Design

In summary, developing component models requires (1) mapping component model inputs to discipline specific model inputs, (2) mapping discipline specific model outputs to component model outputs, (3) establishing the component model’s domain $X$, and (4) mapping the component model evaluation sequence to the discipline specific model evaluation.

System model evaluation requires (1) computing system model outputs $y$ by solving the set of system model equations $S$ based on the system inputs $u$, (2) ensuring that all component model inputs are within that model’s domain, (3) ensuring each component model computes a solution, (4) iterating within algebraic loops until they converge, and (5) if the model simulation fails, re-initializing all component models before restarting the simulation.

Section 2.2 has established a mathematical formulation for compositional system models from existing discipline specific models that was not previously available in literature. We have demonstrated that this formulation, while being bounded and useful for system model analysis, is general enough to describe most engineering systems including algebraic, differential, event-based, and software systems. Using this system model formulation, we have presented an approach for solving compositional system models and have identified a set of conditions to reliably solve system models. This problem formulation, the solution requirements, and these modeling assumptions will be demonstrated in Section 2.3.

While not simple, this approach provides significant advantages for the design of product families. By coupling system models to component designs, ongoing design decisions in various
disciplines can be explored before physical prototypes are available. Because new products are frequently adapted and improved from previous designs and use many of the same components, component and system models from previous systems become the foundation for new system models, thus greatly reducing the effort of creating new models.

2.3 Solar Powered UAV System Model

To illustrate the process of modeling system behavior from existing discipline specific models, this section develops a compositional system model for the solar powered unmanned aerial vehicle (UAV) propulsion system shown in Figure 2.2, which was designed and built as part of this research. This section describes the process of creating component models from exiting discipline specific models, illustrates developing and solving a system model, and verifies the results of this proposed approach by comparing its results to a second system model implemented in Mathworks’s® Simulink® and to measured UAV behavior.

This UAV propulsion system contains all of the model types discussed in Section 2.2 implemented in a variety of modeling tools. The system model requires algebraic loop convergence and structural singularity resolution. Therefore, it illustrates and validates this approach of modeling system behavior from existing discipline specific models.

The process of creating a system behavioral model involves: (1) defining system model inputs and outputs, (2) identifying components and interactions, and determining ports, (3) assembling discipline specific models and creating component models, (4) creating a system model, and (5) verifying the system model.

2.3.1 UAV Component Models

With system model inputs and outputs identified, this section identifies system components, interactions, and ports, and develops component models from existing discipline specific models. The UAV components, which were selected in advance of system modeling, are shown in Figure 2.2. The components are:

1. Solar Array: SunWorld >17.5% efficient mono crystalline solar cell array.
2. MPPT: ST Microsystems ISV005V1 maximum power point tracker evaluation board.
3. Battery: Tenergy 3-cell, 2200mAh lithium-ion polymer battery (31261 V2).


5. Speed Control: E-flite 20A brushless DC motor controller (EFLA311B).

6. BLDC Motor: E-flite 480 brushless DC motor.
7. Prop.: APC 10x5 propeller.


The physical connections between components, also defined before system modeling, assist in defining component model ports and determining port direction. For example, because the propeller and BLDC motor are directly connected by the motor shaft, the composition of these component models is defined by this physical connection. Several established techniques such as structural digraphs [75], bond graphs [90], and linear graphs [65] can assist in establishing component model ports and port direction (i.e., component model causality).

Because most system model interactions can be defined in different ways involving different ports and port directions, building system models from existing component models is usually an iterative process. For example, two functionally equivalent solar array models were available to the UAV system model. The first computes solar power based on MPPT resistance, the second computes solar current from MPPT voltage. The second formulation was chosen because it enabled reconfiguration of the system model (e.g., composition with the battery model). This illustrates both that system models are often not unique and that only pair-wise composition between component models is needed within the system model.

As was identified in Section 2.1.2, interfaces are needed to communicate between the system model and discipline specific models. We developed a model composition library to provide this functionality. This library enables system models to write inputs, read outputs, and evaluate models implemented in Excel, Matlab®, Simulink®, PSpice, NX, and other discipline specific modeling tools. This library’s implementation is not discussed in this dissertation.

Discipline specific models were assembled or developed in a variety of different modeling tools. The solar array model is a nonlinear algebraic model implemented in Microsoft® Excel [91]. The solar array component model

\[ i_s = \text{SolarArray}(v_s, r_s, \phi, T) \]  

(2.34)
produces an output current $i_s$ based on an input voltage $v_s$, solar irradiance $r_s$, incidence angle $\phi$, and temperature $T$.

Because the solar power captured is dependent on the solar cell voltage, a maximum power point tracker (MPPT) separates the solar array voltage and provides an intelligent DC to DC converter that adjusts the solar array voltage to extract the maximum power. It is modeled as a DEVS model based on the SPV1020 application notes and is implemented as a Matlab “.m” file [92]. The ISV005V1 board was modified to disable battery current regulation and to set the voltage levels needed for this system. The MPPT component model

$$[t_m, v_m, i_l] = \text{MPPT}(t, i_m, v_l)$$

computes the next MPPT simulation time $t_m$, solar array voltage $v_m$, and load current $i_l$ based on the simulation time $t$, solar array current $i_m$, and load voltage $v_l$.

A lithium ion battery provides energy for take-off and temporary loss of solar power (e.g., scattered clouds or flight maneuvers). The battery model, implemented as a C library, is a nonlinear system of differential equations with direct feedthrough [93, 94]. The battery component model

$$[v_b, q] = \text{Battery}(t, i_b)$$

computes voltage $v_b$ and charge $q$ based on simulation time $t$ and current $i_b$.

A capacitor filters the MPPT output voltage and reduces the battery’s peak current. It is a differential equation model implemented in Simulink. The capacitor component model

$$[v_c] = \text{Capacitor}(t, i_c)$$

computes voltage $v_c$ based on simulation time $t$ and current $i_c$.

The speed control provides electronic commutation for the brushless DC motor as well as power regulation for the servo controller [95]. The speed controller model is computer computation based on micro controller logic implemented in C. The speed control component model

$$[t_s, v_p, v_s, i_p, q_p] = \text{SpeedControl}(t, v_p, c_p, \theta_p, i_pb, i_ps)$$

34
computes next computation time $t_s$, brushless DC motor voltage $v_p$, servo controller voltage $v_s$, current drawn $i_p$, and brushless DC motor MOSFET gate commands $q_p$ based on simulation time $t$, system voltage $v_p$, speed command $c_p$, motor angle $\theta_p$, brushless DC motor current $i_{pb}$, and servo motor current $i_{ps}$.

The brushless DC motor is an electromechanical circuit implemented as a differential algebraic equation (DAE) in PSpice [96]. This DAE includes asymmetric sequencing of three parallel LRC circuits. Although the mechanical and electrical behavior could have been placed into separate models, because our focus is on reusing existing models, we chose to use the MicroSim model as it was developed. The brushless DC motor model

$$\begin{bmatrix} \theta_m, \omega_m, i_m, \end{bmatrix} = \text{BLDCMotor}(t, v_m, \tau_m, q_m) \quad (2.39)$$

computes motor angle $\theta_m$, angular velocity $\omega_m$, and motor current $i_m$, from simulation time $t$, supply voltage $v_m$, motor torque $\tau_m$, and MOSFET gate commands $q_m$.

The propeller model is an algebraic model that converts input angular velocity $\omega$ to thrust $T$ and torque $\tau$ [97]. The propeller model was implemented as a Matlab “.m” function. The propeller component model

$$\begin{bmatrix} T_p, \tau_p \end{bmatrix} = \text{Propeller}(\omega_p) \quad (2.40)$$

computes thrust $T_p$ and torque $\tau_p$ based on propeller angular velocity $\omega_p$.

The servo controller distributes servo commands and power to the servo motors and brushless DC motor. It is an algebraic model implemented in Excel. The “ServoControl” component model

$$\begin{bmatrix} v_{so}, i_{so}, c_{so} \end{bmatrix} = \text{ServoControl}(v_{si}, i_{si}, c_{si}) \quad (2.41)$$

produces output voltage for servo motors $v_{so}$, sums current draw $i_{so}$ from the servo motors, distributes the command signals $c_{so}$ for the servo, and brushless DC motors. Its inputs are supply voltage $v_{si}$, current $i_{si}$ from each servomotor, and input command sequence $c_{si}$.

Each of the 4 servomotors (elevator, left aileron, right aileron, and rudder) is a subsystem composed of a microprocessor, potentiometer, power switching circuit, gear train, linkage, and UAV control surface. The servomotors are modeled as a DC motor regulated by a PID feedback
controller resisting a load proportional to servo angle [98]. This becomes a differential-algebraic equation that was implemented in Simulink. The servo motor component model

\[ [\theta_e, i_e] = \text{ServoMotor}(t, v_e, c_e) \] (2.42)

computes servo angle \( \theta_e \) and current \( i_e \) based on simulation time \( t \), supply voltage \( v_e \) and servo command \( c_e \).

These component models, which conform to Equation 2.1, provide building blocks to model system behavior. Each of these component models evaluates a discipline specific model, represented by Equation 2.2, and is implemented using a variety of modeling tools. The UAV discipline specific models are from the three types of models discussed in Section 2.2.1.

In addition to the UAV system model presented here, we also developed several system models of a commercial system not presented in this paper. This enabled us to investigate developing component models from a diverse set of discipline specific models that were already part of system design. These system models resulted in insights into system behavior that were previously unavailable. Each of the discipline specific models investigated was able to be represented by Equation 2.2 and formulated into a component model (Equation 2.1). The primary challenge to accomplish this was the one-time effort of mapping design tool variables to component model variables. When this was completed, CAD models, finite-element models, finite-difference models, statistical models, and component measurements, among others, became component models from which we developed system models.

The range of component models developed for these systems motivates our claim that Equations 2.1 and 2.2 are able to represent most types of models used in engineering design.

### 2.3.2 UAV System Model

With UAV system model inputs and outputs established in Section 2.3 and component models developed in Section 2.3.1, the next step is creating a solar powered UAV propulsion system model. This requires the composition of the system model inputs through the component models to the system model outputs.
Because Equation 2.1 requires component models to have unidirectional inputs and outputs, the system model is created by mapping system model inputs to component model inputs, component model outputs to component model inputs, and component model outputs to system model outputs.

The relationship between components has already been established as part of system design (e.g., system design established that the solar array interacts with the MPPT). The primary difficulties encountered when developing the UAV system model were (1) ensuring the needed component model inputs and outputs are available (discussed in Section 2.3.1) and (2) handling constraints in the system model.

The UAV model has one constraint (i.e., structural singularity): battery and capacitor voltage must be at the same voltage. Unfortunately, no simple relationship exists between the capacitor and battery voltages and currents. Each of these models was developed separately in a different modeling tool so analytic constraint solutions are not applicable. Therefore, the ConvergeV component model is added to the system model to enforce this constraint.

Since both battery and capacitor models produce a voltage output based on a current input, ConvergeV determines the current inputs where capacitor and battery voltages are equal and the sum of the currents into this node is zero. Because both capacitor and battery are dynamic models and DEVS requires time inputs to progress in a positive direction \( T \subseteq \mathbb{R}^+ \) in Equation 2.8), ConvergeV converges battery and capacitor voltages over time. Voltage convergence, however, must be quick enough to not significantly reduce system model accuracy.

ConvergeV implements proportional feedback control to adjust capacitor current based on the difference in voltages. The ConvergeV component model

\[
[v_v,i_{v1},i_{v2}] = \text{ConvergeV}(i_{vm},i_{vs},v_{v1},v_{v2})
\]

(2.43)

computes output voltage \( v_v \) by adjusting the battery \( i_{v1} \) and capacitor \( i_{v2} \) currents. It receives the MPPT current \( i_{vm} \), the servo current \( i_{vs} \), the battery voltage \( v_{v1} \), and capacitor voltage \( v_{v2} \) as inputs.

Using Equations 2.34–2.43, our next objective is to model UAV system behavior. Each block of the system model graph in Figure 2.2 is a component model forming Equation 2.44. Each vector is composition and is written as an assignment in Equation 2.45.
\[ i_s = \text{SolarArray}(v_s, r_s, \phi, T) \]
\[ [t_m, v_m, i_l] = \text{MPPT}(t, i_m, v_l) \]
\[ [v_p, q] = \text{Battery}(t, i_b) \]
\[ [v_c] = \text{Capacitor}(t, i_c) \]
\[ [v_v, i_{v1}, i_{v2}] = \text{ConvergeV}(i_{vm}, i_{v1}, v_{v1}, v_{v2}) \]
\[ [t_s, v_p, v_s, i_p, q_p] = \text{SpeedControl}(t, v_p, c_p, \theta_p, i_{pb}, i_{ps}) \]
\[ [\theta_m, \omega_m, i_m] = \text{BLDCMotor}(t, v_m, \tau_m, q_m) \]
\[ [v_{b}, q] = \text{Battery}(t, i_b) \]
\[ [v_{c}] = \text{Capacitor}(t, i_c) \]
\[ [v_{v}, i_{v1}, i_{v2}] = \text{ConvergeV}(i_{vm}, i_{v1}, v_{v1}, v_{v2}) \]
\[ [t_s, v_p, v_s, i_p, q_p] = \text{SpeedControl}(t, v_p, c_p, \theta_p, i_{pb}, i_{ps}) \]
\[ [\theta_m, \omega_m, i_m] = \text{BLDCMotor}(t, v_m, \tau_m, q_m) \]
\[ [t_s] = \text{Time}(t, t_m, t_s) \]

\[ t = t_i \quad v_{v2} = v_c \quad c_{is} = u(t, 4) \]
\[ v_s = v_m \quad v_p = v_v \quad v_e = v_{so} \]
\[ r_s = u(t, 1) \quad c_p = c_{so}[1] \quad c_e = c_{so}[2] \]
\[ \phi = u(t, 2) \quad \theta_p = \theta_m \quad v_m = v_{so} \]
\[ T = u(t, 3) \quad i_{pb} = i_m \quad c_m = c_{so}[3] \]
\[ i_m = i_s \quad i_{ps} = i_{so} \quad v_a = v_{so} \]
\[ v_l = v_v \quad v_m = v_p \quad c_a = c_{so}[4] \]
\[ i_b = i_{v1} \quad \tau_m = \tau_p \quad v_r = v_{so} \]
\[ i_c = i_{v2} \quad q_m = q_p \quad c_r = c_{so}[5] \]
\[ i_{vm} = i_l \quad \omega_p = \omega_m \]
\[ i_{vs} = i_p \quad v_{si} = v_s \]
\[ v_{v1} = v_b \quad i_{sl} = [i_c, i_l, i_a, i_r] \quad y(t) = T_p \]
Due to the number of cycles in the system model graph, iterative system model convergence is necessary. For example, in Figure 2.2, a forward arrow assigns solar array current to the MPPT. The reverse arrow assigns MPPT voltage to the solar array. In practice, not every graph cycle results in an algebraic loop. For example, the SPICE BLDC Motor model requires a time increment to apply changes in supply voltage to changes in motor velocity. Nevertheless, an algebraic loop does exist among the MPPT, ConvergeV, Battery, and Speed Control component models. Therefore, these models must be evaluated iteratively holding the simulation time constant until the solution converges. The same solution could be found by iteratively solving all component models but the computational expense would be greater.

The UAV system model is simulated by repeatedly solving Equations 2.44 and 2.45 while progressing the system simulation time and stepping through the system model input sequence $u$. Figure 2.3 compares the step response of the system model computed value (solid lines) to the UAV measured value (dashed lines) for the voltage and current between the SpeedControl and ConvergeV component models, as well as for the propeller thrust output.
Figure 2.3: Solar Powered UAV Measured vs. Computed Voltage, Current, and Thrust

Figure 2.3 shows the system model computed solution approximates the actual UAV behavior. The largest deviations are the spikes in the current plot as the propeller accelerates and
decelerates. This error then induces the error in the voltage and thrust plots, illustrating error propagation through the system model. This error is due to the low-fidelity speed control model that does not provide the current limiting functionality of the actual speed controller.

Implementing a simplified version of the UAV model entirely within Simulink® results in the identical solution as the corresponding compositional system model solution. The maximum deviation in the output between these two models for the simulation sequence shown in Figure 2.3 is $4.62 \times 10^{-7}$N. This is within the numerical noise of these two different implementations. This shows that the compositional system modeling approach developed in this paper is able to produce the same results as traditional stand-alone system modeling even for dynamic, nonlinear system models.

An important motivation for compositional system models is their potential for reuse. This UAV model, in fact, was initially developed and validated as 4 separated subsystem models: (1) MPPT, solar array, (2) battery, capacitor, ConvergeV, (3) speed control, BLDC motor, propeller, and (4) servo controller, servo motor. These subsystem models were combined incrementally by including the appropriate composition equations from Equation 2.45. The working UAV system model can now be readily broken into other system models such as solar battery chargers or ground based robots.

In addition to reusing system models for new purposes, system models can be reused with different fidelity models in order to analyse different aspects of a design. For example, when developing the solar power circuit, we used a low fidelity motor model as we verified power point tracking and convergence. The compositional system model structure enables developers to rapidly switch between different models and enables new approaches to multifidelity modeling.

This section has illustrated that, using the system model formulation from Section 2.2, system models can be developed from existing discipline specific models. These system models are able to predict system behavior and are able to produce the same results as traditional stand-alone system modeling techniques.

The UAV system model highlights the usefulness of this proposed approach within the system design process. Most organizations repeatedly design a specific type of product (e.g., machines, airplanes, cars). As such, at the start of design of a new system, a population of applicable discipline specific models is available from previous systems. These existing models can become
component models to predict a new system’s behavior. Some uses for these system models include evaluating design concepts, verifying ongoing component and system design decisions, identifying design sensitivities to be tested, investigating the root cause of failures, and determining requirements for next generation systems.

### 2.4 Summary

In this chapter, we have developed an approach of predicting system behavior using existing discipline specific models. To accomplish this, compositional system modeling requirements were first identified from literature. Conforming to these requirements, Section 2.2 established a general mathematical formulation for developing system models by composition of existing discipline specific models and established that this formulation is able to cover most models in system design. Based on this system model formulation, Section 2.2.3 established a solution method and a set of necessary requirements to find a solution. Finally, Section 2.3 evaluated and verified this approach using a solar powered unmanned aerial vehicle propulsion system.

The primary benefits of this proposed approach are (1) existing discipline specific models become a large population of trusted component models from which system models can be quickly developed, (2) the discipline specific models used to design the system’s components can be combined to provide immediate and ongoing analysis of the impact of design decisions on the system’s behavior, and (3) system modeling is able to build upon the growing power and capabilities of discipline specific modeling tools.

Two additional obstacles impede this proposed approach from fully benefiting system design. The first is that in order to validate design decisions, system model accuracy must be quantified before the actual system behavior is known. The second is that compositional system models must compute feasible solutions even while underlying discipline specific models change during system design. Solutions to these two obstacles are the subjects of the following chapters.
To be useful for validating design decisions, system models must reliably compute system model solutions even as component and system models change during system design as designs are refined. Compositional system models introduce several failure modes that often result in infeasible or failed model evaluation. Based in the system model formulation developed in Chapter 2, this chapter introduces a new theory and method to identify the system model feasible domain. Based on this method, we developed several approaches to control and improve the feasible domain. This enables reliable system model evaluation even as underlying component and system models change during system design [10]. To do this, this Section 3.1 evaluates the system model formulation in Section 2.2 to determine model failure modes, Section 3.2 creates a formulation for system model feasibility to identify several important failure modes, Section 3.4 develops a design space exploration algorithm that quantifies the system model domain, and illustrates this algorithm using a solar powered unmanned aerial vehicle model. The method presented here enables systematic improvements of compositional system model feasibility.

System models developed by model composition have several potential sources of failures, including each component model [88], each composition [99], data communication between models [100], and system model convergence [84]. For system models to be useful for validating design decisions, they must produce results over the desired model domain and in the presence of design changes.

Model verification, which is the process of ensuring correct model evaluation, includes manual testing, automated test vector execution, random test generation [101], and metamorphic testing where tests “mutate” to explore new areas of the system under test [99]. While these methods have proven to be effective for software and digital logic verification, they do not take advantage of the nature of system models. We propose that system model feasibility can be more effectively quantified based on known system model behavior and failure modes.
In this chapter we will build upon sequential sampling techniques that have proven to be efficient methods for design space exploration. The “Efficient Global Optimization” (EGO) algorithm built on Gaussian process regression has demonstrated both high accuracy and efficiency on a variety of data sets [102]. Kleijnen illustrates selecting new samples based on previous metamodel predictions [103]. Xiong expands this work and combined Gaussian and Bayesian processes to, among other objectives, introduce uncertainty into the Gaussian process model [104]. Shan uses sequential sampling using radial basis functions to quantify the behavior of unknown functions for high dimensional problems [105].

To quantify system model feasibility, this paper also builds on several recent publications on design space exploration. Specifically, Huang employs Gaussian process regression to identify the feasible design space of various constrained functional designs [106]. Devanathan, similarly, identifies design space boundaries as a polytope (n-dimensional polygon) [107]. Finally, Malak identifies valid input domain boundaries of a fixed set of input data using support vector machines [108]. We extend these methods in two ways: (1) this paper addresses dynamic compositional system models in addition to functional models, (2) it explores a binary space (system model domain) and identifies the boundaries of this system model domain.

In summary, this chapter develops a design space exploration algorithm to quantify a system model’s feasible domain, identifies portions of the system model domain where solutions exist, and enables failure boundaries to be classified based on failure mode. This algorithm can be used during system model validation to quantify and improve the feasibility of system models. Section 3.1 presents a formulation for system model feasibility and identifies types of system model evaluation failures. Section 3.2 develops a design space exploration algorithm to determine system model feasible domains and illustrates this using a solar powered unmanned aerial vehicle system model.

3.1 System Model Evaluation Failures

To be useful in design, a system model must compute valid results. Based on the system model formulation in Chapter 2, this section identifies various types of evaluation failures exhibited by compositional system models.
Table 3.1 summarizes system model failure modes. The top half of the first column (component failure) identifies failure modes for component models (which include discipline specific models). The bottom half of the first column (system failure) are system model failures that combine all of the component model failures and introduces several additional failure modes. Because the reuse of existing discipline specific models includes both the component model definition and model evaluation, we have not distinguished between model definition failures and model evaluation failures (e.g., errors introduced by the solver). The column Failure Type describes how these failures behave in relation to model evaluation. Specifically, **dynamic failures** are deterministic failures that depend both on model inputs and model states. **Static failures** (a subset of dynamic) are deterministic failures that depend only on model inputs. **Constant failures** are deterministic failures that do not depend on model inputs or states. **Stationary failures** are probabilistic failures that are described by a constant statistic (e.g., data communication between models is 99.999% reliable), and **dynamic statistical failures** are probabilistic failures that are described by a changing statistic (e.g., the likelihood of failure is proportional to the model state $x$).

Because this failure classification focuses on the behavior of a failure during simulation, it differs from other model failure characterizations describing failures in terms of model structure. For example, a structural singularity describes a system model structural error resulting in a behavioral formulation failure. An algebraic loop in the model structure could result in convergence failure during evaluation. Stiffness or instability in component or system models could result in state or range failures during model evaluation. This focus on the behavior of failures, rather than the structure of failures, enables us to develop a behavioral model of the feasible system model domain.

In this paper, we focus specifically on static and dynamic failures. The constant and stationary failures can be effectively addressed by techniques available from reliability research [101, 109]. Resource failures (dynamic statistic) can be addressed directly by adding the necessary resources.

Quantifying system model feasibility due to static and dynamic failures is the process of finding the intersection of the valid component model inputs $U_j$, component model states $X_j$, valid computation, model convergence, and valid system model range $Y$. Model composition, however, does not typically produce a geometric combination of feasible spaces. Rather, it is a combination
Table 3.1: Component and System Model Failures

<table>
<thead>
<tr>
<th>Component Failure</th>
<th>Description</th>
<th>Failure Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>( u_j \notin U_j ) (Eq. 2.1)</td>
<td>static</td>
</tr>
<tr>
<td>State</td>
<td>( x_j \notin X_j ) (Eq. 2.2)</td>
<td>dynamic</td>
</tr>
<tr>
<td>Computation</td>
<td>( f_j ) or ( h_j ) failure (Eq. 2.2)</td>
<td>dynamic</td>
</tr>
<tr>
<td>Resource</td>
<td>evaluation resources unavailable</td>
<td>dynamic</td>
</tr>
<tr>
<td>Development</td>
<td>improper definition</td>
<td>constant</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System Failure</th>
<th>Description</th>
<th>Failure Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component</td>
<td>any component failure [14]</td>
<td>see above</td>
</tr>
<tr>
<td>Convergence</td>
<td>fails to converge [40]</td>
<td>dynamic</td>
</tr>
<tr>
<td>Communication</td>
<td>data transfer failure [68]</td>
<td>stationary</td>
</tr>
<tr>
<td>Formulation</td>
<td>unable to define ( y = S(u) )</td>
<td>constant</td>
</tr>
<tr>
<td>Composition</td>
<td>incorrect composition [58]</td>
<td>constant</td>
</tr>
<tr>
<td>Range</td>
<td>( y \notin Y ) (Eq. 2.19)</td>
<td>dynamic</td>
</tr>
</tbody>
</table>

defined by the structure and dynamics of the system model. While this hinders an analytical computation of the system model feasible space, we can discover this feasible space computationally.

Some important considerations for investigating dynamic failures, which include static failures, are: (1) a failure at any point results in a system model failure, (2) failures are interdependent, and (3) a failure may be induced by a preceding action from another part of the system model. We next develop a domain exploration algorithm to determine system model feasibility.

### 3.2 Feasible Domain Exploration

To validate design decisions using system models, feasible solutions \( y \) must exist for the input sequences \( u \) that the physical system will encounter. In this section, we develop a design space exploration algorithm to identify the domain where valid system model solutions exist. This enables us to quantify and improve system model feasibility due to dynamic system model failures.

System model feasibility exploration is a tool to improve system model verification before the model is used in system design. The up-front computational expense of formally quantifying system model feasibility should be offset by more effective utilization of the system model during the later, more computationally expensive design, analysis and optimization processes.
To quantify system model feasibility, Section 3.2.1 presents a formulation for the feasibility of system models. Section 3.4 develops a design space exploration algorithm to quantify system model feasibility. This method is illustrated using a solar powered UAV propulsion system described in Section 3.3. Section 3.4.2 classifies the valid design space boundaries based on the failure mode. Section 3.4.3 identifies methods to improve system model feasibility.

### 3.2.1 Defining a Searchable Space

Because system models are general dynamic systems, feasible solutions depend on both the input magnitude and sequence. For example, a sequence that successfully lands a UAV would result in mostly disastrous landings if the sequence was arbitrarily rearranged. Since the system model input domain $U$ includes all possible values of each element of the sequence, the full system model domain cannot be effectively explored. Most systems, however, have a much smaller set of useful input sequences. This section presents a method of establishing a searchable input domain for feasibility exploration by composition of the system model with two functions: an input function and a feasibility function.

The input function is

$$u_\phi = I(\phi) \quad \{ \phi \in N \subset \mathbb{R}^n \}, \{ u_\phi \in U_\phi \subset U \}$$

(3.1)

where $\phi$ is an input configuration parameter array and $u_\phi$ is the system model input sequence. The input function configuration parameters $\phi \in N$ provide a continuous space suitable for evaluating model feasibility, even for discontinuous and event-based system models. For example, a UAV throttle command could be any arbitrary sequence of commands, most being inappropriate for UAV flight. The input function $I(\phi)$ in this example confines system model input $u$ to a set of sequences common to UAV flight such as takeoff, climb, cruise, descent and landing. The input parameter array $\phi$ configures these sequences.

The feasibility function interprets the system model evaluation and output sequence as being valid (1) or invalid (0). The feasibility function

$$F(S(u)) = \{1, 0\}$$

(3.2)
receives as input the system model and the input sequence it is to evaluate. It then evaluates the system model and determines if the model evaluation is valid. It evaluates the system model and produces a “1” if the system model $S$ was able to compute valid results and “0” if it was not. This requires both range checking and process protection in system and component model execution, which was addressed in Chapter 2.

The composition of the feasibility function (Equation 3.2), system model (Equation 2.19), and input function (Equation 3.1) produces $F(S(I(\varphi))) = \{1, 0\}$, which enables functional system model evaluation over the continuous space $N$ to be evaluated as either a valid or invalid simulation. We will refer to the set of inputs $\varphi$ where the feasibility function returns “1” as the feasible solution set $F_\varphi$. With this, we define the system model feasibility ratio as

$$R = \frac{\int \cdots \int F(S(I(\varphi)))) d\varphi}{\int \cdots \int d\varphi},$$

(3.3)

which is the hyper-volume ratio of the feasible design space to the total design space.

The challenge our design space exploration algorithm must solve is to efficiently quantify the size of the feasible solution set $F_\varphi$ when it can only be evaluated one configuration, $\varphi$, at a time and each model evaluation can be computationally expensive. Before addressing this, Section 3.3 introduces the UAV system model that is used to illustrate this algorithm.

### 3.3 UAV System Model

This section introduces a solar powered UAV propulsion system model with which we will illustrate feasibility exploration. Although this model (shown in Figure 3.1) is a subset of the system model developed in Section 2.3, this model does exhibit all of the dynamic and static failures identified in Table 3.1.

The solar powered UAV propulsion system illustrated in Figure 3.1 has two inputs (the throttle command and initial battery charge) and produces a single output (thrust). It is comprised of the following:

1. solar array (sixteen 0.5-V wing-mounted solar cells),
2. lithium ion polymer battery (2-cell, 800mAh),
3. servo controller (Pololu Micro Maestro 6-Channel USB servo controller),
4. speed controller (E-flite 20A EFLA311B),
5. brushless DC motor (ERC BL300 1400KV),
6. propeller (APC 7x6 LP07060SF).

![Graph of a Simple Solar Powered Unmanned Aerial Vehicle Propulsion System](image)

Figure 3.1: Graph of a Simple Solar Powered Unmanned Aerial Vehicle Propulsion System

The first step to prepare this UAV system model for feasibility exploration is to ensure component models are only evaluated within their valid domain \((u_j \in U_j)\) and state \((x_j \in X_j)\). If system model execution exceeds these bounds, execution is terminated. This is usually a simple comparison of the input and state to their valid ranges. We next create appropriate input and feasibility functions. For valid UAV model execution, the following conditions must be satisfied:

1. The solar panel model must be from 6.6V–7.4V (domain failure).
2. The battery model must be from -20A–1A (domain failure).
3. The battery model must be from 1.0%–100.0% of its total charge (state failure).
4. The motor torque must be \( \leq 100\text{Nm} \) (domain failure).

5. The motor current must be \( \leq 20\text{A} \) (state failure).

6. The propeller model speed must be from 0–95Hz (domain failure).

7. Component models must compute solutions (component failure).

8. The system model must converge to a solution (convergence failure).

9. Propeller thrust must be sufficient for takeoff (range failure).

These ranges of valid model execution were readily available during component model development and should be captured for each component model as part of system model development.

Accounting for these failures, the feasibility function

\[
F_{\text{UAV}}(S(u)) = \begin{cases} 
1 & \text{if } \int_{0}^{t_{\text{max}}} dy > T \\
0 & \text{otherwise}
\end{cases}
\]  

(3.4)

produces a 1 if the various failure modes are avoided and if the integral of the thrust is greater than the threshold \( T \) within the take-off duration \( t_{\text{max}} \). The input function

\[
I(\varphi) = \begin{pmatrix} \varphi_1 \\ C(\varphi_2) \end{pmatrix}
\]  

(3.5)

defines the system model inputs: the initial battery charge \( \varphi_1 \) and the throttle command waveform \( C(\varphi_2) \). The throttle command \( C(\varphi_2) \) is a piecewise-linear function that linearly interpolates between a fixed low and an adjustable high throttle speed.

The system model, feasibility function, and input function shown here for the solar powered UAV propulsion system can be developed for most engineering systems. Although developing system models is typically challenging, creating input and feasibility functions should not be difficult. These two functions capture design information that is a standard part of system design specifications: system inputs and performance ranges. These functions produce a searchable input domain and a binary output range for evaluating system model feasibility.
3.4 Model Feasibility Design Space Exploration

Using the feasibility (Equation 3.2) and input (Equation 3.1) functions creates a searchable input space $U_\phi$. This section creates a system model design space exploration algorithm to efficiently find the set of feasible solutions to the system model and to quantify system model feasibility (Equation 3.3). The set of model solutions, referred to as the feasible solution set $F_\phi$, and its boundaries describe where model solutions exist and how to improve system model feasibility.

Several properties of system model feasibility aid the development of a design space exploration algorithm. The first property is that feasible solutions are part of closed sets but may not be connected sets. This means that the boundary between feasible and infeasible solutions is within the feasible solution set though the set may have holes or may be in multiple isolated regions. Second, the system model solution is either feasible “1” or infeasible “0” with no gradient. As a result, any slope between feasible and infeasible regions is a prediction error. Next, because we know that (1) boundaries are within our feasible solution set, (2) all values within this set are 1, and (3) all values outside are 0, the feasible solution set can be described by its boundaries. As such, the design space exploration algorithm searches for the feasible solution set boundaries.

To quantify system model feasibility based on these properties, we propose a design space exploration algorithm that builds upon other sequential sampling methods [106]. This method, illustrated in Figure 3.2, performs the following steps:

1. sample the input space $U_\phi$ at random until both feasible and infeasible samples exist,
2. estimate the feasible design space by linearly interpolating available samples,
3. identify the most gradual transitions between the feasible and infeasible regions to refine,
4. identify the least explored areas of design space $U_\phi$ to refine,
5. prioritize the next samples by combining transition and unexplored area metrics,
6. add new samples at the highest scoring locations,
7. exit based on number of samples or the stability of the model feasibility metric,
8. repeat this sequence from step 2 until the feasibility estimate has stabilized or the maximum number of samples is reached.
The implementation in this paper addresses deterministic failures of system models but could be extended to address statistical failures.

The first step is to sample the input space at random until both valid and invalid samples exist. Latin hypercube sampling is used because it ensures that samples are spread over the input
space by selecting new sample locations at random from distinct segments of the input space [110]. This is typically just a few samples to seed the Gaussian process regression design space estimation.

With a sample population containing both valid and invalid results, the second step is to estimate the valid design space. Gaussian process regression (Kriging) provides an accepted and flexible method of estimating the design space. It only requires an arbitrary set of sample locations \( \varphi \) and system responses \( f(\varphi_i) \) as input with which it predicts both the system response \( \tilde{f}(\varphi) \) and estimates the prediction error \( \varphi(\varphi) \) [111]. The predicted system response

\[
\tilde{f}(\varphi) = \sum_{i=1}^{n} \lambda_i(\varphi)f(\varphi_i) + Z(\varphi)
\]  

(3.6)

estimates system model validity \( \tilde{f}(\varphi) \) for unexplored model inputs \( \varphi \) [111]. The weighting functions \( \lambda_i(\varphi) \) and zero-mean stochastic process \( Z(\varphi) \) are determined as part of Gaussian process regression. Prediction error

\[
\varphi(\varphi) = E[(\tilde{f}(\varphi) - f(\varphi))^2]
\]

(3.7)

is defined as the expected value (\( E \)) of squared error between the predicted value \( \tilde{f} \) and actual value \( f \) at location \( \varphi \). It is also estimated as part of Gaussian process regression using the set of system responses \( f(\varphi_i) \) as described and implemented by Lophaven [112].

Using the predicted response and error, we next sample \( \tilde{f}(\varphi) \) and \( \varphi(\varphi) \) at regular intervals from the maximum to the minimum value of each input dimension of \( \varphi \) to produce the \( n \)-dimensional prediction matrix \( \tilde{F} \)

\[
\tilde{F} = \tilde{f}(\varphi_n)
\]

(3.8)

and the error matrix

\[
\Phi = \varphi(\varphi_n).
\]

(3.9)

Segmenting the prediction matrix \( \tilde{F} \) produces the feasibility space estimate

\[
\tilde{F}_b = \begin{cases} 
1 & \text{if } \tilde{F} >= 0.5 \\
0 & \text{otherwise}
\end{cases}
\]

(3.10)
with 0.5 as the threshold between valid and invalid samples. The number of dimensions of each of these matrices is the number of input function configuration parameters in $\phi$. The number of elements in each dimension establishes the prediction and error resolution.

The prediction matrix $\tilde{F}$ is used in step 3 to identify transitions between valid and invalid regions. The estimated error $\Phi$ is used by step 4 to identify unexplored areas in the design space.

The average value of $\tilde{F}_b$ estimates system model feasibility ratio defined in Equation 3.3. The number of elements in each dimension of $\tilde{F}$ is $j$.

The third step is to identify the most gradual transitions between the valid and invalid regions. Although the true boundary of the valid region changes immediately from 1 to 0, the feasibility space prediction $\tilde{F}$ interpolates between sampled locations. Consequently, a slowly changing transition in $\tilde{F}$ means the valid region boundary is not well defined. To identify gradual transitions, we will first identify the transition regions and then attenuate the quickly changing transitions.

Gradient-based edge detection, which is used in image processing, would be ineffective at finding gradual transitions in $\tilde{F}$ because both valid and invalid regions contain gradually changing values. Because we are looking for transitions in a binary space, we can simply look for transitions through the 0.5 threshold. To do this, we multiply each element of the predicted response by the Gaussian weighting function

$$G(V, \mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

which amplifies the feasible solution set boundary.

Although the Gaussian weighting function identifies transitions, it weights them equally. To identify slowly changing boundaries, an averaging convolution kernel (shown as a 2-dimension $m \times n$ convolution kernel in Equation 3.13) is convolved (written as $*$) with the results of the Gaussian weighting function [113]. This results in a spatial average over adjacent model predictions. For slowly changing transitions, adjacent values of $\tilde{F}$ will receive a similar value from the Gaussian weighting function. The averaging convolution will, consequently, have little effect on slow transitions but will attenuate sharp transitions more strongly.
\[ A = \frac{1}{mn} \begin{pmatrix} 1 & \ldots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \ldots & 1 \end{pmatrix} \]  \quad (3.13)

Combining these operations produces the n-dimensional model transition matrix

\[ T = A \ast G(\tilde{F}, \mu, \sigma) \]  \quad (3.14)

whose maximum values are the slowest model transitions.

The fourth step is to identify the least explored areas of the input space \( U_\phi \). The feasibility space error (in Equation 3.9) provides an estimate of error that increases as the distance from samples increases. This can be used directly to identify the least explored areas of the design space.

The fifth step determines the next sample(s) by combining transition and unexplored area matrices. The resample priority matrix

\[ P_i = \Phi_i T^k \quad \forall \ i \in \phi_m \]  \quad (3.15)

is created by the element-by-element multiplication of the feasibility error matrix \( \Phi \) and the transition matrix \( T \) raised to the power of \( k \). Because the error is 0 at existing sample locations, multiplying by \( \Phi \) ensures that new samples are not from previously sampled locations. To prioritize refining boundaries over exploring unexplored regions, \( k \) can be set to a value greater than 1. The sixth step adds new samples at the location of the maximum values of \( P \).

The seventh step tests the exit criteria. Feasibility exploration can either exit based on the number of samples or the standard deviation of the feasibility prediction over some number of previous samples. If the standard deviation is less than a threshold

\[ \sigma(R_{n-j}, \ldots, R_n) < \sigma_t, \]  \quad (3.16)

feasibility exploration can terminate. A larger threshold results in a faster search but less accurate boundaries. A \( \sigma_t \) of 0.01 would exit after the 37th sample in Figure 3.4.
The eighth step continues feasibility exploration if the exit criteria is not satisfied.

Figure 3.3 shows the result of the first 40 samples of the feasibility exploration using the UAV model. The configuration parameters $\varphi$ are the throttle command (vertical axis) and the initial battery charge (horizontal axis). Figure 3.3(a) shows the prediction matrix $\tilde{F}$. The light region in the center of this plot is the feasible solution set $F_{\varphi}$. The circles are the samples taken to identify the feasible solution set. These circles are shaded according to the failure type. Dark circles (all within the feasible region) indicate a successful simulation.

The UAV model’s feasible region, shown in Figure 3.3(a), is due to interactions within the model and cannot be fully captured by a fixed range of throttle command and initial charge. Although the feasibility search algorithm is able to find multiple disconnected regions, the UAV model valid region is a simply connected set, which is typical of the feasible regions we evaluated.

Figure 3.3(b) shows the re-sample priority matrix $P$ (Equation 3.15). The larger black circle ("Throttle Cmd" $\approx$ 5700 and "Charge" $\approx$ 0.9) has the greatest resample priority and is therefore the location of the next sample. We set $k = 2$ in Equation 3.13 in order to preferentially refine known boundaries over exploring new areas for this example.

Feasibility exploration estimates system model feasibility using Equation 3.11. The solid trace in Figure 3.4 shows the change in the feasibility estimate versus the number of samples. Both the feasibility estimate and the feasible solution set boundary stabilize in about 40 samples.

### 3.4.1 Search Method Evaluation

The input function (Equation 3.1) and feasibility function (Equation 3.2) enable various design space exploration algorithms to evaluate the feasibility of dynamic compositional system models. In this section, we compare the evaluation of 3 different design space exploration algorithms by comparing their computed feasibility (Equation 3.11). Our objective is to show that this proposed method converges to the same feasibility value at an acceptable rate.

To do this, we compare the proposed method to Latin hypercube random sampling and mean-squared error sampling. Both of these sampling methods perform unbiased global searches. The feasible space of each of these sampling strategies is interpolated using Gaussian process regression. Latin hypercube random sampling produces random samples that are guaranteed to cover all portions of a space. Therefore, it is preferred over pure random sampling for design
exploration. Mean-squared error sampling chooses the next sample at the location farthest from existing samples.

Figure 3.4 plots the convergence of the feasibility metric of these three sampling methods over the number of samples using the solar powered UAV system model. Each method converges
to a similar feasibility value. The method developed in this paper converges to within 1% of the final solution within 40 samples. Mean-squared error achieves a similar accuracy in 80 samples. Latin hypercube sampling converges to this same feasibility value after 178 samples. The faster rate of convergence of this feasibility search illustrates the value of searching based on our a priori model of the feasible system model domain.

![Convergence Comparison Graph](image)

**Figure 3.4**: Comparison of Random, Mean-Squared Error, and the Proposed Feasibility Design Space Exploration

### 3.4.2 Feasible Solution Set Boundary Classification

In addition to quantifying feasibility and identifying the feasible solution set, the feasible solution set boundaries can be classified based on the nearest failure. For example, in Figure 3.3(a), the bottom boundary is due to the thrust range failure, the top boundary is the propeller valid speed domain failure, the left boundary is the battery current domain failure, and the right boundary is the solar panel voltage domain failure defined in Section 3.3. While this boundary classification
can be accomplished visually for feasibility spaces with three or fewer dimensions, our future work addresses automatic boundary identification.

Categorizing the feasible solution set boundary in this way enables failures that define the system model feasible solution set to be identified. This also allows system model feasibility improvements to be prioritized. For example, if the valid range of propeller speeds were increased, the top boundary of the feasible solution set in Figure 3.3(a) would rise, the size of the feasible solution set would increase, and system model feasibility would increase.

3.4.3 Increasing Feasibility using System Model Feasibility Exploration

Feasibility exploration facilitates integrating compositional system models into system design. To use feasibility exploration in system design:

1. system models are developed to only produce results within the valid model ranges $U_j$, $X_j$, etc.,

2. input and feasibility functions are defined to produce a continuous, searchable input space and binary response for the system model: $F(S(I(\phi)))$,

3. algorithm parameters $k$ (exploration priority), $\sigma_t$ (termination threshold), and maximum samples are set, and

4. feasibility exploration is performed and recorded periodically during system design.

Using feasibility exploration, system model developers can qualify system model execution, identify valid input ranges, determine primary failure modes, identify model changes that reduce feasibility, and progressively improve system model feasibility.

The feasibility estimate alone can serve as a threshold to qualify and compare system models. If feasibility is too small, it is unlikely that a system model will effectively predict system behavior especially in the presence of design changes. In addition, comparing the feasibility of different versions of a system model can identify the impact of specific model changes on system model feasibility.

The feasible solution set provides a more complex view into system model feasibility, as is seen in Figure 3.3(a). Specific simulation conditions can be compared to the feasible solution
set to determine whether the needed simulation conditions will evaluate correctly. Identifying the failure type along the feasible solution set boundaries indicates the source of system model failures. The larger the boundary of a specific failure mode, the more significant the failure mode. These measurements enable model improvements to be prioritized in order to increase the feasible solution set and determine needed simulation conditions. For example, a faster UAV take-off requires a greater throttle command and larger initial charge. This is limited by system model failures due to the propeller model’s valid speed range. By increasing this range, system model feasibility would grow and new simulation conditions would become feasible.

3.5 Summary

For system models to be useful in system design, they must produce feasible results over the desired model domain and throughout the entire design process. This is especially important for compositional system models due to their additional sources of failure. Of the many types of system model failures, dynamic and static failures, which are not well addressed by reliability research, have been the focus of this chapter.

In this chapter, we presented a formulation for system model feasibility and developed a design space exploration algorithm to identify the feasible system model domain. This enables developers to (1) determine whether solutions to specific simulation conditions exist, (2) identify if changes in a system model affect feasibility, (3) identify significant sources of system model failures, and (4) select system model improvements that will lead to feasibility improvements.

Our initial studies into evaluating and improving the high dimensional performance of the proposed design space exploration found that the growth in number of samples appears to be less than exponential with respect to the number of dimensions. Other computational enhancements, however, would benefit high dimensional design space exploration. For example, the literature recommends radial basis functions over Gaussian process regression for high dimensions [105]. In addition, Equation 3.15 should be represented by a sparse, rather than a full matrix, to reduce the growth in data size. Although good high dimensional performance is important, the number of dimensions of feasibility design space exploration is the number of the input function parameters ($\phi$), not size of the system model domain or number of model failures. Therefore, the search dimensions are likely to grow much more slowly than the complexity of system models.
CHAPTER 4. SYSTEM MODEL ACCURACY

Having addressed system modeling based on an existing population of discipline specific models in Chapter 2 and feasible model evaluation in Chapter 3, this chapter proposes a method to quantify system model accuracy early in system design. This is a significant part of the system modeling process because decisions at the beginning of system design, when the least is known about the new system, have the greatest impact on the system [114]. Without quantifying their accuracy however, system models cannot be used to validate design decisions because unquantified errors could result in unbounded deviations between a system model and the actual system behavior [115].

This chapter proposes a deterministic method of bounding system model accuracy early in system design. This method bounds accuracy by determining the maximum system model error by introducing known or estimated variational models estimating specific errors into system models. It bounds system model error within the maximum difference between the original system model and the variational system model. By bounding system model error, design decisions can be validated early and throughout system design.

System model accuracy is the measure of how well a system model approximates the actual system’s behavior [116]. We have classified the various approaches of quantifying system model accuracy discussed in the literature into the following three general approaches: (1) comparing model results to trusted system data, (2) expert review, and (3) sensitivity analysis. While each of these methods have desirable attributes, none is sufficient to quantify system model accuracy early in system design [16].

The most common approach for quantifying system model accuracy is to compare the system model’s prediction to measured system data [5, 117]. Uncertainty analysis and other methods of statistical model validation are examples of this approach [118]. One key limitation to this approach is that measured system responses or resulting probability density functions are usually
only available at the end of the system design process [119]. This hinders our objective of design validation early in system design [120].

Expert review is a versatile approach for validating system models before measured behavior is available [121]. This method benefits from the expertise of designers supported by various analysis tools and has proven to be a reliable method of model validation [122]. Expert review, however, does not quantify model accuracy but typically provides a pass/fail judgement on model validity. Also, unexpected and emergent behavior found in systems can be difficult to verify using expert review [123].

A third approach of quantifying system model accuracy is sensitivity analysis [124]. Sensitivity analysis is a powerful method of estimating the output variation based on known design variations [125]. It is useful for providing an estimate of system model steady-state errors. These errors are propagated to and combined at the system model output. Our previous work using this approach illustrates some of its limitations for nonlinear functions and general dynamic systems [12]. To facilitate this analysis, the impact of different errors on system outputs are frequently assumed to be linear and independent [126]. Consequently, these results are typically only valid locally [127].

While each of these methods provides important benefits, none is sufficient to quantify system model accuracy early in system design [128]. This chapter develops a quantitative measure of system model accuracy that does not require measured system behavior. It addresses both dynamic and steady-state errors over the range of a system’s operation and is applicable across most engineering systems.

In this chapter we combine aspects of these approaches with dynamic system theory to provide a method to bound error for compositional system models by (1) augmenting system models with variational models describing known design variations, (2) bounding system model accuracy by searching for the maximum deviation over a set of simulation conditions, and (3) determining if the design’s objectives are satisfied at this maximum deviation.

To establish this method, Section 4.1 presents a mathematical formulation for the accuracy of compositional system models, Section 4.2 develops a method to quantify the accuracy for these system models, and Section 4.3 illustrates and verifies this method using a solar powered unmanned aerial vehicle (UAV) propulsion system model.
4.1 System Model Accuracy

To analyze system model accuracy, this section develops a general formulation for compositional system model accuracy of engineering systems and identifies various sources of system model error.

Accuracy is the measure of how well a system model predicts physical system behavior. This is quantified as the magnitude of model error

\[ \varepsilon_a = \|S_a(u) - S(u)\| \] (4.1)

which is the p-norm of the difference between the actual physical system \(S_a\) and the system model \(S\) responses [80].

This definition requires knowing the physical system’s response \(S_a(u)\), which is unknown until the end of system design. To quantify accuracy early in system design, Section 4.1.1 evaluates the system model definition to determine sources of model error and Section 4.2 replaces the measured system response \(S_a(u)\) in Equation 4.1 with a variational system model augmented with models of the known errors.

4.1.1 System Model Error Sources

Before we can effectively predict system model accuracy, we must understand the different sources of error between a system model \(S\) and the physical system \(S_a\). Based on the system model formulation in Chapter 2, error is introduced in the following four locations [128].

1. Component model error
2. Unmodeled behavior
3. Error propagation
4. Incorrect structure

Component model error and unmodeled behavior could be either epistemic uncertainty (due to lack of knowledge) or aleatory uncertainty (irreducible errors) [129].
Component model error is the error each component model introduces into the system model. It could be due to complex component dynamics being approximated with simpler models (low order modeling), approximate model parameters, deterministic component models representing statistical behavior, computational inaccuracies, or other deviations.

Unmodeled behavior is physical system behavior that is missing from the system model. It is caused by either explicitly omitting aspects of a system or due to unknown system behavior. It includes unmodeled system dynamics, measurement error, error in initial conditions, or unmodeled external disturbances.

Once error is introduced into a system model, error propagation occurs within the system model. While the error in a component model in isolation will distort the model results, errors propagated through a system model could be amplified or cause system model evaluation to fail.

The final error source, incorrect structure includes incorrect component models and incorrect component model composition such as mismatched units, missing signal conversions (e.g., expecting pressure but receiving force), or incorrect component models for a system design. Because this chapter focuses on designed, rather than natural systems, the system structure is known. This is because the system design defines the components in a system and the relationship components to each other. Incorrect structure can therefore be identified and corrected by comparing the known system structure to the system model structure. As a result, further analysis of errors due to incorrect structure will not be presented in this chapter.

4.2 Quantifying System Model Accuracy

This section builds upon established interval analysis techniques to create proposes a deterministic method of bounding system model error early in system design [130]. This method involves (1) developing variational models of known errors, (2) augmenting the system model with variational models describing model error, and (3) maximizing the difference between the system model and the error augmented system model by adjusting variational parameters. To focus this chapter on system model accuracy, we assume that component models are available with known accuracy over a defined domain. We also assume that error due to unmodeled behavior is known.
Our objective is to bound the actual system’s behavior \( y_a \) within some bound \( \varepsilon \) from the system model’s prediction \( y \). Stated mathematically,

\[
y_n + \varepsilon \geq y_{a,n} \geq y_n - \varepsilon
\]

(4.2)

where \( n \) is the index to the output sequence. Therefore, \( y_n \) and \( y_{a,n} \) are corresponding outputs from the system model \( y \) and actual system \( y_a \), respectively. The constant \( \varepsilon \) must be greater than or equal to the maximum deviation of the actual system behavior from the system model. The subscript \( n \) indicates some index of the output sequence \( y \). Equation 4.2 differs from Equation 4.1 in that rather than trying to find an exact error value \( \varepsilon_a \), we are only trying to bound the difference between the actual system behavior \( y_a \) and the system model prediction \( y \).

Equation 4.2 is illustrated in Figure 4.1. The system model \( S \) estimates a system’s response (center line) to the input sequence \( u \). Its response varies from the actual system response \( S_a \) (lower line). By augmenting the system model with known design variations, the variational system model \( S_v \) defines a range of possible system behavior (shown as the inner dark region in Figure 4.1). To simplify accuracy analysis, the region \( S(u) \pm \varepsilon \) defines the maximum difference between the system model \( S \) and variational system model \( S_v \), and satisfies Equation 4.2. The horizontal axis is the system model independent variable \( t \), which is often simulation time.

![Figure 4.1: System Model S, Physical System Behavior S_a, Variational System Model S_v, and Error Bounds S ± ε Response to Input u](image-url)

The ability to bound the actual system behavior (Equation 4.2) early in system design allows designers to determine if the actual system produced from a system design can achieve its objectives, thus enabling system design decisions to be validated. This section develops a method
of satisfying Equation 4.2 for compositional system models defined in Chapter 2. To do this, Section 4.2.1 develops variational system models from known system model errors. Section 4.2.2 identifies conditions enabling actual system behavior to be contained within the maximum deviation of the system model and variational system model. Section 4.2.3 finds the maximum error ε from the variational system model. Section 4.2.4 establishes criteria for variational models to satisfy these conditions.

### 4.2.1 Variational Models

Based on available component models, known component model accuracy, and known unmodeled behavior, this section develops variational models quantifying the range of possible behavior due to system model errors.

Component model error can be described by the variational model

\[ y'_j = V_j(y_j, u_j, v) \]  

(4.3)

which adds the variational parameter

\[ \{ v : -1 \leq v \leq 1, v \in \mathbb{R}^n \} \]

that adjusts \( V_j \). The variable \( j \) is the component model index. The variational model \( V_j \) re-interprets a component model’s output \( y_j \) based on the component model input \( u_j \), and the variational parameter \( v \) to produce the variational component model output \( y'_j \). This output combines the model behavior and model error. Equation 4.3 can represent both structured and unstructured errors such as additive, multiplicative, and parameter uncertainty [115].

The variational model \( V_j \) (Equation 4.3) can be developed based on expected model deviations for new components, from measured error distributions for components in development, or from field and manufacturer’s data for legacy components. Developing variational models will be illustrated in Section 4.3

For a more concise notation for component model error, we combine the component model \( C_j \) with the variational model \( V_j \) as

\[ y'_j = C'_j(u_j, v) = V_j(C_j(u_j), u_j, v) \]  

(4.4)
and refer to $C'_j$ as a variational component model. This resulting variational component model $C'_j(u_j, v)$ describes a region of possible behaviors for the component. The variational parameter $v$ defines one possible behavior from the region.

In addition to error within component models, unmodeled behavior within the system model may also cause the system model to deviate from the actual system behavior. The variational model

$$y'_j = V_u(y_j, v) \quad (4.5)$$

where

$$\{v: -1 \leq v \leq 1, v \in \mathbb{R}^n\}.$$ 

describes unmodeled behavior by producing a new model output signal $y'_j$ based on the variational model input signal $y_j$ and variational parameter $v$. The unmodeled behavior variational model does not have the input $u_j$ from the component model error variational model in Equation 4.3 because it operates on a specific system model value $y_j$.

To evaluate system behavior with design variations, the system model $S$ is augmented with variational models to produce the variational system model $S_v$. To illustrate this, Figure 4.2 (a) shows a system model containing a motor and propeller. Adding variational models results in the variational system model shown in Figure 4.2 (b). The variational system model $S_v$ describe the range of expected system behavior. Variational model $V_\tau$ adds unmodeled behavior in the propeller torque load such as losses and inertia due unmodeled bearings. $V_\omega$ modifies the motor angular velocity output signal (e.g., includes error in the motor speed constant).

Of the error sources in Section 4.1.1, we have explicitly addressed component model error and unmodeled behavior and added these to the variational system model. Error propagation his handled by the variational system model due to the system model structure. As error introduced by the variational models, it propagates through the system model as part of the simulation.

The variational system model

$$y_v = S_v(u, v) \quad (4.6)$$

is evaluated in the same manner as the original system model $S(u)$. The system model variational vector $v$ includes all of the variational parameter vectors within the variational system model.
4.2.2 Bounding Actual System Behavior

Having introduced known errors into the variational system model, our objective in this section is to identify conditions where the actual system’s behavior is bounded by the variational system model. To do this, we want to find conditions where the actual system behavior is always bounded between the system model plus or minus the error limits $\varepsilon$ shown here:

$$y_n + \varepsilon_j \geq y_{a,n} \geq y_n - \varepsilon_k.$$  \hspace{1cm} (4.7)

The index $n$ indicates the corresponding sample location between the system model results $y$ and actual system behavior $y_a$. We define

$$\varepsilon = \|y_v - y\|_\infty$$  \hspace{1cm} (4.8)

as the maximum difference between the variational system model outputs $y_v$ and system model outputs $y$ [53]. The indices $j$ and $k$ in Equation 4.7 indicate $\varepsilon$ at two locations defined by variational parameter values $v_j$ and $v_k$.

The infinity norm used in Equation 4.8 is defined as the maximum of the absolute value of the values within the norm.
Subtracting $y_n$ in Equation 4.7 results in

$$\varepsilon_j \geq y_{a,n} - y_n \geq -\varepsilon_k$$

(4.9)

for each output $y_{a,n}$ and $y_n$ of the simulation sequence. Substituting Equation 4.8 into Equation 4.9 and replacing the actual error at each sample location $(y_{a,n} - y_n)$ with the maximum error over the simulation ($\|y_a - y\|_\infty$) results in the inequalities

$$\|y_{v,j} - y\|_\infty \geq \|y_a - y\|_\infty$$

(4.10)

$$\|y_{v,k} - y\|_\infty \geq \|y_a - y\|_\infty$$

(4.11)

because simplifying $-\varepsilon_k$ reverses the direction of the inequality. This is helpful because it means that only one bound is needed to satisfy Equation 4.9. We will refer to this bound as $y_{v,j}$.

To this point, we are able to bound the error between the actual system response $y_a$ and system model $y$ by the maximum difference between the variational system model $y_{v,k}$ and system model $y$. To simplify our analysis of variational models in Section 4.2.4, we would like show that if we bound the actual system behavior by the variational system model, Equation 4.7 is satisfied. To do this, we use the reverse triangle inequality

$$\|a - b\|_\infty \geq \|a\|_\infty - \|b\|_\infty$$

(4.12)

that provides a relationship between the norm of the difference of vectors and the difference of the norm of the vectors. In context of this development, the triangle inequality says that the maximum error is greater than or equal to the error at the maximum.

Applying this to each side of Equations 4.10 and 4.11 results in

$$\|y_{v,j} - y\|_\infty \geq \|y_{v,j}\|_\infty - \|y\|_\infty$$

(4.13)

and

$$\|y_a - y\|_\infty \geq \|y_a\|_\infty - \|y\|_\infty.$$
but Equation 4.13 does not indicate whether Equation 4.13 remains greater than or equal to Equation 4.14.

To re-establish the relationship between Equations 4.13 and 4.14, we establish the requirement

\[ \|y_{v,j}\|_\infty \geq \|y_a\|_\infty \geq \|y_{v,k}\|_\infty \]  

(4.15)

and

\[ \|y_{v,j}\|_\infty \geq \|y\|_\infty \geq \|y_{v,k}\|_\infty \]  

(4.16)

ensuring the infinity norm variational system model bounds both the actual system behavior and system model. In that case

\[ |\|y_{v,j}\|_\infty - \|y\|_\infty| \geq |\|y_a\|_\infty - \|y\|_\infty| \]  

(4.17)

for some variational parameter \( v \). We simplify Equation 4.18 further by adding \( \|y\|_\infty \) to both sides resulting in

\[ \|y_{v,j}\|_\infty \geq \|y_a\|_\infty \]  

(4.18)

as a necessary condition to satisfy Equation 4.7.

Therefore, if the variational model at some variational parameter \( v \) bounds the \( \infty \)-norm of the actual system and system model, we are then able to bound actual system behavior using Equation 4.7. This result applies to any system model defined by Section 2.2.

### 4.2.3 Maximum Error

With the ability to bound the actual system behavior with some \( \varepsilon \) established in Section 4.2.2, this section identifies one variational parameter \( \varepsilon_{\text{max}} \) satisfying Equation 4.7.

If the variational system model \( S_v \) satisfies Equation 4.7 for the input sequence \( u \), the maximum difference between the variational system model \( S_v(u, v) \) and system model \( S(u) \), or

\[ \varepsilon_{\text{max}} = \max_v \|S_v(u, v) - S(u)\|_\infty \]  

(4.19)
also satisfies Equation 4.7. Global optimization techniques could be used to find this maximum [131]. This yields

\[ y_n + \varepsilon_{max} \geq y_{a,n} \geq y_n - \varepsilon_{max} \]  

(4.20)

which satisfies Equation 4.7.

It is important to note the dependency of maximum error in Equation 4.19 on the input sequence \( u \). Because we are also considering non-linear inputs to non-linear system models, it is uncommon that this dependency can be removed.

### 4.2.4 Error Model Requirements

Sections 4.2.2 and 4.2.3 have addressed conditions where variational system models are able to bound actual system behavior. In this section, we evaluate properties of variational models that enable them to satisfy Equation 4.16. The system model formulation in Section 2.2 provides a method to solve multidisciplinary system behavior that typically cannot be formulated into a closed-form analytical solution. In the same way, we do not expect a closed-form solution for variational system models. However, we would like to be able to perform experiments using the variational system model to bound the actual system behavior based on known system model uncertainties. This involves (1) determining the appropriate variational model to bound known model uncertainties, (2) exciting the variational system models with expected system inputs, and (3) ensuring each variational model bounds system behavior independently according to Equation 4.16.

For this evaluation, we expand Equation 4.16 in terms of the system models

\[ \| S_v(u, v_j) \| \_\infty \geq \| S_a(u) \| \_\infty \]  

(4.21)

Using this equation, we first investigate parameter uncertainty. Next we determine variational model requirements for a simple system model composed of linear time-invariant component models. This section concludes by extending these requirements to general system models.
Parameter Uncertainty

Parameter uncertainty is when the correct models are available but the correct model parameters are uncertain [115]. In this case, variational models are used to define the possible parameter range. At some value within this range, the variational system model equals the actual system model

\[ S_a(u) = S_v(u, v_a). \]  \hspace{1cm} (4.22)

At the value \( v_a \) where the actual system equals the variational system model, Equation 4.21 is satisfied.

While the ability to bound actual system behavior with system models containing parameter uncertainty is a useful first step, system models for complex continuous systems are unlikely to contain only parameter uncertainty. Continuous systems are frequently represented by reduced order models which do not satisfy Equation 4.22. The following section determines variational model requirements needed to satisfy Equation 4.21 for a simple system model that does not satisfy Equation 4.22.

Two-Component Linear Time-Invariant System Model

The objective of this section is to determine conditions for the variational model \( V \) to satisfy Equation 4.21 for the simple linear time-invariant system model

\[ y = S(u) = C_2(C_1(u)) \]  \hspace{1cm} (4.23)

where the component models \( C_1 \) and \( C_2 \) are the linear state equations

\[ \dot{x}_i = F_i x_i + G_i u_i \]  \hspace{1cm} (4.24)

\[ y_i = H_i x_i + J_i u_i \]  \hspace{1cm} (4.25)

with vector variables and constant matrices.
and where \( i \) is the component model index. In addition, we assume the actual system behavior for Equation 4.23 is

\[
y_a = S_a(u) = C_2(C_a(C_1(u)))
\]

(4.26)

where \( C_a \) contributes the dynamic behavior of the actual system not included in \( C_1 \). Finally, the variational system model for Equation 4.23 is

\[
y_v = S_v(u, v) = C_2(V(C_1(u), v))
\]

(4.27)

and the variational model \( V \neq C_a \) must be selected to satisfy Equation 4.21.

Recognizing that \( y_1 = C_1(u) \) is identical in Equations 4.26 and 4.27 and substituting Equation 4.25 into Equation 4.21 produces

\[
\|H_2x_{2,v} + J_2V(y_1, v)\|_{\infty} \geq \|H_2x_{2,a} + J_2C_a(y_1)\|_{\infty}
\]

(4.28)

where each inequality is a weighted sum of the component model \( C_2 \) matrices \( H_2 \) and \( J_2 \) with the state \( (x_{2,v}, \ldots, x_{2,a}) \) (for the variational system model and actual system) and input vectors \( (V(y_1, v), \ldots, C_a(y_1)) \) (for the variational system model and actual system).

To illustrate Equation 4.28, we interpret Equation 4.28 geometrically as region defined by \( x_{2,v} \) and \( V(y_1, v) \) within a set of intersecting open half-spaces defined as

\[
a_1x_1 + a_2x_2 + \ldots + a_nx_n \geq b.
\]

(4.29)
Each row of Equation 4.28 defines a positive and negative open half-space of the variational system model

$$\|H_{2x_2,v} + J_2V(y_1,v)\|_\infty \geq y_a$$

(4.30)

within which we want to contain the actual system behavior. The intersection of the positive half-spaces is the dark region on the top of Figure 4.3. The intersection of the negative half-spaces is the light region on the bottom of Figure 4.3. The variational model defines some range values of $x_2$ and $u_2$ in Figure 4.3 illustrated as an ellipse.

![Figure 4.3: Variational Model Vector Ranges Satisfying Equation 4.28](image)

For the variational system model to contain the actual system behavior, we would want the variational model $V(y_1,v)$ to bound the actual system behavior $C_a(y_1)$ and the resulting component model $C_2$ state $(x_2,v)$ to contain the actual system state $x_{2,a}$. This means that we would want to choose a value for the variational parameters $v$ that is on the edge of the region illustrated as the ellipse in the center of Figure 4.3. In addition, the variational model parameter $v$ should also place the system model in the strictly positive region (dark) or strictly negative region (light) of...
Figure 4.3 of the $C_2$. These regions contain the maximum and minimum variational system model output $y_v$.

Figure 4.3 illustrates several variational model requirements that are needed to satisfy Equation 4.21. These are: (1) The variational model $V$ should bound the uncertainty within the system model $C_a$, represented by the horizontal axis of the ellipse in Figure 4.3. (2) The variational model should bound downstream component state $x$ illustrated by the vertical axes of the ellipse in Figure 4.3. Downstream component models are component models that receive variational model output either directly or indirectly. (3) The variational model must be within an area that is strictly greater than or strictly less than the actual system behavior in the space defined by the downstream component models. This is illustrated by the dark region at the top and light region at the bottom of Figure 4.3.

**Extension to General System Models**

Although the previous section considers a simple, two-component system model, these conclusions extend directly to both general linear time-invariant system models and system models in general. For example, a linear time-invariant motor-propeller system illustrated in Figure 4.2 (b) can be written

\[
\dot{x} = \begin{bmatrix} F_m & 0 \\ 0 & F_p \end{bmatrix} \begin{bmatrix} x_m \\ x_p \end{bmatrix} + \begin{bmatrix} G_u & G_\tau & 0 \\ 0 & 0 & G_\omega \end{bmatrix} \begin{bmatrix} u \\ V_\tau(y_p) \\ V_\omega(y_m) \end{bmatrix}
\]

\[
y = \begin{bmatrix} H_\tau & 0 \\ 0 & H_\omega \end{bmatrix} \begin{bmatrix} x_\tau \\ x_\omega \end{bmatrix} + \begin{bmatrix} J_u & J_\tau & 0 \\ 0 & 0 & J_\omega \end{bmatrix} \begin{bmatrix} u \\ V_\tau(y_p) \\ V_\omega(y_m) \end{bmatrix}
\]

(4.31)

where the motor and propeller state matrices form a system model block matrix. This results in the same structure as the single component model in Equations 4.24 and 4.25 and can be applied to any structure of linear time-invariant component models. This extends the results in Section 4.2.4 to linear time-invariant system models in general.
In addition to linear systems, applying the analysis in Section 4.2.4 to any discipline specific model defined by Equation 2.2 replaces the constant slope illustrated in Figure 4.3 by some other function \( h \) (Equation 2.2). This changes the location of the solution to Equation 4.16 but does not change the conclusions made in Section 4.2.4, which apply to system models in general.

### 4.2.5 Verification of Variational Model

Because specific variational models are required to bound system error, a set of simulation experiments is used to ensure variational models are appropriate. Section 4.2.4 highlighted three challenges to ensure that a specific variational model is able to bound a specific model error. First, the variational model should bound a specific error source. To do this, an appropriate variational model can be determined for each error source independently. This process is described in Section 4.3 for various component model errors. Second, each variational model should independently bound the system model outputs which have been transformed by downstream component models. This accounts for both downstream model transformations and component model dynamic behavior. This is accomplished by comparing one system model with the measured or actual error injected into the system model to a second variational system model. The variational system model should bound the system model with error injected into the simulation using Equation 4.21 for some variational parameter \( v \). This process is continued for each error source independently. Finally, interactions between separate variational models could excite new behavior within the variational models. This will not be addressed in this chapter. Instead, error sources and variational models will be assumed to be independent from each other. This is a valid assumption for many systems because error sources are most frequently independent. This does not mean that the impact of different error sources on the system model is independent as is frequently assumed in sensitivity analysis. We only assume that the behavior of one error source does not change due to another error source, which is usually a correct assumption.

Ensuring these conditions are satisfied for each variational error model enables the variational system model to bound the actual system behavior as defined in Equation 4.7. The process of establishing these conditions is illustrated in Section 4.2.4.
4.2.6 Method Summary

Section 4.2 proposes a method to quantify system model accuracy by bounding actual system behavior within some offset from the system model (Equation 4.20) based on information available early in design. This enables validation of foundational design decisions as well as validation of a design’s implementation early in system design before system measurements are available.

This method involves (1) developing variational models (Equations 4.3 and 4.5) bounding model uncertainty, (2) independently ensuring each variational model bounds the system behavior based on expected model uncertainty, (3) determining the maximum error based on Equation 4.19, and (4) evaluating if the system response range $y_n \pm \varepsilon_{\text{max}}$ is within design objectives.

This results in constant bounds describing a range of possible responses of the actual system based on a set of input conditions and expected range of model behavior. If this range is within the design objectives, then the system will satisfy these design objectives if the assumptions of this method are correct: known, independent errors. If the range is partially within the system design objectives, either design improvements are needed to improve the system model or model refinements are needed to reduce the model variation. If the range is outside the design objectives, the system design is insufficient to meet design objectives.

4.3 Method Application

This section illustrates and verifies this method for quantifying system model accuracy developed in Section 4.2 using the solar powered unmanned aerial vehicle (UAV) propulsion system illustrated in Figure 3.1. This section computes the maximum error of the propeller thrust using Equation 4.19, providing an envelope within which the actual system behavior should lie.

The process illustrated in this section involves (1) developing models for each component, (2) creating variational models describing model error, (3) composition of these models to create both a system model and variational system model, (4) verification that variational models independently bound system error, and (5) computing the maximum error using Equation 4.19.

The UAV system model shown in Figure 3.1 is segmented into four component models: solar panel, battery, motor (which includes the motor, speed controller, and servo controller), and
propeller. It has two inputs (speed command and battery charge) and produces a single output (thrust). The following section describes these component and variational models.

### 4.3.1 Component and Variational Models

The solar array model plotted in Figure 4.4 receives a voltage input and produces a current output shown as a solid line [91]. An additive uncertainty model (Figure 3.1 (b) dashed line)

\[ V_s(y_s, v) = y_s + W_1 v W_2 \]  

(4.32)

bounds the measured solar array data shown as dots in Figure 4.4. The solar irradiation and temperature are assumed to be constant.

![Solar Cells Model with Max Error Window Overlaid on Experimental Data](image)

Figure 4.4: Solar Cells Model with Max Error Window Overlaid on Experimental Data

The battery model is a nonlinear system of differential equations that produces an output voltage dependent on the battery charge and current [93,94]. Figure 4.5 (a) shows the measured and
simulated voltages of one battery discharge test. Figure 4.5 (b) shows the magnitude of the voltage error between the model and measured voltage (dashed line). The additive uncertainty described in Equation 4.32 bounds both static and dynamic errors (Figure 4.5 (b) solid line at 0.08v).

Figure 4.5: Measured Data vs. Battery Model of Battery Voltage During Discharge
The motor component model encompasses the servo controller, speed controller, and servo motor. It is a piecewise linear system of differential equations producing torque and current outputs based on command and voltage inputs [65]. Figure 4.6 (a) shows the simulated (solid line) and measured (dots) current. Motor error (Figure 4.6 (b) solid line) is large when motor speed changes but close to 0 elsewhere. This large error is due to the simplified piecewise linear motor model not containing the current limits present in the actual motor controller. Therefore, motor error is modeled as the linear transfer function

\[
i_v(i,v) = i + vL^{-1}\left(\frac{s + k_1}{s + k_2}\right) \cdot L(i)
\]

\[
\omega_v(i,v) = \omega + vL^{-1}\left(\frac{s\omega + k_3}{s\omega + k_4}\right) \cdot L(\omega) \cdot (k_4\omega + k_5)
\]

where \(L\) is the Laplace transform, \(L^{-1}\) is the inverse Laplace transform, and \(k_i\) are constants, \(v\) is variational parameters, \(s\) are Laplace variables, \(i\) is the motor current, and \(\omega\) is motor angular velocity. Dynamic variational models do not present any difficulty in the system model since they are designed for dynamic system analysis.

The propeller model converts angular velocity to thrust and torque [97]. The propeller thrust error model

\[
T_t(y_t, v_1, v_2) = y_t + W_1v_1W_2 + (W_3v_2W_4)y_t
\]

combines both additive and multiplicative uncertainty models. Figure 4.7 shows the propeller thrust model as a solid line, measured propeller response as dots, and max and min of the propeller model as a dashed line. The variational model bounds the range of propeller measurements as specified in Section 4.2.6.

### 4.3.2 System Model and Variational System Model Development

Now that component models are available, the system model is developed by composition of component models and described by the system model graph in Figure 3.1.

The error augmented system model is next created by starting with the system model. The variational models are then added to the system model and variational parameters become system model inputs. For example, the solar array variational model is included between the solar
array model and current sum model. The process of adding variational models continues until all variational models from Section 4.3.1 have been added to the system model.

The error augmented system model can be evaluated in the same method as the system model. It allows model permutations to be evaluated by adjusting the variational parameters $v$.  

Figure 4.6: Measured vs. Modeled Motor Speed and Current Over a Step in Supply Voltage
4.3.3 Variational Model Verification

In Section 4.3.1, each variational model was verified to bound actual system behavior. When this process is completed, we next ensure that each variational model bounds the system model output as some $v$. We do this by injecting the measured error in the system model and ensuring that Equation 4.21 is satisfied for some $v$. This was accomplished by testing that either $v = 1$ or $v = -1$ satisfied Equation 4.21 which was the case for each error source and variational system model. The variational models illustrated in Section 4.3.1 were sufficient to bound the system model output.

4.3.4 Max Error Computation and Method Validation

With variational models validated within the system model, the final step is to solve Equation 4.19 to determine the maximum error. This results in $\epsilon_{\text{max}} = 0.095719$ at $e = [-1.0, -1.0, 1.0, -1.0, -1.0]$. Figure 4.8 plots $S \pm \epsilon_{\text{max}}$ as dashed lines, measured system data as points, and $S_v$ as solid lines. The
region $S \pm \varepsilon_{\text{max}}$ can be interpreted as error bounds. Similar to a control chart in statistical process control, actual system behavior is expected to lie within these error bounds.

The assurance that system behavior is contained within $S \pm \varepsilon_{\text{max}}$ enables design objectives to be tested against this range of behavior. For example, if the propeller thrust should be greater than 0.1 lbs, Figure 4.8 shows that the design is acceptable. If, however, our design objective is that thrust should be greater than 0.20 lbs, our design would be unacceptable although the nominal model thrust is greater than 0.20 lbs at the high command. The maximum error bounds show that some model conditions will not achieve 0.20 lbs thrust.

Figure 4.8: Max Error Results Graph Showing Measured Data and Model Prediction with Worst-Case Error Bounds
Figure 4.8 shows that the accuracy of a system model can be bounded based on component error models and known system model structure. Because measured system behavior is not used to bound the model error, this method can be used to validate system models early in system design.

We would like to highlight that maximum error provides a very conservative measure of system model error. To illustrate this, in Figure 4.8, the maximum error results from a transient spike in predicted thrust. The difference in steady-state thrust between the system model and variational system model is much smaller. If the application is interested in steady-state, rather than dynamic behavior, this can be taken into account as the variational models are developed.

4.4 Summary

This chapter has proposed and demonstrated a method that is able to quantify system model accuracy before system behavior is known. This enables system design decisions to be validated early in system design. This method requires developing variational models to describe known errors, creating a system model and variational system model, and finding the maximum error using these models. Maximum error then defines a region around the system model containing the actual system output. This method does not account for unknown system errors or error interactions.
CHAPTER 5. CONCLUDING REMARKS

5.1 Summary

Although modeling is an integral part of engineering design, the modeling of multidisciplinary systems remains a challenge in system design today. This dissertation focused on three areas to expand the current capabilities of system modeling: system model development, system model feasibility, and system model accuracy.

Chapter 2 developed an approach for predicting system behavior by composition of existing discipline specific models. This involved establishing a mathematical formulation for developing compositional system models from existing discipline specific models and verifying that these system models are able to model engineering system behavior.

Chapter 3 developed a formulation for system model feasibility, defined as the region where a system model is able to produce valid results, and developed a design space exploration algorithm to identify this region. Feasibility is especially important for compositional system models due to their additional sources of failure. This algorithm enables developers to (1) determine whether solutions to specific simulation conditions exist, (2) identify if changes in a system model affect feasibility, (3) identify significant sources of system model failures, and (4) select system model improvements that will lead to feasibility improvements.

Chapter 4 developed a method to bound system model accuracy early in system design. This enables system design decisions to be verified at the point when they are made and while there is still design flexibility.

These contributions enable design organizations to build upon their existing expertise to develop system models and effectively use these models to validate design decisions early in system design. This helps to streamline the long and iterative design-prototype-redesign cycle that characterizes system design today.
5.2 Conclusions

The foundation developed in this dissertation enables the existing discipline specific models within an organization, that are currently used to model individual components, to be combined to model system behavior. With this large population of trusted models, system models can be quickly developed to analyze system behavior and to evaluate system interactions. Because new systems often share many components with previous systems, previous component and system models provide a starting point and analysis tool for future system designs. In addition, compositional system models can be quickly reconfigured and altered to explore other design concepts both manually and by automated design synthesis.

The design of complex multidisciplinary systems is one of the significant engineering challenges of our day. This dissertation has addressed a key limitation of the system design process: being unable to predict the results of design decisions at the point these decisions are made. Without this ability, the current practice in system design has been to perform small incremental changes on known system designs coupled with prototype verification. This approach becomes increasingly impractical as system complexity, design times, and design constraints increase.

Combining this system modeling formulation with quantified system model feasibility enables reliable use of system models even in a dynamic system design environment. Feasibility provides a new understanding about not only how a system model can be used, but also the limitations of a system model. By examining a feasible domain, a designer can immediately determine if the model is feasible for the needed range of simulation conditions. If it is found that a model’s required working domain is outside of its feasible domain, the unpredictable behavior commonly encountered when modeling system behavior can be avoided by allowing designers to change either the system model or simulation domain before simulation begins. If, instead, the model is feasible over the needed range of system behavior, the feasibility metric still provides valuable, ongoing feedback about the impact of design change on the system model.

Regardless of these benefits of system modeling and system model feasibility, if system model accuracy cannot be quantified, system models cannot be used to validate design decisions. This new method presented in this dissertation of quantifying accuracy provides this capability early in system design, something not possible before. Although dependent on the assumptions of
known and independent system model error, this method provides a capability that is not otherwise
available to system models.

These three aspects of this research – developing system models from existing discipline
specific models, understanding system model feasibility, and quantifying system model accuracy
– result in a much more capable, reliable system design process.

5.3 Future Work

This dissertation has provided a foundation for additional research and development into
system modeling leading ultimately to high level system design synthesis and system design au-
tomation.

Our immediate focus is to apply this theory to a wider range of commercial and aca-
demic systems models. This will grow our knowledge of this theory and identify its limitations. Some specific areas of interest are human-in-the-loop simulations, hierarchical simulations, and distributed simulations.

A practical aspect of growing this work is growing the number and type of component mod-
els immediately usable by system models. To achieve this, the model composition library discussed in Section 2.3.1 has been released as an open-source library at http://sourceforge.net/p/modellink [132]. Growing the capabilities of this library will make the theory developed in this dissertation more attainable.

Future work into system model feasibility includes extending feasibility exploration to sta-
tistical failures, quantifying the relationship between computational cost and the feasibility space dimension, classifying failure importance based on boundary size, and quantifying the sensitivity of the feasible domain to design changes.

Future work into system model accuracy involves investigating statistical and geometric
methods of quantifying system model error. The maximum error method presented in Chapter 4, which benefits from developments in system theory, is a very conservative measure.

These foundational developments enable future efforts into determining appropriate high
level system design languages. Several limitations to the graph theory representation of system models were highlighted in Section 2.2.2. These languages will facilitate both advanced system model development and system design automation.
REFERENCES


88


[38] Board, I. S., 2000. IEEE standard for modeling and simulation (M&S) high level architecture (HLA) object model template (OMT) specification Sept. 5, 6


90


93


[92] ST, 2011. Designing with SPV1020, interleaved boost converter with MPPT algorithm Application Note AN3392, STMicroelectronics, Geneva, Switzerland. 34


94


96
