FUNCTION-BASED DESIGN TOOLS FOR ANALYZING THE BEHAVIOR AND SENSITIVITY OF COMPLEX SYSTEMS DURING CONCEPTUAL DESIGN

A Dissertation

by

RYAN SCOTT HUTCHESON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2009

Major Subject: Mechanical Engineering

FUNCTION-BASED DESIGN TOOLS FOR ANALYZING THE BEHAVIOR AND SENSITIVITY OF COMPLEX SYSTEMS DURING CONCEPTUAL DESIGN

A Dissertation

by

RYAN SCOTT HUTCHESON

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2009

Major Subject: Mechanical Engineering

ABSTRACT

Function-based Design Tools for Analyzing the Behavior and Sensitivity
of Complex Systems During Conceptual Design. (May 2009)

Ryan Scott Hutcheson, B.S., Missouri University of Science and Technology;
M.S., Missouri University of Science and Technology;
Chair of Advisory Committee: Dr. Daniel McAdams

Complex engineering systems involve large numbers of functional elements. Each functional element can exhibit complex behavior itself. Ensuring the ability of such systems to meet the customer's needs and requirements requires modeling the behavior of these systems. Behavioral modeling allows a quantitative assessment of the ability of a system to meet specific requirements. However, modeling the behavior of complex systems is difficult due to the complexity of the elements involved and more importantly the complexity of these elements' interactions.

In prior work, formal functional modeling techniques have been applied as a means of performing a qualitative decomposition of systems to ensure that needs and requirements are addressed by the functional elements of the system. Extending this functional decomposition to a quantitative representation of the behavior of a system represents a significant opportunity to improve the design process of complex systems.

To this end, a functionality-based behavioral modeling framework is proposed along with a sensitivity analysis method to support the design process of complex systems. These design tools have been implemented in a computational framework and have been used to model the behavior of various engineering systems to demonstrate their maturity, application and effectiveness. The most significant result is a multi-fidelity model of a hybrid internal combustion-electric racecar powertrain that enabled a comprehensive quantitative study of longitudinal vehicle performance during various stages in the design process. This model was developed using the functionality-based framework and allowed a thorough exploration of the design space at various levels of fidelity. The functionality-based sensitivity analysis implemented along with the behavioral modeling approach provides measures similar to a variance-based approach with a computation burden of a local approach. The use of a functional decomposition in both the behavioral modeling and sensitivity analysis significantly contributes to the flexibility of the models and their application in current and future design efforts. This contribution was demonstrated in the application of the model to the 2009 Texas A&M Formula Hybrid powertrain design.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

LIST OF TABLES

1. INTRODUCTION: THE DESIGN OF COMPLEX SYSTEMS

As developments in technology have matured, the task of designing engineering systems has become more difficult. The design of a modern engineering system includes complex multidisciplinary problems, multi-level optimization and shorter design cycles. As a result, engineering designers have been increasingly relying on structured design processes along with model-based design techniques as a means of facilitating and organizing the design process of systems. Within both the Systems Engineering [1, 2] and Design Theory communities [3-6], model-based approaches to design have been proposed as a means of developing, representing, storing and communicating critical information about a system, its desired functionality and the various potential solutions to its functionality.

During conceptual design, the use of quantitative models to assist concept generation and selection has become more prevalent. These models, often referred to as behavioral models, describe the performance of a concept relative to a set of target metrics and are used to evaluate the concept's ability to meet requirements and customer needs. For example, a dynamic drivetrain model may be used during the design of a hybrid automobile in order to select an appropriate drivetrain configuration.

Unfortunately, the actual process used to create such models during conceptual design is often not well explained in design literature. For example, the Systems Engineering and Analysis text [2] used in engineering design courses at several universities (including Texas A&M) clearly demonstrates the need for creating behavioral models during design but does not present a process for how to actually implement them within the design process or how to create them using the physical modeling skills learned in traditional engineering courses (statics, linear systems, etc.). [1]

This dissertation follows the style of the Journal of Mechanical Design.

In practice, these models are made once a concept has been fully developed and all form solutions to desired functionality are known. Recent developments in component-based modeling have facilitated this model definition process by associating model elements with specific components [7-9]. This approach promotes model re-use and allows multiple combinations of components to be investigated. The basic modeling process used in this approach is as follows:

1. Describe the desired functionality of the system,
2. Find component solutions for functions,
3. Combine solutions into concepts,
4. Model the components (or re-use model elements),
5. Combine the elements to produce a complete system model,
6. Evaluate the concept using the model,
7. Manually restart the process at Step 2 until a satisfactory set of concepts has been found.

The first two steps are generally espoused in design engineering texts [1-6] but not in the context of behavioral model creation. However, for such component-based approaches to be used, a full set of components must be established before the system can be modeled and a function-based approach (Steps 1 and 2) is generally recommended to find these components. Additionally, manual modeling and model assembly work is required in order to develop behavioral models for each concept once components have been identified. While component-based modeling approaches promote model re-use, it is only at the component level.

Creating behavioral models based on the functionality of a system presents several opportunities to improve current system design practices during early design. Three specific tools that utilize the function-based behavioral modeling approach were developed and are presented in this work. These tools include a quantitative behavioral analysis based on a formal functional decomposition, a function-based sensitivity analysis along with new sensitivity metrics and a software implementation of the behavioral and sensitivity modeling techniques.

1.1. OBJECTIVES OF THE RESEARCH

To overcome the limitations of traditional component-based modeling approaches, it is proposed that recent developments in functional representations [10] along with precedents found in object-oriented programming [11] be used to re-structure the behavioral modeling process of systems during the early design phase. Specific objectives of this restructuring include:

- Integration with activities currently performed in the early design process,
- A broad solution domain and range of applications,
- Promotion of model reuse at the functional level along with the solution family and component levels,
- Multi-fidelity modeling,
- A computational implementation that utilizes modern programming philosophies.

1.2. FOUNDATIONS OF THE SOLUTION – FUNCTIONAL MODELING

Functional models are graphical tools for representing the routing and transformation of energy, material and signal flows through a system. Traditionally, functional models have been created informally as schematics, block-diagrams and function structures. Recently, significant research has been performed to streamline and formalize the process of creating and representing functional models [10]. This research has enabled the development of several function-based design tools aimed at improving the design process in all phases (from product planning to detailed design). Specifically, several tools have been proposed that use the functions in a functional model as a starting point for component identification [12, 13]. In general, these tools use qualitative means for selecting valid component combinations such as historic compatibility and failure rates. Additionally, attempts have been made to create behavioral models based on these functional models in order to provide a quantitative means of performing concept identification and selection [14]. However, these methods are generally limited in scope with respect to model representation and system complexity.

The implementation of such an approach has a precedent in the transition in software development from procedural programming to object-oriented software design. Essentially, behavioral modeling currently resembles the approach used in procedural programming. The object in procedural programming, as described by Bjarne Stroustrup (the creator of the C++ programming language) in [11] is to "decide which procedures you want" then "use the best algorithms you can find." Such programming practices have a strong focus on the process the algorithms employ rather than the objects they are acting on. This style of programming is analogous to the concept of building behavioral models for a system once the components of that concept have been completely identified.

Modular programming is described as "decide which modules you want" then "partition the program so that data is hidden within modules" [11]. This paradigm focuses more on the "organization of data" [11] instead of the "design of procedures" [11] and relates to the practice of using component-based modeling during the design of systems in that modules are created to contain the data related to specific components and then combined to produce a complete model.

Object-oriented programming is described as "decide which classes you want; provide a full set of operations for each class" and then "make commonality explicit by using inheritance" [11]. Inheritance provide a means for describing the behavior of an object (using a class) then allowing this description to be modified by creating new classes that inherit the interface and base attributes of the parent class. This concept provides a precedent for using a functional model to guide the behavioral modeling process of systems. Essentially, in such an approach a class (functionality) is used to describe an aspect of a system and then specific solutions to this functionality are described with classes that inherit from the parent functionality and include the parent's interface (boundary flows in the functional model). As stated in [15], the object-oriented "paradigm closely reflects the structure of systems in the real world and is therefore well suited to model complex systems with complex behavior." Many of the concepts used in object-oriented programming relate to concepts in the behavioral modeling world including:

- Mapping between functions (in the design domain) and classes (in the programming domain),
- Mapping between flows (in the design domain) and messages (in the programming domain),
- Mapping between behavioral models (in the design domain) and methods (in the programming domain),
- Mapping between parameters (in the design domain) and attributes (in the programming domain),
- Polymorphism (inherited classes overriding the behavior of their parent classes while using the same interface), which naturally enables multi-fidelity modeling,
- Inheritance and instancing as it relates to the mapping of the concept of functions, solution families for functions and solution instances,
- Encapsulation (hiding class attributes from other non-related classes) and the hiding internal parameters for a solution to a function from neighboring functions that do not physically have access to this information,
- Abstraction (inheriting classes to the level required by the program) and modeling functions at the appropriate level of abstraction required for the behavioral model.

To this end, object-oriented behavioral modeling approaches (Modelica most significantly) have been created. However, these approaches fail to recognize the role of functionality as the highest level in the model hierarchy. By leaving this aspect out of the models, such approaches limit their application in the conceptual design stage of systems engineering when components and solutions are not fixed. As a result, there is an opportunity to develop a comprehensive, function-based modeling paradigm that includes the beneficial aspects of object-oriented programming as well as current systems modeling practices. The result of this research is such an approach.

1.3. PROCEDURE

To develop this approach, modern design methods and Systems Engineering practices were investigated to produce a clear picture of the recommended activities during early design (Section 2). The ability of current component-based modeling methods to assist these activities was then investigated. Next, various model representation formats were identified for use in the method. The space between a conceptual design and the mathematical definition of a concept's performance was then investigated and a design framework that bridges this gap was developed (Section 3). The framework was then applied to the development of an appropriate sensitivity analysis method (Section 4) and a functioning software implementation was made (Section 5). A series of examples were completed to demonstrate the application of the method and appear through the work. A comprehensive example of the approach appears in Section 6. The work is summarized and concluded in Section 7. Terminology used throughout the work is defined next.

1.4. TERMINOLOGY

1.4.1. Early Design

Formal design processes are prescribed in virtually every modern engineering design textbook. Each proposed method differs from the next but in general the same basic set of design activities and general sequence is promoted. In general, the proposed design methods include four basic stages:

> **Product Planning** – Determining what product to make and what attributes it should have,
>
> **Conceptual Design** – Identifying desired functionality and potential physical solutions to the desired functionality. Combining these solutions to develop concepts,
>
> **Embodiment Design** – Translating a complete design into a description that can be used to create actual physical implementations of the design,

Detailed Design – Refining, detailing and analyzing selected concepts to make sure they will be capable of satisfying requirements.

These four stages of design are explicitly described in several engineering design texts and can be abstracted from the proposed processes in others. Within the early stage of design (product planning and conceptual design), behavioral models are recommended as a means to facilitate the evaluation of concepts. In order to develop a new approach for creating and managing behavioral models, the role these models play in the design process must be completely understood. Section 2 presents a comprehensive look at the early design process of systems.

1.4.2. Behavioral Modeling

Several tools have been developed to assist the process of creating and managing behavioral models for systems. These include component-based modeling languages and implementations such as Modelica [7], Simscape [8] and 20-Sim [9] as well as general systems modeling languages such as SysML [16]. Additionally, there have been attempts to adapt component-based modeling to conceptual design including the Schemebuilder project outlined in [14]. Generally, current modeling tools and languages fail to include the explicit definition of system functionality and as such are ill-suited for use in conceptual design. SysML includes functional models based on the Enhanced Function-Flow Block Diagram [1, 2] approach but does not include a provision for implementing complex behavioral models based on this functional model (simple behavioral model are specified as a separate model class). Often, prior attempts at developing behavioral modeling tools for use in conceptual design immediately limit their applicability by restricting the models to a specific domain (Bond Graphs [17] in the case of Schemebuilder [14]). The newly proposed modeling approach (Section 3) was developed with these factors in consideration and provides an approach that utilizes the strengths of existing tools while overcoming their limitations.

1.5. COMPREHENSIVE EXAMPLE

To assess the ability of the method to meet its pre-defined objectives, a sufficiently complex example was conducted using the method and the various tools developed that implement the method. The system selected for this example was a Formula Hybrid racecar. The specific aspect of the system explored in the example was the early design process with a focus on powertrain development. This system was selected due to its complexity, relevance to modern engineering challenges and the ability to work with Texas A&M's 2009 Formula Hybrid team.

The developed modeling framework was applied to this system in order to explore the performance and sensitivity of various solutions to the required functionality of the system. The ability of the method to promote the generation and exploration of a variety of concepts in a timely matter was benchmarked relative to current practices in the design of hybrid vehicles [18-20]. The result of the example is a set of feasible concepts for the system and provided to Texas A&M's Formula Hybrid team along with the tools used to develop and model the concepts. Thus, in additional to validating the modeling approach the chosen example provides a real, tangible benefit to the local engineering community. This example appears in its entirety in Section 6.

2. PRODUCT DESIGN

Formal design processes are prescribed in virtually every modern engineering design textbook [1-6, 21, 22]. Each proposed method differs from the next but in general the same basic set of design activities and general sequence is promoted. In general, the proposed design methods include four basic stages:

Product Planning – Determining what product to make and what attributes it should have,

Conceptual Design – Identifying desired functionality and potential physical solutions to the desired functionality and combining these solutions to develop concepts,

Embodiment Design – Translating a complete design into a description that can be used to create actual physical implementations of the design.

Detailed Design – Refining, detailing and analyzing selected concepts to make sure they will be capable of satisfying requirements,

These four stages of design are explicitly described in several engineering design texts [3, 4, 21, 22] and can be abstracted from the proposed processes in others [5,6]. Table 2.1 includes several popular engineering design texts along with the highest-level proposed stages of engineering design.

Table 2.1. Engineering Design Process Stages

Source	Design Stage
Cross	Exploration
	Generation
	Evaluation
	Communication
Suh	Customer Domain
	Functional Domain
	Physical Domain
	Process Domain
Blanchard and Fabrycky	Conceptual Design
	Preliminary Design
	Detailed Design
Ulrich and Eppinger	Concept Development
	System-level design
	Detail Design
	Testing and Refinement

Table 2.1 Continued.

Source	Design Stage
Pahl and Beitz	Planning and Clarifying the Task
	Conceptual Design
	Embodiment Design
	Detailed Design
Ullman	Specification development/planning
	Conceptual Design
	Product Design
Otto and Wood	Understand the opportunity
	Develop a Concept
	Implement a Concept

2.1. EARLY ENGINEERING DESIGN

For the remainder of this work, the phrase "early engineering design process" will refer to the first two general stages of a structured design process: *product planning* and *conceptual design*. In general, at the completion of the early engineering design process a number of potential concepts have been developed that are predicted to be able to meet the customer's needs and system requirements as well as the design firm's own expectations. As with the overall design process, engineering design texts propose basically the same set activities and sequence during early engineering design. The remainder of this section presents an outline of various early engineering design activities as proposed by several popular engineering design texts. These activities are then reconciled to produce a complete early engineering design process for use in the development of the function-based modeling tools and the comprehensive example.

2.1.1. Cross

In *Engineering Design Methods*, Cross [21] defines three basic activities for the design process including exploration, generation and evaluation. All three activities occur through the design of a product but in general they can be mapped to the two standard phases of early design in the follow manner:

Product planning – exploring the market, generating customer needs and requirements, evaluating your firm's position to complete the design process,

Conceptual design – generating functional requirements, exploring solutions for functionality, generating concepts, evaluating concepts.

The specific activities recommend by Cross along with a mapping to the type of activity appear in Table 2.2.

Table 2.2. Cross's Design Activities

Activity	Stage
Clarifying objectives	Exploration
Establishing functions	Exploration
Setting requirements	Exploration
Determining characteristics	Exploration
Generating alternatives	Generation
Evaluating alternatives	Evaluation
Improving details	Exploration, Generation and Evaluation

2.1.2. Suh

Rather than describe engineering design as a process, in *Axiomatic Design*, Suh [22] identifies various domains in which an engineering designer functions. For an engineering designer operating in the early design phase, three basic domains are identified including customer, functional and physical. These domains along with characteristics of the domains as they relate to early design appear in Table 2.3.

Table 2.3. Characteristics of Early Design

Characteristic	Domain
Attributes that customers desire	Customer
Desired performance	Customer
Customer satisfaction	Customer
Attributes desired of the overall system	Customer
Functional requirements specified	Functional
Required properties	Functional
Physical variables for functional requirements	Physical
Components	Physical

In order to reconcile the various product design methods, Suh's characteristics were mapped to appropriate design activities. This mapping was completed by identifying activities from other design processes and associating them to the listed characteristics. The results of this mapping process appear in Table 2.4.

Table 2.4. Mapping Characteristics to Activities

Characteristic	Mapping to activity
Attributes that customers desire	Find attributes customer's desire
Desired performance	Determine desired performance
Customer satisfaction	Develop metrics to ensure customer satisfaction
Attributes desired of the overall system	Determine overall system requirements
Functional requirements specified	Determine functional requirements
Required properties	Develop a set of desired physical attributes
Physical variables for functional requirements	Define performance metrics
Components	Determine components that satisfy functions

2.1.3. Ulrich and Eppinger

The early design process is described as a single concept development stage by Ulrich and Eppinger in *Product Design and Development* [3]. In this all-inclusive early stage of design, several activities are proposed. These activities are outlined in Table 2.5.

Table 2.5. Early Design Activities

Activity	Stage
Identify customer needs	Concept Development
Establish target specifications	Concept Development
Analysis of competitive products	Concept Development
Concept generation	Concept Development
Concept selection	Concept Development
Refinement of specifications	Concept Development
Economic analysis	Concept Development
Project planning	Concept Development

2.1.4. Pahl and Beitz

Engineering Design by Pahl and Beitz [4], long considered a staple of modern product design, defines the four major stages of product development explicitly. The specific activities proposed by Pahl and Beitz for the early design process appear in Table 2.6.

Table 2.6. Early Design Activities Recommended by Pahl and Beitz

Activity	Stage
Analyze market and company position	Planning and Clarifying the Task
Find and select product ideas	Planning and Clarifying the Task
Formulate a product proposal	Planning and Clarifying the Task
Clarify the task	Planning and Clarifying the Task
Elaborate a requirements list	Planning and Clarifying the Task
Identify essential problems	Conceptual Design
Establish function structures	Conceptual Design
Search for working principles and working structures	Conceptual Design
Combine and firm up into concept variants	Conceptual Design
Evaluate against technical and economic criteria	Conceptual Design

2.1.5. Ullman

In *The Mechanical Design Process* by Ullman [5], the early engineering design process can be broken down into two stages: specification development/planning and conceptual design. These stages directly translate to the previously described product planning and conceptual design stages. A list of Ullman's proposed activities for these stages appears in Table 2.7.

Table 2.7. Activities in Ullman's Early Design Process

Activity	Stage
Developing customer requirements	Specification Development/Planning
Assessing the competition	Specification Development/Planning
Generating engineering requirements	Specification Development/Planning
Establishing engineering targets	Specification Development/Planning
Planning for design	Specification Development/Planning
Functional decomposition	Conceptual Design
Generating concepts from functions	Conceptual Design
Judging feasibility	Conceptual Design
Assessing technology readiness	Conceptual Design
Go/no-go screening	Conceptual Design
Using the decision matrix	Conceptual Design

2.1.6. Otto and Wood

Otto and Wood [6] summarize the early engineering design process in two steps as well in *Product Design*: understanding the opportunity and developing a concept. Once again, these two stages correspond identically to the product planning and conceptual design stages previously described. The specific activities proposed for these two stages appear in Table 2.8.

Table 2.8. Otto and Wood Early Design Activities

Activity	Stage
Develop a vision	Understand the Opportunity
Market Opportunity Analysis	Understand the Opportunity
Customer Need Analysis	Understand the Opportunity
Competitive Analysis	Understand the Opportunity
Portfolio Planning	Develop a Concept
Functional Modeling	Develop a Concept
Product Architecture Development	Develop a Concept
Concept Engineering	Develop a Concept

2.2. RECONCILING THE VARIOUS METHODS

To create a comprehensive summary of early design, the proposed activities outlined in the previous section were reconciled. To perform this reconciliation, the activities proposed by the various authors as previously described were combined into a single list of activities. These activities were then sorted in general temporal order and grouped by commonality. The resulting complete list appears in Table 2.9.

Table 2.9. Combined List of Design Activities

#	Activity	Source
1	Develop a vision	Otto and Wood
2	Analyze market and company position	Pahl and Beitz
3	Market opportunity analysis	Otto and Wood
4	Find and select product ideas	Pahl and Beitz
5	Formulate a product proposal	Pahl and Beitz
6	Clarify the task	Pahl and Beitz
7	Clarifying objectives	Cross
8	Customer need Analysis	Otto and Wood
9	Developing customer requirements	Ullman
10	Identify customer needs	Ulrich and Eppinger
11	Find attributes customer's desire	Suh
12	Assessing the competition	Ullman
13	Competitive analysis	Otto and Wood
14	Analysis of competitive products	Ulrich and Eppinger
15	Elaborate a requirements list	Pahl and Beitz
16	Generating engineering requirements	Ullman
17	Establishing engineering targets	Ullman
18	Develop a set of desired physical attributes	Suh
19	Define performance metrics	Suh
20	Establish target specifications	Ulrich and Eppinger
21	Determine desired performance	Suh
22	Develop metrics to ensure customer satisfaction	Suh
23	Determine overall system requirements	Suh
24	Identify essential problems	Pahl and Beitz
25	Planning for design	Ullman
26	Portfolio Planning	Otto and Wood
27	Functional Modeling	Otto and Wood
28	Functional decomposition	Ullman
29	Establish function structures	Pahl and Beitz
30	Establishing functions	Cross
31	Determine functional requirements	Suh
32	Setting requirements	Cross
33	Determining characteristics	Cross
34	Product Architecture Development	Otto and Wood
35	Concept Engineering	Otto and Wood
36	Generating concepts from functions	Ullman
37	Search for working principles and working structures	Pahl and Beitz
38	Combine and firm up into concept variants	Pahl and Beitz
39	Concept generation	Ulrich and Eppinger
40	Identify working mechanisms	Suh
41	Determine components that satisfy functions	Suh
42	Generating alternatives	Cross
43	Evaluating alternatives	Cross
44	Concept selection	Ulrich and Eppinger
45	Evaluate against technical and economic criteria	Pahl and Beitz
46	Judging feasibility	Ullman

Table 2.9 Continued.

#	Activity	Source
47	Assessing technology readiness	Ullman
48	Go/no-go screening	Ullman
49	Using the decision matrix	Ullman
50	Improving details	Cross
51	Refinement of specifications	Ulrich and Eppinger
52	Economic analysis	Ulrich and Eppinger
53	Project planning	Ulrich and Eppinger

Activities that accomplish the same basic tasks in the design process were then grouped together and re-labeled to create a condensed list that includes all of the activities proposed in the various engineering texts. The result of this reconciliation is the proposed early design process that appears in Table 2.10. The grouping of activities from the various authors is included as well. This reconciled process represents a comprehensive approach for completing the early design of a product. Each step in this process will be outlined in the next section.

Table 2.10. Reconciled Early Design Process

Stage	Reconciled Activity	Original Activity
Product Planning	Develop a vision	1
	Analyze the market for opportunities	2,3,4
	Formulate a product proposal with objectives	4,5,6,7
	Perform a comprehensive assessment of customer needs	8,9,10,11
	Compare needs to competitive products	12,13,14
	Translate needs to engineering requirements	15,16,23
	Develop performance metrics for requirements	17,18,19
	Set target values for metrics to ensure that needs are met	20,21,22
	Identify specific design challenges	24
	Plan for the remainder of the design process	25,26
Conceptual Design	Develop functional requirements	31,32,33
	Develop a conceptual functional model of the product	27,28,29,30
	Generate functional alternatives using the functional model	34
	Identify component solutions to functions	35,36,37,41,43
	Combine components to produce concepts	38,39,40,42
	Estimate performance of concepts relative to target metrics	43,44
	Select a set of concepts for further consideration	45,46,47,48,49
	Refine system and functional requirements	51
	Finalize concepts	50
	Develop plan for embodiment and detailed design of concepts	52,53

2.3. PRODUCT PLANNING

2.3.1. Developing a Vision

In general, developing a vision includes identifying products that are not currently being offered in the market and assessing why these products are not being produced or why current products fail to satisfy certain needs [6]. In certain cases, the vision is already prescribed to the designer by management or is a result of other outside influences. Once a vision for a product has been achieved, the next step is to assess whether continuing a design process around the vision is a viable option.

2.3.2. Analyze the Market for Opportunities

In a business environment, assessing the viability of a new product requires an estimation of the potential revenue that can be extracted from the market [6]. This includes both an estimate on price as well as market volume. From the product of price and volume, profit may be deducted resulting in an estimate of cost limits for producing the product.

Pahl and Beitz describe a structured approach to completing both the development of a vision and market analysis. This approach includes the following general steps [4]:

1) Analyzing the current situation of the company and its products (a situation analysis),
2) Formulating search strategies using the strengths and weaknesses of the company,
3) Finding product ideas,
4) Selecting product ideas,
5) Defining products by elaborating on the ideas.

The analysis of the market is embedded in the fourth step, selecting product ideas. In this step, price, volume and profit concerns are used to assess which product ideas to pursue further.

2.3.3. Formulate a Product Proposal with Objectives

At the conclusions of the two prior steps, a product proposal with associated objectives should be completed. In general, a product proposal should include the highest-level desired functionality of the product along with a preliminary set of requirements and objectives. Requirements should be listed in a solution-neutral way [4] (i.e. the requirement should not prescribe a specific set of physical solutions to desired functionality). Cost targets and a basic budget should be included as well [4]. Tools such as an objectives tree can be used in formulating this proposal to outline and order specific objectives [21].

2.3.4. Perform a Comprehensive Assessment of Customer Needs

Once a proposal has been created for a product, a comprehensive assessment of the needs of the customer must be performed. The high-level requirements and objectives previously developed should be used as a guide for performing this assessment. Often this step in the design process is described as listening to and interpreting the "voice of the customer" [6]. The most essential aspect of this step is to ascertain the desires of the customer and to isolate those desires from the desires of the engineers developing the product [5]. Otherwise, the result of the design process may be a product that the engineers love but the customer, and hence the market, abhors.

This activity is often a source of problems for the engineering designer. Potential problems include customers not being aware of the specific item their opinion is being solicited for and obtaining only information about what customers don't like about current products rather than what they do like [6].

To determine and manage customer needs, tools have been developed including QFD and the device used to represent the activities used in QFD: The "House of Quality" [5, 6]. Essential early steps in QFD include identifying the customer, determining customer requirements and defining relative importance of requirements. The remainder of the steps in QFD occur later in the reconciled design process.

2.3.5. Compare Needs to Competitive Products

The next step in QFD, and the reconciled design process, is to compare the generated set of customer needs to competitive products in the market [5]. In general, to complete this process a table of customer needs along with importances is developed and competitor products are qualitatively assessed in their ability to satisfy these needs [3]. For a quantitative analysis, engineering requirements and performance metrics are required.

2.3.6. Translate Needs to Engineering Requirements

In most cases, the customer is not an engineer and does not think like one. To formally represent the objectives of the design process, customer needs must be translated into engineering requirements. Recent work in the field of Systems Engineering has led to excellent descriptions of the process of identifying and managing requirements. At the highest level, requirements should be set to scope:

- The product’s mission,
- Performance and physical parameters,
- Operational deployment or distribution,
- Operational life-cycle,
- Utilization requirements,
- Effectiveness factors,
- The environment in which the product will operate.

These factors are described by Blanchard and Fabrycky in *Systems Engineering and Analysis* [2]. The essential process of identifying requirements is to take each customer need and identify as many measures as possible that assess the ability to satisfy that need [5]. These measures often take on numerical values, which leads to the next step in the reconciled product design process.

2.3.7. Develop Performance Metrics for Requirements

For each requirement, a performance metric should be created in order to quantify the satisfaction of the requirement and hence ensure the underlying customer need is met. Once again, Systems Engineering offers useful insight into quantifying requirements. Technical Performance Measures (TPMs) are "measures of the attribute and/or characteristics which are inherent within the design" [2]. These TPMs assess a product's ability to meet requirements and often form contradictory sets [2]. For example, a performance hybrid automobile must be able to complete quick lap times (a TPM of lap time in seconds) but also must be fuel efficient (a TPM of fuel economy in kilometers per unit of fuel). These contradictions are what makes the lives of engineering designers difficult but also are the key to developing successful products.

Once these metrics have been identified, it is possible to quantify the ability of competitive products to meet established requirements. It is also possible to set target values for the product to be designed.

2.3.8. Set Target Values for Metrics to Ensure that Needs are Met

Target values for the established metrics should be created as a benchmark for assessing concepts and ensuring that the ultimate product will adequately satisfy customer needs. These target values generally take on two classes: ideal targets and marginally acceptable targets [3]. Ideal targets are the optimal target for the design team while marginally acceptable targets "just barely make the product commercially viable" [3]. The metrics usually have engineering units and the targets consist of equalities (equal to) or inequalities (greater than, less than, between) or a set of discrete values. These metrics will be used for the remainder of the design process to compare concepts and guide the development of the chosen concept(s) into complete products. Some aspects of the product might not be able to be reduced to a set of metrics. These specific design challenges should be identified at this point.

2.3.9. Identify Specific Design Challenges

Aspects of a product that cannot be fully described with a set of metrics should be identified and characterized at this point. Attributes such as the aesthetic appeal of an automobile or its handling cannot be completely captured with a set of numerical values. These factors often result from complex interactions between the product and human operators. The user's senses play a large role in defining these characteristics.

2.3.10. Plan for the Remainder of the Design Process

The final step in the product planning stage of design is to plan for the remainder of the design process. Ullman recommends a five-step process for completing this planning task [5]:

1. Identifying the tasks to be performed in the remainder of the design process,
2. Stating objectives for each task,
3. Establishing the personnel and time required for each task,
4. Sequencing the tasks,
5. Estimating product development costs.

These tasks and their attributes can be represented in a Gantt Chart as recommended by Otto and Wood [6]. This step is critical to ensure that the remainder of the design process proceeds in an orderly and timely manner. At the completion of this activity, the product planning stage of design is over and the conceptual design stage is ready to be initiated.

2.4. CONCEPTUAL DESIGN

The reconciled conceptual design stage contains ten basic activities. Each of these activities is summarized in the following section. The primary goal of this stage of the early engineering design process is to identify a single concept, or a small set of concepts, that are capable of meeting the requirements established in the product planning stage.

2.4.1. Develop Functional Requirements

Most engineering design textbooks agree that the first step in developing concepts is to translate customer needs and system requirements into functional requirements [3-6, 21, 22]. The basic process for performing this task varies based on the specific method used for the functional modeling task that occurs in the subsequent step. In general, developing functional requirements involves the identification of all of the energy, material and information flows that impact the ability of the product to satisfy customer needs and meet system requirements. Next, the essential transformations of flows the product must perform in order to satisfy the customer needs and requirements must be identified. These flows and functions will be structured in the next step in the conceptual design process.

2.4.2. Develop a Conceptual Functional Model of the Product

A functional model is a graphical depiction of the functional requirements outlined in the previous step. Functional models include functions, generally represented as verbs, which describe the desired transformations of flows, which are generally described using nouns. The process for creating a functional model depends on the modeling methodology chosen but in general involves the following basic steps:

1. Create a black-box model that includes the overall functionality of the product along with external flows,
2. For each input flow in the black-box model identify the sequence of functional transformations that are required to produce one or more of the output flows,
3. Aggregate these function sequences into a complete functional model for the product,
4. Assess the model's coverage of customer needs and system requirements, add functions/flows or decompose as required.

The multitude of reasons for creating a functional model during product design are detailed by Otto and Wood [6]. In general, the primary reason is to create a solution-neutral method of representing what a product needs to do without assuming how it is going to do it. This mapping of *what* to *how* represents the remainder of the conceptual design process.

2.4.3. Generate Functional Alternatives Using the Functional Model

For many design problems, and especially in complex systems with multiple energy domains, there may be multiple combinations of functionality that utilize varying energy, material and information flows to satisfy the customer needs and requirements. In this case, a functional decomposition for each identified configuration should be created. Ideally, these various decompositions should abstract up to a single uniform model (which at the highest level is the single-function black box model). There is rarely a single functional description that adequately models all possible form-solutions to a design problem.

2.4.4. Identify Component Solutions to Functions

Once various functional configurations of the product have been identified, component solutions to the functions in each configuration should be found. The objective of this task is to use historical information and designer experience to identify physical components that implement the desired functionality. Sources for this component information include [3]:

- Interviewing lead users (users who will have access to the product before the general populace),
- Patent searches,
- Searching published literature,
- Benchmarking related products,
- Internal searches.

Once identified, component solutions can be represented in a morphological matrix [6]. A morphological matrix contains product sub-functions as rows and component solutions to sub-functions in columns. For each cell (representing a single component solution to a specific sub-function) a graphic or text description is used to represent the solution. This chart is useful for performing the next task in the design process, combining components to product concepts.

2.4.5. Combine Components to Produce Concepts

To create complete concepts, a component element for each sub-function should be selected and then these elements should be combined. A morphological matrix is a useful tool for completing this process [6] in that it provides a graphical guide for combining the various identified component solutions. Ulrich and Eppinger recommend a similar construct called a concept combination tree [3] for performing this task. Pahl and Beitz suggest a similar approach [4].

Once component combinations have been identified, they must be firmed up in order to produce viable concepts. This process usually involves a design team taking each configuration and determining aspects including component compatibility, function-sharing possibilities and additional components necessary to produce a product that satisfies all desired functionality. Once a set of concepts has been developed, they must then be evaluated based on the established performance metrics.

2.4.6. Estimate Performance of Concepts Relative to Target Metrics

To evaluate concepts relative to each other and to existing target metrics, tools such as a Pugh chart [23] or weighted scoring matrix [3] can be used. These tools include weights for each evaluation criteria and a qualitative measure of the ability of each concept to satisfy the criteria. Quantitatively evaluating potential concepts is generally difficult at this stage in the design process due to the time, resources and knowledge required to produce mathematical models capable of predicting performance (behavioral models). Without detailed models, performance is generally estimated through simple calculations and experience with similar products.

2.4.7. Select a Set of Concepts for Further Consideration

Once the performance of the concepts has been estimated, it should be entered into a selection tool (Pugh chart, scoring matrix, etc.). Since the estimation of performance is inherent in using such tools, they are generally used to sort and identify a few good concepts rather than select the “best” concept. Some caveats that result from using such selection methods include [3]:

- The quality of the decomposition of the product’s functionality,
- The subjectivity of the selection criteria,
- How cost is factored into the approach,
- Selecting elements of complex design problems,
- The use of concept development throughout the design process.

The result of this step is a set of good concepts that are to be considered for the remainder of the design process. At any time during the remainder of this process, concepts that are found to be incapable of meeting the overall needs and requirements should be eliminated. Additionally, in later stages of design aspects of the various chosen concepts can be combined to produce new concepts. Ultimately, the design process converges on a single concept that is fully developed and described for production.

2.4.8. Refine System and Functional Requirements

The identification and comparison of concepts often reveals missing or incomplete system and/or functional requirements. At this point, the requirements tables and functional models should be updated to reflect any of these identified requirements. Once recorded, the set of concepts should be evaluated based on these new requirements to ensure that they still offer satisfactory performance.

2.4.9. Finalize Concepts

Next, the concepts should each be considered in the context of the design problem. The ability of each concept to satisfy the designer’s vision, the customer’s needs, system and functional requirements and any other identified criteria should be assessed by the design team.

2.4.10. Develop Plan for Embodiment and Detailed Design of Concepts

The final step in the conceptual design stage is to develop a plan for completing the remainder of the design process including the embodiment and detailed design stages. This includes updating project management tools (the Gantt chart if one was used) and restructuring cost and personnel evaluations based on the new information available regarding potential concepts and their form-specific solutions.

2.5. FUNCTIONAL ORGANIZATION OF EARLY DESIGN FOR COMPLEX SYSTEMS

The design approaches summarized in Section 2.1 have their roots in the Design Theory community and the traditional (sometimes called the European) school of design. These methods were developed as a means of classifying the various steps that generally occur during the design of a system. For the most part, these methods were developed by studying particular design examples in detail and abstracting the tools and techniques that resulted in successful designs.

In parallel, the Systems Engineering community took a different approach to classifying the design of systems. Systems Engineering evolved from the software and industrial engineering communities. Both software engineering and modern industrial designs usually involve very complex multi-level problems. As a result, the Systems Engineering community developed tools and methods for managing complexity (usually in the form of some kind of model) and then proposed methods for using these tools in new systems engineering projects.

Both approaches to representing design have their merits and limitations. The traditional school of Design Theory provides a clear process for designing products that can be taught to new designers (generally starting at the Freshmen level in the collegiate curriculum). However, to generate a process that was simple enough to be communicated to novice designers, the applicability of the process to large multi-scale design problems was compromised. As can be seen in the reconciliation of these approaches as shown in Sections 2.3 and 2.4, the design of a system is reduced to a sequential set of steps with only a cursory decomposition of the design problems occurring during conceptual design (during the functional analysis step). This works for simple problems with a relatively small number of constitutive functional elements but is ill-suited for larger problems such as those found in the modern engineering industry.

The Systems Engineering approach has resulted in a number of modeling tools for representing complex systems (culminating in the development of SysML [16]). However, the focus on models and tools and then processes has resulted in a technique that is not easily relatable to those outside of the engineering profession. Most models of the systems engineering process that appear in Systems Engineering texts [1, 2] are vague in the descriptions of the various steps in the process and are often highly circular in nature and include complex nests of loops within the various general steps.

While these inner loops are inherent in the design process of complex multi-level systems, the overall early design process itself is essentially linear at the highest level. There is a clear starting point (a product idea), a clear ending point (some set of valid concepts) and a general set of activities that must be conducted in a rough temporal series to get from the start to the end. Essentially, the process of design is a set of general operations that are performed in sequence on a set of inputs to result in a set of outputs. As a result, this process is capable of being represented with a transformational function-type representation. Since functional modeling is proposed to solve such problems within the context of design activities, the same general idea should apply to solving the representation of the design process itself! In the following section, this idea is explored.

2.5.1. Functional Modeling of the Reconciled Design Process

Developing a functional representation of the design process presents a few problems. First, no standard lexicon (such as the Functional Basis) was intended for such an approach and therefore new function and flow terms will have to be used. However, the same principles used in developing the Functional Basis [10] can be used to develop the terms for the design process model. These principles include identifying a minimal set of terms to describe the functions and flows in the model and minimizing overlap between the terms. Additionally, the concept of using verbs for functions and nouns for flows was preserved. Unlike the development of the Functional Basis, the terms used in the design process model were defined from a common language dictionary [24] to ensure a common understanding of their meaning without creating new definitions. The terms identified in the creation of the model appear in the following section.

2.5.1.1. Definitions

Functions (verbs):

Conceive – "to cause to begin"

Validate – "to support or corroborate on a sound or authoritative basis"

Prescribe – "to lay down as a guide, direction, or rule of action"

Perform – "to do in a formal manner or according to prescribed ritual"

Assess – "to determine the importance, size, or value of"

Observe – "to take notice of and be guided by"

Translate – "to express in more comprehensible terms"

Quantify – "to determine, express, or measure the quantity of"

Find – "to come upon by searching or effort"

Combine – "to unite into a single number or expression"

Compare – "to examine the character or qualities of especially in order to discover resemblances or differences"

Select – "chosen from a number or group by fitness or preference"

Decompose – "to separate into constituent parts or elements or into simpler compounds"

Flows (nouns):

Concept – “an abstract or generic idea generalized from particular instances”

Process – “a series of actions or operations conducing to an end”

Design – “the arrangement of elements or details in a product or work of art”

Need – “a lack of something requisite, desirable, or useful”

Characteristic – “a distinguishing trait, quality, or property”

Metric – “a standard of measurement”

Target – “a goal to be achieved”

Form – “the component of a thing that determines its kind”

Behavior – “the way in which something functions or operates”

Performance – “the ability to perform (see definition of *perform* in functions)”

Model – “a system of postulates, data, and inferences presented as a mathematical description of an entity or state of affairs”

All definitions are taken from Merriam-Webster’s Online Dictionary [24].

2.5.1.2. Models

The reconciled design theory presented in Sections 2.3 and 2.4 was used as a starting point for creating a functional model for the process of designing systems. To create the model, the steps in the reconciled theory were reduced to functions and the boundary flows of information for each function were identified. Next, a verb-noun pair was selected to represent each function. The list of verbs (for functions) and nouns (for flows) is shown in the prior section. Each function is presented from the context of the designer (or design firm) and each function represents some transformational action performed by the designer.

The first function can be extracted from the need to develop a vision. In this step in the design process, an idea is conceived by the designer and reduced to a product concept.

Next, this concept is validated through a market analysis. Once validated, a design process is prescribed (usually represented in a product proposal). This process is then performed by the designer to result in a product design. This chain of four functions represents the entirety of the reconciled design theory with the first three steps corresponding each to a single function in the chain and the remaining seventeen steps condensed into the act of performing an actual design process. Essentially, this model isolates the act of preparing and planning a design process from the process itself. The chain of functions resulting from this analysis appears in Fig. 2.1.

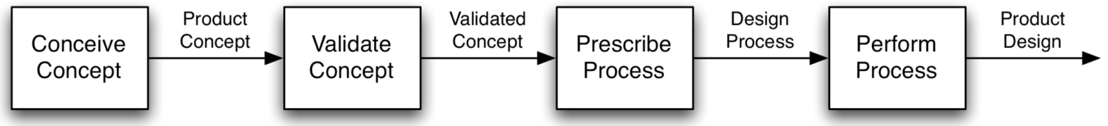


Fig. 2.1. Functional Model of the Design of a System

The remaining seventeen steps in the reconciled design theory were then represented by a separate series of functions (each contained within the *perform process* function appearing in the prior model). The first step identified in this chain is the observation of customer needs. These needs are then assessed to determine their relative importance by comparison to other products that satisfy the observed needs. Next, these needs (the lack of something desirable) are translated into characteristics (this broad classification includes the traditional concept of requirements) of the product to be designed. Next, these characteristics are quantified to result in a series of product metrics. From these metrics, a series of target values is generated. This chain of five functions represents the seven final activities proposed in the reconciled design theory's product planning phase and appears in Fig. 2.2.

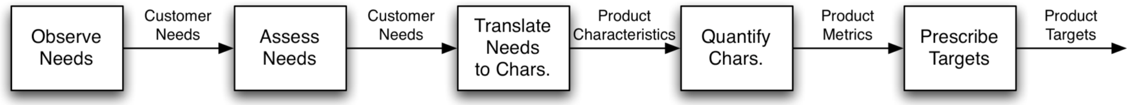


Fig. 2.2. Planning Functional Model

The design activities in the conceptual design phase of the reconciled design theory were also reduced to a chain of functions. This chain essentially represents the transformation of the desired characteristics of the product into a validated concept for the product. The first function identified in this chain is to translate the characteristics into functions. The result of this step is a functional model. This functionality is then used to find form solutions. These forms are then combined to produce a set of product concepts. Next, the behavior of the concepts is assessed and then compared to the target values generated in the product-planning function chain. From this comparison, concepts can be selected and finally validated based on the customer needs and characteristics that were identified in the product planning process. The result of this chain of functions (Fig. 2.3) is a validated set of concepts.

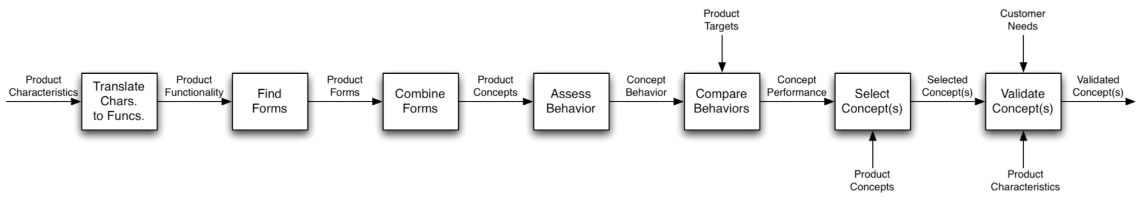


Fig. 2.3. Conceptual Design Functional Model

2.5.2. Update of Model for Complex Systems Design

Since the functional analysis of the early design process was based on activities reconciled from traditional Design Theory texts, it inherits the general weaknesses of such approaches. Namely, an inability to address the difficulties inherent in complex multi-level design problems and the need for rigorous modeling and validation across these multiple levels. To overcome these limitations, the models developed from traditional Design Theory were extended to include concepts from Systems Engineering including multi-level modeling and validation as well as requirements flowdown.

In this modification, the first function in the product planning chain (observing needs) was removed. For the remainder of the modified design process, it is assumed that a set of needs has been established (with or without the involvement of the designer). The next step, translating needs to characteristics, is decomposed into two separate functions: translating needs to functions and then finding characteristics based on functions. This decomposition is necessary due to the fact that needs represents a lack of something and a characteristic represents the presence of something. Between these representations, an intermediate form is necessary to ensure a complete transformation between these concepts. In the modified design process, functionality forms this bridge. This transformation can be qualified as follows: needs, representing the lack of something, must be fulfilled through operation of the product; as a result the product has to include some kind of functionality to ensure this fulfillment; once the desired functionality of the product has been identified it can then be used to define characteristics of the product that must exist to ensure this functionality can be performed. Once identified, these characteristics can then be quantified as done in the previous model of the design process. However, the next step, prescribing targets, is decomposed into three separate functions in the new model. This decomposition results from the implementation of models to perform the target setting process as recommended in Systems Engineering methods. The first step in this newly decomposed chain is to find a prescriptive (rather than descriptive) model for the metrics. This model is then assessed and the resulting behavior used to set the target values for metrics.

The design process diverges at this point. In a traditional design process the targets, needs and characteristics identified would be used to find concepts. However, for complex multi-level systems there are different approaches that can be taken. Rather than model all of the various approaches, a modular and scalable representation was developed. At the completion of the first chain of design functions (shown in the top row of Fig. 2.4) a system-level concept generation, descriptive modeling and concept selection process (referred to as a synthesis chain) may be implemented to develop concepts at the system level. These three steps are modeled in the synthesis function chain appearing in Fig. 2.5 and are essentially identical to the chain developed for conceptual design in the prior design process model (Fig. 2.3). At the completion of this sequence of functions, decomposition can be performed using the same basic set of functions as those in the top row of Fig. 2.4. This decomposition results in a set of sub-system needs, characteristics and targets that can be used as the inputs to a new synthesis chain at the sub-system level. Once completed, the results of this chain can be used to validate the behavior of the refined system relative to the prescribed targets generated from the previous higher-level analysis. This decomposition and synthesis sequence can be repeated until a detail level has been achieved that is appropriate for the level of complexity of the system being designed. Such an approach would be classified as "top-down".

An alternative approach would be to proceed with the complete decomposition of the system and then begin the synthesis chain at the lowest level and proceed with synthesis and validation from the bottom-up. This approach would be classified as a "top-down-bottom-up" approach along with any derivatives of this sequence. A purely "bottom-up" approach is not classified because some form of the decomposition sequence must be performed before the synthesis sequence can be completed. A purely "bottom-up" approach is not practical during systems design as it would mean that the system is synthesized at the lowest level with no information regarding the system's overall intended functionality, needs to fulfill or desired characteristics. The resulting models for the decomposition and synthesis function chains appear in Figs. 2.4 and 2.5 respectively along with groupings of related functions.

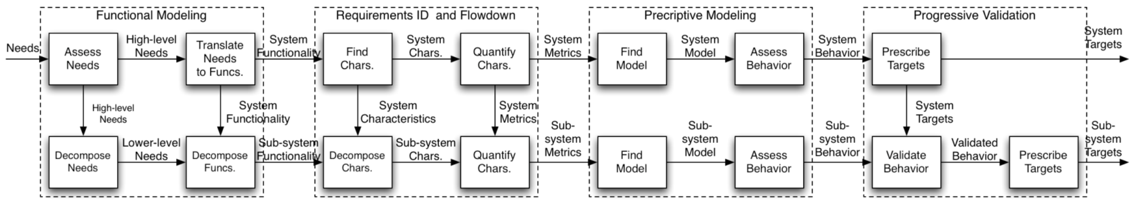


Fig. 2.4. Modified Design Decomposition Functional Model

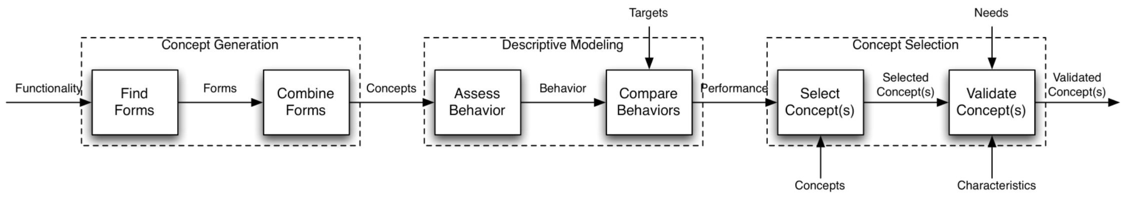


Fig. 2.5. Modified Design Synthesis Functional Model

2.6. TOOLS TO ASSIST DESIGN OF COMPLEX SYSTEMS

To assist the implementation of the modeling approach identified in the previous section, several design tools are proposed. The first design tool proposed is functionality assisted behavioral modeling. This tool utilizes the functional models created during system decomposition to assist the prescriptive and later descriptive modeling tasks required in the design process. Sensitivity measures are useful metrics during systems design and must also be implemented in the modeling approach. As a result, a sensitivity analysis method based on a functional decomposition is also proposed. This sensitivity analysis method is driven by the functionality assisted behavioral modeling approach. Finally, a software implementation of these tools is presented. A comprehensive application of the design process developed in Section 2 along with the design tools developed to support the process is shown in Section 6.

3. FUNCTIONALITY ASSISTED BEHAVIORAL MODELING

A behavioral model is a quantitative representation of selected aspects of the behavior of a system. Such models are often used during the design of systems, and specifically complex systems, to find quantitative answers to specific questions including:

- What should the values for performance target metrics be?
- How can I design a system that satisfies these targets?
- What can be changed within a system to affect its performance and to what degree do these changes impact performance?
- How do changes in fidelity of models affect the performance predictions from the models?

In general, the questions tasked to behavioral models are "what" and "how." Specifically, *what* is the behavior of a system under a specific set of assumptions and *how* does it accomplish that behavior? Both questions must be answered in the course of a complex design problem. As a result, a modeling framework that enables designers to answer both questions in parallel provides a significant contribution to the practice of modern systems design. To produce such a framework, it is necessary to know when each question ("what" versus "how") should be asked and then apply the answers to further the design process.

The same "what" versus "how" scenario can be seen in the realm of functional modeling. Functional modeling originally evolved as a means of representing "what" a system should do independently from "how" it is done [25, 26]. The difficulty in this distinction lies in the fact that the two questions are not entirely independent from each other. What something does is generally dependent, to some measure, on how it is done. During design, there is a general trend of "what" initially being the most important question and "how" becoming more important as the process continues.

As functional modeling matured, it was found that models created by reverse engineering products differed in information content from those created during the course of a new design [27]. This trend of a model expressing more content focusing on "how" something is done rather than "what" it does is consistent with the flow of information in the design process. Essentially, the changes in the functional representation of system between the early design stages and at the completion of a design process capture the evolution from what a system is supposed to do into how it is done in the ultimately designed system. Since the same transitions from "what" to "how" appear in both functional and behavioral modeling, there is merit in investigating whether these technologies can be used in conjunction during the design of systems. The result of such an investigation is presented in this section.

The results of this work are presented in four sub-sections. The first sub-section outlines functional modeling and behavioral modeling in the context of design. The second sub-section describes a modeling approach that combines both function and behavior along with a cursory example. A complete case study for a hybrid automotive system is then presented in the third sub-section. Finally, conclusions and continuations of the work appear in the fourth sub-section.

3.1. THE DESIGN OF SYSTEMS

As stated in Section 2, two broad schools of formal design practices for systems exist: the "European," or traditional, school of design and the "Systems Engineering" approach. Traditional thoughts on design are best summarized by the work of Pahl and Beitz [4]. They define the design process for a system in four basic phases: (1) product planning, (2) conceptual design, (3) embodiment design and (4) detailed design. Other research into design methodologies within this body of work define similar processes involving more or less the same four steps [5, 6, 22]. Some of these methodologies move the boundaries of the steps around and regroup the basic activities differently but in total represent the same basic sequence of events: 1) The first phase of design is to gather information about the system's user and their needs and to map this information to the highest level functionality of the system. 2) The required functionality is explored in detail and potential solutions for this functionality are created and evaluated. 3) After selecting a concept for production, the physical requirements for the concept are defined along with solutions for the product's auxiliary functionality. The overall performance of the concept relative to the user's needs is then assessed. 4) A complete description of the system is made to allow the product to be manufactured. A complete description of these steps appears in Section 1.

A full account of the "Systems Engineering" method of design can be found in Sage and Armstrong [1]. Systems Engineering focuses more on management and information control during the design process for large systems and emphasizes the understanding and development of requirements (formal statements of a user's needs). Specific processes for designing electromechanical systems are not generally defined. However, the classifications used in Systems Engineering: formulation, analysis and interpretation [1] can be mapped to the first two phases in the traditional method of engineering design. Formulation, which includes problem definition, value system design and system synthesis, straddles the product planning and conceptual design phases of the traditional design method. The problem definition and value system design activities in Systems Engineering map to activities in the product planning and system synthesis phases of conceptual design. The analysis classification includes system analysis and modeling along with refinement of alternatives. These activities are followed by an interpretation that includes decision making and planning for action. The activities that occur in the analysis and interpretation classification in Systems Engineering map to activities that occur during conceptual design in the traditional engineering design method.

In summary, when it comes to the actual process of designing systems, both schools of engineering design promote the same basic order of operations during the initial stages of system design: determine what needs to be done at a functional level, find solutions that can potentially accomplish this functionality, compare the solutions through the use of models and make a decision about which solutions to investigate further.

3.1.1. Functional Modeling During Design

The functional decomposition of a complex design problem is promoted in both traditional engineering design [4] and Systems Engineering [1]. Several methods exist for performing this decomposition such as formal methods [6] and the Function-Flow Block Diagram (FFBD) approach as used in Systems Engineering [1, 2].

The functional decomposition approach used in this work evolved within the traditional engineering design approach beginning in the early 1960s [25]. This functional analysis approach focuses on the use of verb-noun pairs along with an explicit breakdown of flows into energies, materials and signals [26]. This flow-based approach has been extensively researched as of late and significantly formalized in the work of Hirtz et al. [10]. The benefits of this approach over the FFBD approach include the use of a standard modeling language (the Functional Basis) and an emphasis on the energy, material and flow-based identification of functions (FFBDs do have advantages in the modeling of logic and data flow within the model). A complete survey of this branch of functional analysis appears in the work of Nagel and Hutcheson [28]. For the remainder of this work, a functional model will refer to a functional decomposition of a system using the flow based methodology and Functional Basis lexicon approach [29].

The functional decomposition of a system allows a complex design problem to be broken down into smaller elements based on those elements' intended functionality. These elements can then be analyzed in greater detail individually. In most formal system design methods, the functional decomposition is also recommended as the starting point for identifying potential solutions to the design problem in order to decouple the task of representing what a system needs to do from how it is going to do it [3-6]. Thus, the desired functionality of a system can be described before actual solutions are identified. From morphological charts [6] to current knowledge driven concept generation algorithms [30], significant research has been conducted into expanding the ability to explore the solution space for a problem by using a function-based solution identification method.

3.1.2. Behavioral Modeling During Design

A behavioral model is a quantitative representation of a system, or a specific aspect of that system. Such models can be used to prescribe performance targets early in the design process and can predict the performance of systems relative to these targets later in the design process. There are two general approaches for creating such models: the abstraction approach and the component-based approach.

In the abstraction approach, a behavior of interest is identified and a model is created based on an abstraction of the system that exhibits the behavior of interest [31-33]. For example, if the performance of an internal combustion engine needs to be modeled an abstracted model of the engine may be created by using an approximation of the combustion processes or results from dynomometer testing. Such a model represents a parameterized abstraction of the behavior of the system. The component-based approach for modeling the same system would be to model the constitutive elements of the engine separately and then combine them (generally through an automated or semi-automated process) to produce a complete system model.

Both approaches have their strengths and weaknesses. Abstract models are generally more focused on the behavior of interest while component-based models are generally more closely associated with the system itself. As a result, an abstract model may provide a good description of the behavior of interest but may be of little use if the system changes or if the assumptions made about the system during the abstraction process are faulty. Conversely, a component-based model can provide a variety of information about a system but may not able to predict a specific aspect of its behavior as well as a model abstracted solely for that purpose. Essentially, these models answer two separate questions. Abstract models answer the question of *what* a system does and component-based models answer the questions of *how* a system functions through the action of its constitutive elements.

The abstraction method of modeling systems is generally the method taught in engineering educations. Traditional engineering classes (physics, thermodynamics, etc.)

focus on modeling systems by creating a set of equations or relationships that represent an abstracted behavior of a complete system. In general, the result of this type of modeling process is set of algebraic or differential equations, which are then used to investigate some aspect of the system's performance. This method of modeling provide good insight into what a system does by providing elegant analytical equations but is generally limited to small systems that exhibit rather simple behavior (when compared to larger, more complex systems).

In contrast, generic component-based system modeling platforms such as Simulink [8], Dymola [34] and Bond Graph based applications [14] have been developed as a means to model complex systems across multiple domains. The Modelica approach has its roots in the object-oriented modeling approach developed by Elmqvist [35] while Bond Graphs originated from Paynter [36]. In these approaches, the behavior of a component of the system is modeled independently from the other components of system and then these model elements are automatically assembled to produce a complete system model. Such an approach is necessary for large complex systems and does a good job modeling how system behavior results from the behavior of its constitutive elements.

3.1.3. Functional Abstraction and Behavioral Modeling

A functional model contains encapsulations of desired transformations (in the form of the functions themselves), explicit definitions of energy, material and signal flows and flow routing information. In general, this approach mimics the one used in component-based modeling but in a qualitative, rather than quantitative manner. In component-based modeling, a system is broken down into constitutive elements, boundary variables are identified, model elements are created to represent the relationships between these variables and then the elements are combined according to a connectivity schematic.

In prior work, functional modeling techniques have successfully been used to drive quantitative reasoning in systems in limited application [14, 37-40]. The use of functional models to assist model-based design of systems using discrete state behavioral models is demonstrated in Bhatta et al. [37]. In Bhatta's work, the functions in the functional models are represented as “a schema that specifies the behavioral state the function takes as input, the behavioral state it gives as output” [37] along with a link to internal causal behavior that identifies the behavior that enables the function. Behaviors are “represented as sequences of state transitions between behavioral states” [37]. This explicit input to output transformational representation is generally not suited to modeling the behavior in large complex systems that require acausal modeling techniques [38]. Research into the augmentation of SysML with multi-aspect behavioral models for components has also been performed [39]. While SysML provides an object-oriented framework for representing systems, it does not include a formal representation of functionality.

The objective of the work presented here is to investigate the use of formal functional modeling techniques in the behavioral modeling process of systems. A formal functional model is one created using a standard method and a fixed lexicon of function and flow terms. To perform this investigation, a functional model driven behavioral modeling approach will be developed and applied to the behavioral modeling of a multi-domain engineering system. The results of the approach and insights gained in its application are used to assess its effectiveness and potential uses in the design process of systems.

3.2. FUNCTION-BASED BEHAVIORAL MODELING

The proposed process of using a functional model to drive the behavioral modeling of a system consists of the following four basic steps along with iterations between the steps:

1. Functional modeling,
2. Flow variable identification,
3. Behavioral model element definition,
4. Model assembly and solution.

These steps are outlined in the remainder of this section along with an introductory example.

3.2.1. Functional Modeling

The first step in the process is to create a functional model of the aspect of the system being investigated. The objective of this step is to model all of the important transformations of energies, materials and signals that occur during the behavior of interest. To assist this modeling step and promote model reuse in subsequent steps, a standard lexicon such as the Functional Basis is necessary when creating functional models. The purpose of this step is to establish a baseline set of input and output flows along with a general idea of what the system does.

Functional modeling begins with the creation of a black box model. This model captures the overall functionality of the aspect of the system being investigated along with external energy, material and signal flows. These external flows are used later to create detailed chains of internal functions. For the black box, the overall function may be represented in free language terms (it is sometimes difficult to clearly express overall high level functionality with Functional Basis function-flow pairs). If model reuse at the system level is desired, the overall function should be named with Functional Basis terms along with the free-language terms. An example of a black box model appears in Fig. 3.1. This model was created for use in developing the behavioral models for the powertrain of a hybrid racecar (a complete example appears in Section 3.3). The overall functionality of this system is to store and supply rotational energy for use in propelling the car. When combined, the store and supply functions can be represented as the provision function using the Functional Basis lexicon. For the investigation performed on this system, the inputs selected were the flow of chemical energy from the fuel, control inputs from the driver and status signals from the vehicle. The outputs were the rotational energy from the powertrain along with control signals to the conventional braking system of the vehicle (to be used in conjunction with regenerative braking). In the model, dashed arrows represent signal flows and solid lines represent energy flows (thick solid arrows represent material flows in functional modeling but none are considered in this analysis). A two-way arrow means that the flow is thought to be bi-directional. Two-way arrows are not a standard part of functional modeling but are used for convenience and do not have an explicit relationship with causality in the behavioral models created based on the functional model.

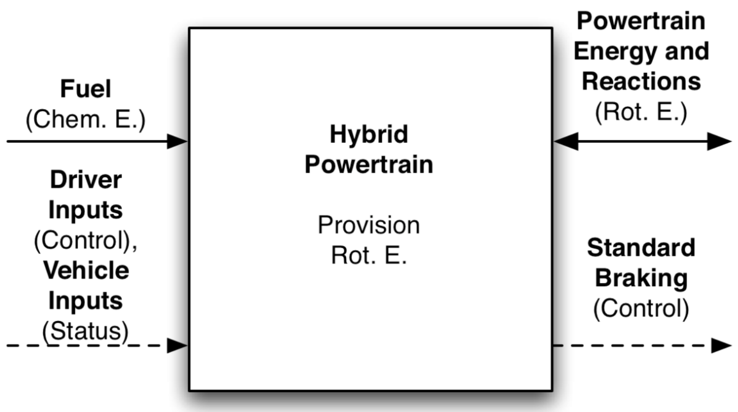


Fig. 3.1. Hybrid Powertrain Black-box Model

Once the black box model has been created, an appropriately abstracted functional model must be made. This model qualitatively captures the set of transformations that must occur on external input flows in order for them to become the desired outputs and vice-versa. The level of detail required in such a model depends on the intended use of the behavioral model. The purpose of creating this model is to identify the distinct functional elements of the system along with the flows that connect these elements. An example of such a model appears in Fig. 3.2. This model is based on a hybrid powertrain and represents a very high-level view of the functions involved in such a system. Specifically, storing and supplying electrical energy (the powertrain is based around the needs of a Formula Hybrid racecar and mandates the use of electrical energy), converting this energy to rotational energy, converting chemical energy to rotational energy, distributing these two rotational energies and controlling the various conversion processes according to driver and vehicle inputs. A simple abstraction of the suspension and chassis functionality is included for reference and because these functions will need to be modeled later on in order to perform a complete investigation on the powertrain system.

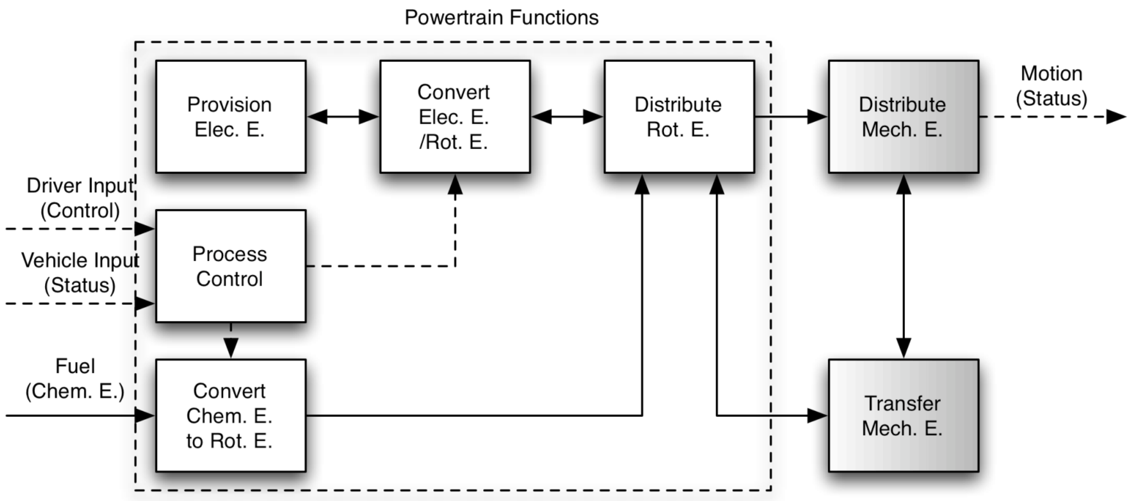


Fig. 3.2. Hybrid Powertrain Functional Model

The functions in a functional model can be identified though a variety of methods including prior experience with similar systems, analogies to other systems and reverse engineering of existing solutions. The objective of this step is to determine exactly *what* needs to be done by the system in order to satisfy the user's needs. The general process for creating such a functional model involves selecting an input flow from the black box and identifying the chains of functions that must occur in sequence in order to transform the input flow into one or more of the output flows. The functions in these models should be expressed with a standard taxonomy such as the Functional Basis to ensure user independence of the model and allow storage of model elements according to functionality. The chains are then aggregated by combining common functions in the chains to result in a complete functional model.

3.2.2. Flow Variable Identification

The functional model for a system qualitatively captures the transformations of energies, materials and signals through the operation of the system. A behavioral model quantitatively expresses these transformations. The first step in creating a behavioral model from the functional model is to select a set of variables for these transformations. These variables come from the flows listed in the functional model and represent numeric quantities that are expressed in the elements of the behavioral model that correspond to each transformational function. In general, three classes of flow variable types have been identified in the course of this work. These three types are summarized in Table 3.1 along with an example for each. For additional reference the type of variable used to represent each flow type is presented in the context of a C++ program (double means double precision floating point, long means long integer, struct means a user-defined data structure and class means a user-defined object).

Table 3.1. Flow Variable Types

Flow Type	Description	Rotational Energy Example	C++ Representation
Type I	Single Parameter Description	Power (kW)	double, long
Type II	Multi-parameter Description	Flow (rad/s), Effort (N*m)	vector<double,long>
Type III	Special Description	Flow, Effort, Constraints	struct, class

In general, the higher the type, the higher the fidelity model needed to represent the transformations required to capture the system's behavior. Most component-based modeling techniques focus on Type I and Type II flows as models using these flows can generally be solved using automated techniques such as the approach used in the Modelica language for representing hybrid differential algebraic equations along with Modelica solvers such as Dymola [34]. For systems involving complex flow variable sets (Type III), custom solvers are generally required.

An example of a set of Type I flows appears in Table 3.2. This table was generated during the creation of a simple model for the hybrid powertrain system introduced previously. The routing of these flows in the functional model for the hybrid powertrain is shown in Fig. 3.3. In Table 3.2 and Fig. 3.3 a FourCC (four character code) identifier was associated to each function and flow from the Functional Basis. A FourCC code allows a simple four character code to be used to identify each flow or function in a format that is human readable but also extremely computationally efficient (each FourCC can be represented as a single 32-bit integer instead of a character string or a custom string class) thus saving storage space and decreasing the computational burden for flow and function matching in a computational application. The use of FourCC codes bridges the gap between computational and human-friendly representations. A description of the FourCC codes used in the models discussed in this section appears in Table 3.3.

Table 3.2. Hybrid Powertrain Type I Flows

Flow	Symbol	Type	Units	Description
Electrical Energy	$ELCE_{Ia,x,0}$	Power	kW	Power Flow
Chemical Energy	$CHME_{Ia,x,0}$	Power	kW	Power Flow
Rotational Energy	$ROTE_{Ia,x,0}$	Power	kW	Power Flow
Control Signal	$CNTL_{Ia,x,0}$	Analog	Varies	A single analog control signal
Status Signal	$STAS_{Ia,x,0}$	Analog	Varies	A single analog status signal

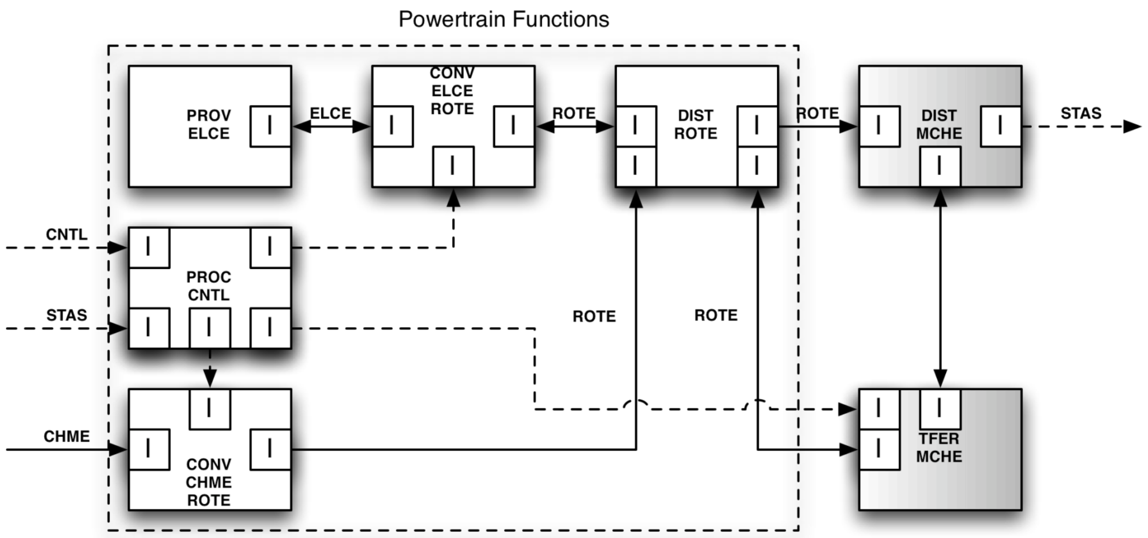


Fig. 3.3. Hybrid Powertrain Type I Flow Routing

Table 3.3. Hybrid Powertrain FourCC Codes

Flow	FourCC	Function	FourCC
Control	CNTL	Convert	CONV
Electrical	ELCE	Distribute	DIST
Mechanical	MCHE	Process	PROC
Rotational	ROTE	Provision	PROV
Status	STAS	Transfer	TFER

3.2.3. Model Element Definition

The next step in the behavioral model development process is to develop or reuse behavioral model elements. These elements correspond to the concept of a component in the field of component-based modeling but are abstracted to the boundaries of the functional elements in the functional model rather than traditional component boundaries. These behavioral model elements express relationships between the flow variables identified in the previous step. Like the identification of flow variables, the development of a behavioral model element is related to the intended use of the model, the desired local and global model fidelity and the amount of information available regarding the aspect of the system being modeled. In the case of new model element development, the flow types identified in the previous step serve as a starting point in identifying the relationships that make up the behavior of the element. From these variables, physical principle and domain specific knowledge can be used to develop equations that relate the variables. This process is explained in the context of object-oriented modeling in [38] and in conventional abstraction modeling in [32, 33].

Essentially, this step is completed in the same fashion as a traditionally abstracted model but at a smaller, and easier to manage scale. As such, the specific process of making these models is left outside the scope of this work. However, like the flow classification system, a model element classification system has been developed as well to guide the application and development of model elements. These types are outlined in Table 3.4 and have a rough correspondence to the types of flows that are connected to the element (e.g. a Type I model element generally has Type I flows connecting to it, although this is not always the case). For some modeling tasks, models with locally different levels or fidelity are appropriate. For example, the hybrid powertrain model developed in the subsequent example used all Type I elements in one model and mixed Type I and Type II elements in another.

Table 3.4. Behavioral Model Element Types

Behavior Type	Description	Convert Elec. E. to Rot. E. Example
Type I	Form neutral representation	Power conservation equation (no form)
Type II	Form family dependant	Constant torque, constant power transitional motor model (based on the motor solution family)
Type III	Solution specific	Multiphysics FEA motor model (based on a specific motor)

The general trends associated with model element type and various factors in the application of the behavioral model appear in Table 3.5. In the context of this work, fidelity is defined as "accuracy in details" [24] with accuracy being "conformity to truth or to a standard or model" [24]. As such, as the model type increases, the predicted behavior more closely approximates a physical implementation of the solution, thus the fidelity, as defined, increases with model element type. With the cost of accuracy comes the additional need for form specific information (which may or may not be available depending on when the model is being used and what it is being used for) and in general increased solution difficulty (both in time and increased modeling resources). The models also become less flexible as they become more closely tied to a specific form of the solution. Essentially, lower type models answer the question "what" and are generally appropriate for prescriptive use (*what* do I want this system to do) and higher type models are more appropriate for descriptive use (*how* does this system work).

Table 3.5. Type Trends

Increasing Type:
+Fidelity
+Form Information
+Solution Difficulty
-Flexibility
-Prescriptive Use
+Descriptive Use

An example of a Type I model element from the hybrid powertrain example follows. The element describes the quantitative transformation of electrical energy to rotational energy in response to a control input. Such functionality is generally realized with an electric motor. However, at the Type I level, no assumptions were made regarding the form of the solution. The variables used in this model element appear in Table 3.6. Flow variables are labeled as such and use the naming convention shown in Table 3.4. Parameters are also listed along with a FourCC code plus index naming convention (for use in a computational implementation). Units and a short description are also included. The actual relationships developed for this model element are shown in Table 3.7. In this element, two relationships were identified that describe the intended behavior: a linear relationship between a control signal and the rotational power output by the element (Equation 0 in Table 3.7) and a two-way efficiency relationship between the rotational energy and electrical energy flows (power can flow both ways through the element in this case, Equation 1). The intended use of this element is to determine an efficiency required by the eventual solution to the element. Once established, this information can be used to select a solution. Once a solution has been identified, a Type II model can be developed based upon the form-specific information obtainable from the selected solution family (e.g. an electrical motor/generator). The results of the Type II model can be used to verify whether or not the selected solution family is capable of meeting the efficiency requirements set using the Type I model. The model element shown is used as part of the example in Section 3.3.

Table 3.6. Convert Elec. E. to Rot. E. Type I Model Variables

Sym.(E)	Sym.(L)	Type	Units	Description
P_M	$ROTE_{Ia,0,0}$	Flow	kW	Rotational Power
P_A	$ELCE_{Ia,0,0}$	Flow	kW	Electrical Power
ξ_M	$PARM_0$	Parameter	(0,1]	Conversion Efficiency
$P_{M,max}$	$PARM_1$	Parameter	kW	Maximum Power
C_M	$CTLS_{Ia,0,0}$	Flow	[-1,1]	Motor Control

Table 3.7. Convert Elec. E. to Rot. E. Type I Model Example

Relationship	Type	Ref. Number
$P_M - C_M \cdot P_{M,\max} = 0$	Exp. Lin. Alg. Eq.	0
$if(C_M) >= 0:$ $P_A - \frac{P_M}{\xi_M} = 0$ $else:$ $P_A - P_M \cdot \xi_M = 0$	Hyb. Imp. Alg. Eq.	1

3.2.4. Model Representation, Assembly and Use

The technique required to represent, assemble and use model elements depends on the type of model developed and the nature of its elements. For Type I and Type II models in the engineering domain, the models often take the form of hybrid implicit non-linear differential algebraic equations (DAEs) or some subset of this rather large class of equations. Certain subsets of this model class have existing commercial solvers. As such, it is left to the model developer to choose which solution is appropriate based on the model elements developed and their intended use. For Type III models, it is generally the case that a custom solution framework must be developed (an earlier example by the author demonstrates this need [40]). In most cases, an existing component based modeling representation and solution framework should be used once model elements have been identified. Such frameworks include Matlab+Simulink for model elements that can be adapted to a fixed causality block diagram format or Modelica+Dymola for general acausal modeling with hybrid DAEs. The models used in the hybrid powertrain example following were represented within an object oriented programming environment (C++) and solved using a sophisticated non-linear equation solver [41] along with a Runge-Kutta 4 integrator. The details of this implementation appear in Section 5. Since in the general case, model representation, assembly and solution is problem dependent, an all-inclusive solution method is not proposed nor presented in this work. An example application of the proposed functional model based behavioral modeling method outlined in this section and implemented in a custom computational framework follows.

3.3. AUTOMOTIVE MODEL EXAMPLE

To investigate the use of a functional model in the derivation of a behavioral model, an example was conducted based on the design challenges faced by Formula Hybrid teams. Formula Hybrid is a collegiate design competition that requires students to design, build, test and compete with a prototype electric hybrid open-wheeled racecar [42] and is a spin-off of the Formula SAE competition [43]. In addition to the challenges presented by Formula SAE, the successful design of a Formula Hybrid car requires an understanding of the behavior of a system with multiple energy domains, complex control problems and a variety of potential solutions outside the domain of conventional automotive solutions. In the design process of such a vehicle, behavioral models are required in order to understand how design choices will impact vehicle performance. Conventional hybrid design knowledge is not well suited to this design problem due to the increased emphasis on performance that a hybrid racecar must demonstrate over a hybrid passenger car. To design a competitive entry, a variety of potential powertrain options must be evaluated and fundamentally sound choices must be made before embodiment and detailed design can be completed. The lack of existing knowledge and expertise in this area, and the wide variety of powertrain functionality configurations, perfectly suit this example to the application of functional modeling to assist behavioral model creation.

In this example, two models were developed. An exclusively Type I model that was primarily based on energy conservation analyses that was targeted for making high-level decisions about powertrain configurations and performance targets (answering the *what* question) and a mostly Type II model that was targeted for solution family selection for the powertrain's various functional elements (to answer the *how* question).

3.3.1. Functional Modeling

The first step in developing the behavioral model was to create an appropriately abstracted functional model. The black-box model shown in Fig. 3.1 was used as the starting point for creating a functional model that described the powertrain system of a hybrid racecar at a level of detail that was appropriate for the analysis to be performed. In this case, two behavioral models were created. The first model was to be used in a preliminary, low-fidelity power flow analysis to prescribe efficiency and performance targets while the second was to be used as a descriptive model of various powertrain concept's performance. To create this model, each input and output flow to the black-box was investigated and a set of functions was developed to represent the transformations necessary to implement the desired functionality of the powertrain. The result of this process was the functional model shown in Fig. 3.2. This model was used for the Type I analysis. For the Type II analysis, a slightly more detailed representation of the vehicle was necessary. As a result, the transfer mechanical energy function in the model (which represents the functionality of the suspension elements of the car) was decomposed into two separate elements for the front and rear suspensions. Thus allowing the analysis of each to be separated during the modeling process. The resulting model appears in Fig. 3.4.

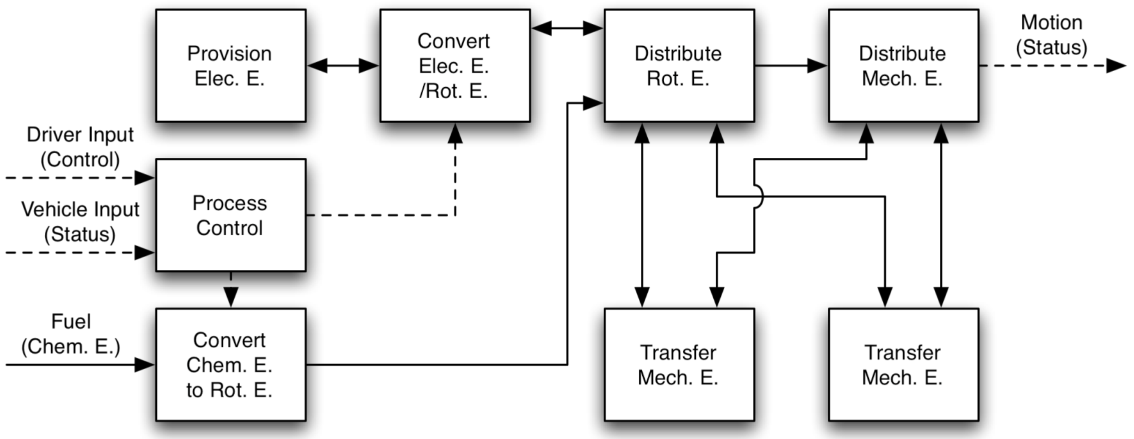


Fig. 3.4. Hybrid Powertrain Functional Model (Version 2)

3.3.2. Flow Variable Identification

The flow variable identification process for the Type I model appears and is described in Section 2. For the newly abstracted model developed for the Type II analysis, the same process was applied and resulted in the flow variable set and flow routing model shown in Table 3.8 and Fig. 3.5 respectively. A complete listing of the variables appearing the models shown in this section appears in Appendix 4.

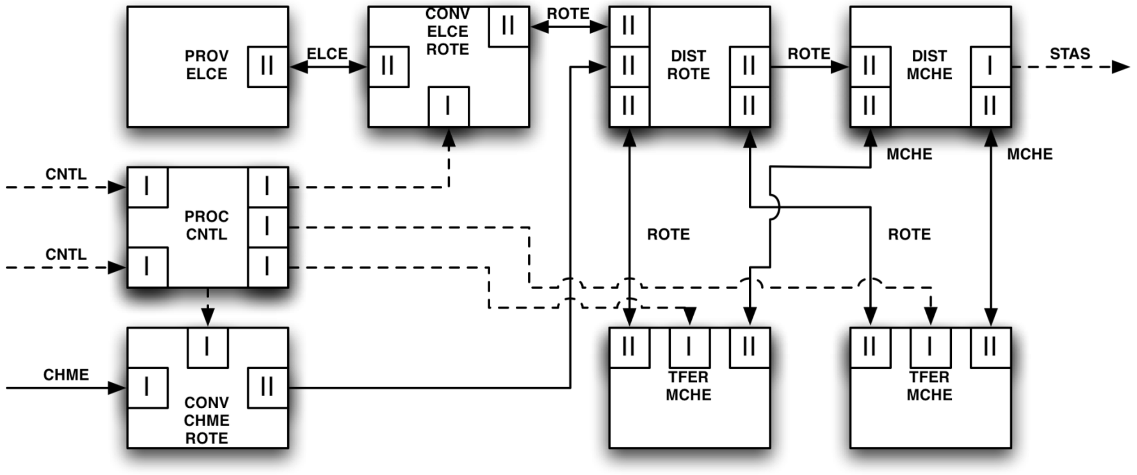


Fig. 3.5. Type 2 Model Flow Routing

As seen in Fig. 3.5, most of the flows in the model are of Type II. The only exceptions are the Type I flows used for the various signals passing through the system as well as the flow of chemical energy into the system. The chemical energy was left at Type I because a breakdown of the flow into constitutive elements was not required for the analysis to be performed (fuel energy delivery rate was the only piece of information required and could be found with a Type I flow).

Table 3.8. Type 2 Model Flow Variables

Flow	Symbol	Type	Units	Description
Electrical Energy	$ELCE_{IIa,x,0}$	Voltage	V	Effort
	$ELCE_{IIa,x,1}$	Current	A	Flow
Rotational Energy	$ROTE_{IIa,x,0}$	Moment	N*m	Effort
	$ROTE_{IIa,x,1}$	Ang. Vel.	rad/s	Flow
Mechanical Energy	$MCHE_{IIa,x,0}$	Moment	N*m	Effort, rot.
	$MCHE_{IIa,x,1}$	Force, x	N	Effort, trans.
	$MCHE_{IIa,x,2}$	Force, y	N	Effort, trans.
	$MCHE_{IIa,x,3}$	Ang. Vel.	rad/s	Flow, rot.
	$MCHE_{IIa,x,4}$	Velocity	m/s	Flow, trans.
	$MCHE_{IIa,x,5}$	Velocity	m/s	Flow. trans.

3.3.3. Model Elements

The next step in developing the behavioral models for the hybrid powertrain system was to create behavioral model elements for each functional element. An example of a Type I model element was shown earlier in Table 3.7. Model development was completed by using the flow variable information selected in the previous step along with the desired level of model fidelity to select a model type and class. The first model elements developed for the hybrid powertrain were targeted for a cursory analysis of the power demands of the system and were intended to be as form-neutral as possible to prevent preconceived ideas for the powertrain configurations from influencing the course of the design. The resulting model elements were developed based on the flow variables routed to each functional element and the relationships that could be developed based on these flows. As a result most of these relationships were derived from power conversion efficiencies and throttling in relation to control signals. The developed model elements (Table 3.9) are rather low in fidelity but allow the desired information from the model to be obtained with as little knowledge about specific form implementations as necessary. In this and subsequent tables, the following abbreviations are made: Imp. – Implicit, Exp. Explicit, Lin. – Linear, Alg. – Algebraic, Eq. - Equation, Hyb. – Hybrid, Diff. – Differential, Ord. – Ordinary. The equations are linked to the flow variables in each function as shown in Figs. 3.6-3.11.

Table 3.9. Type 1 Model Elements

Function	Model Description	Relationship	Details
Convert Chem. E. to Elec. E.	One-way power converter with linear proportional control	$P_E - P_F \cdot \xi_E = 0$	Imp. Lin. Alg. Eq.
		$P_E - C_E \cdot P_{E,\max} = 0$	Imp. Lin. Alg. Eq.
Provision Elec. E.	Energy storage and supply with single parameter efficiency. If power is flowing out of the element, the first equation is used, if power is flowing out the second is used.	$if(P_a) >= 0:$ $\dot{C}_A - \frac{P_a \cdot \xi_A}{E_{Max}} = 0$ $else:$ $\dot{C}_A - \frac{P_a}{\xi_A \cdot E_{Max}} = 0$	Imp. Hyb. Ord. Diff. Eq.
Convert Elec. E. to Rot. E.	Two-way power converter with linear proportional control and single parameter efficiency. The first equation relates power out to a control signal. The second equation relates input power to output power based on the sign of the control signal.	$P_M - C_M \cdot P_{M,\max} = 0$	Exp. Lin. Alg. Eq.
		$if(C_M) >= 0:$ $P_A - \frac{P_M}{\xi_M} = 0$ $else:$ $P_A - P_M \cdot \xi_M = 0$	Hyb. Imp. Alg. Eq.

Table 3.9 Continued.

Function	Model Description	Relationship	Details
Distribute Rot. E.	Power conservation	$P_M + P_E - P_D = 0$	Imp. Lin. Alg. Eq.
Distribute Mech. E.	Application of Newton's 2nd law	$\dot{v}_H - \frac{P_D - P_B}{v_H m_H} = 0$	Imp. Lin. Diff. Eq.
Transfer Mech. E.	Linear proportional power sink	$P_B - C_B \cdot P_{B,\max} = 0$	Imp. Lin. Alg. Eq.
Process Control	Simple hybrid controller with regenerative braking capability. Control error is based on a set speed versus distance spline.	$v_T = f(d_H)$	Cubic Spline Fit
		$\dot{d}_H = v_H$	Exp. Ord. Diff. Eq.
		$if\left(\left(v_T - v_H\right) > 0\right):$ $C_T = \min\left(\left(v_T - v_H\right)g_C, 1\right)$ $else:$ $C_T = \max\left(\left(v_T - v_H\right)g_C, -1\right)$	Hyb. Exp. Alg. Eq.
		$if\left(C_T > 0\right):$ $C_E = C_T$ $C_C = C_T$ $C_B = 0$ $else:$ $C_E = C_T$ $C_C = 0$ $C_B = abs\left(C_T\right)$	Hyb. Exp. Alg. Eq.

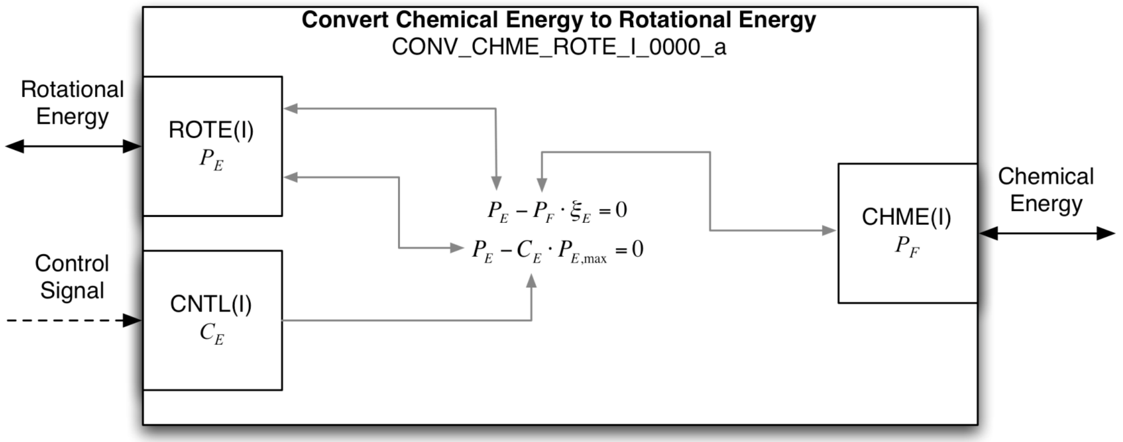
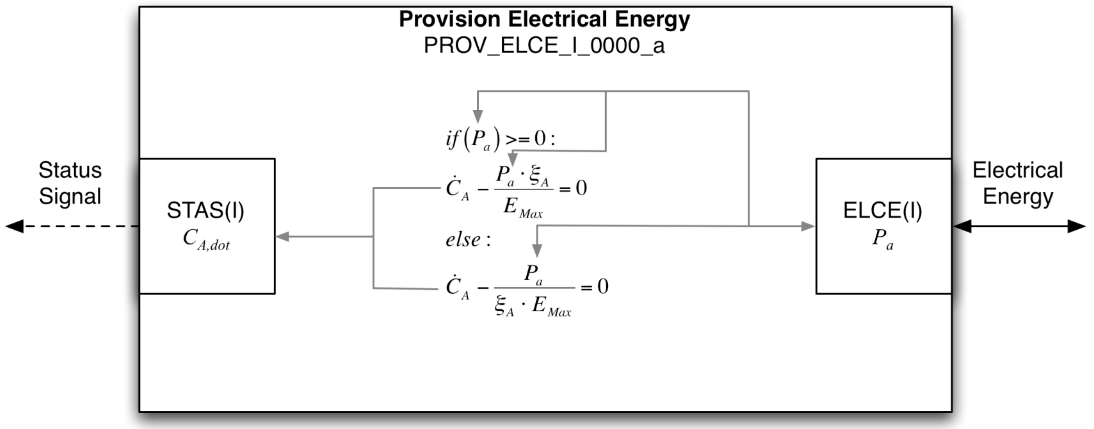
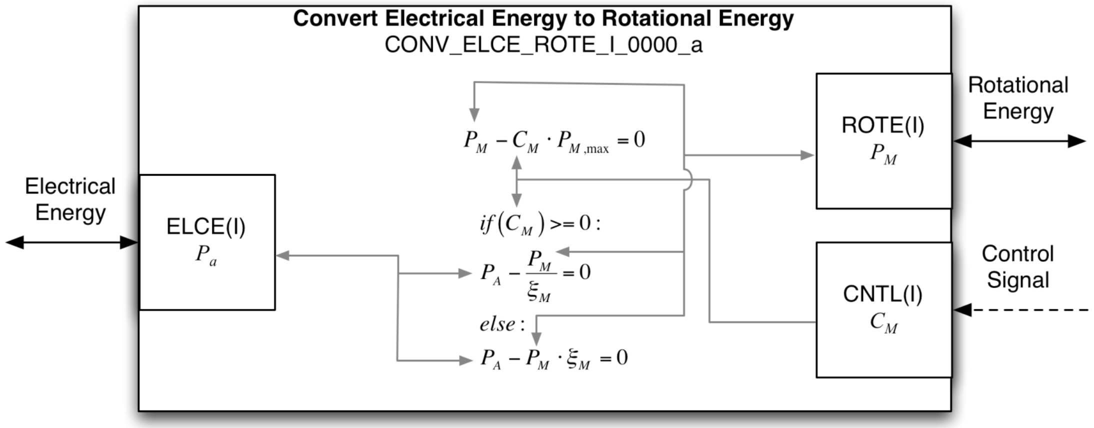


Fig. 3.6. Convert Chemical Energy to Rotational Energy Type I Model Element



Provision Electrical Energy
PROV_ELCE_I_0000_a
Status Signal
STAS(I)
$C_{A,dot}$
$if(P_a) >= 0:$
$\dot{C}_A - \frac{P_a \cdot \xi_A}{E_{Max}} = 0$
$else:$
$\dot{C}_A - \frac{P_a}{\xi_A \cdot E_{Max}} = 0$
ELCE(I)
P_a
Electrical Energy

Fig. 3.7. Provision Electrical Energy Type I Model Element



Convert Electrical Energy to Rotational Energy
CONV_ELCE_ROTE_I_0000_a
Electrical Energy
ELCE(I)
P_a
$P_M - C_M \cdot P_{M,\max} = 0$
$if(C_M) >= 0:$
$P_A - \frac{P_M}{\xi_M} = 0$
$else:$
$P_A - P_M \cdot \xi_M = 0$
ROTE(I)
P_M
Rotational Energy
CNTL(I)
C_M
Control Signal

Fig. 3.8. Convert Electrical Energy to Rotational Energy Type I Model Element

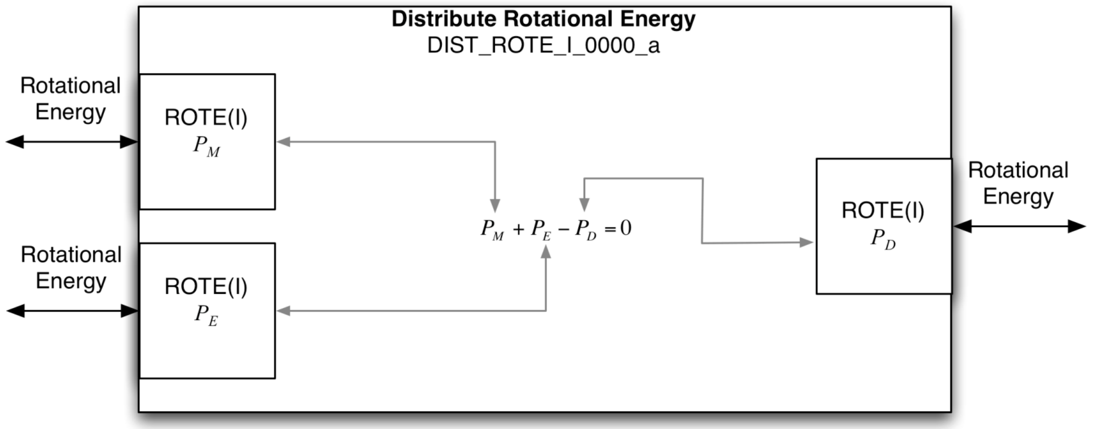


Fig. 3.9. Distribute Rotational Energy Type I Model Element

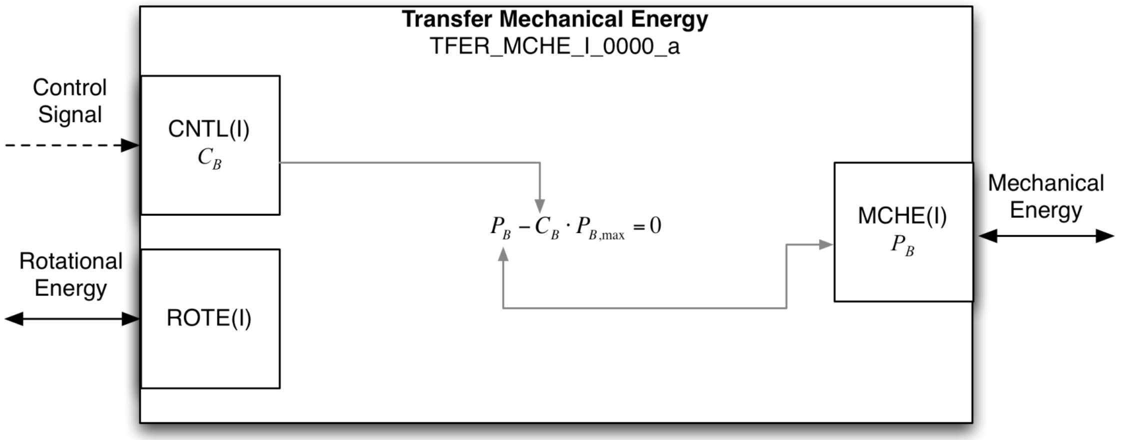


Fig. 3.10. Transfer Mechanical Energy Type I Model Element

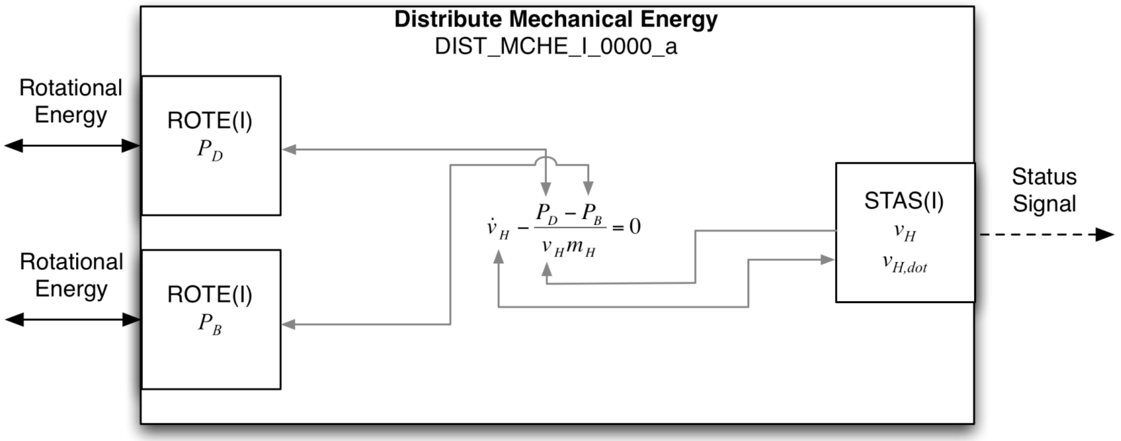


Fig. 3.11. Distribute Mechanical Energy Type I Model Element

Using the same approach but with the Type II flows shown in Table 3.8 and the slightly more detailed functional description in Fig. 3.5, Type II model elements were also developed for the powertrain system. These model elements operated on a larger flow variable set with the power flows in the Type I models broken down into flow and effort pairs in a similar manner to the approach used in Bond Graph modeling [17]. The model elements created based on these flows can generally be created using first principles from physics and resemble the traditional concept of component-based modeling. The increase in fidelity that comes from using such models comes at the expense of needing more information regarding the form solutions to the functionality of the system along with the extra information required to fully parameterize each model element during model use. As a result, these models are more appropriate for a coarse analysis of specific concepts during conceptual design rather than a high-level power flow analysis like the one enabled by the mainly Type I model. The Type II model elements appear in Table 3.10. The equations are linked to flow variables in Figs. 3.12-3.17.

Table 3.10. Type 2 Model Elements

Function	Model Description	Relationship	Type
Convert Chem. E. to Elec. E.	Spline fit to torque curve from engine dynamometer test with single parameter efficiency for fuel power calculation	$M_E = f(\omega_E)\cdot C_E$	Exp. Alg. Eq.
		$P_F - \frac{M_E \cdot \omega_E}{\xi_E}\cdot\frac{kW}{1000W} = 0$	Imp. Lin. Alg. Eq.
Provision Elec. E.	Linear battery model with internal resistance	$V_{B,nom} - V_B - I_B \cdot R_B = 0$	Imp. Lin. Alg. Eq.
		$\dot{C}_A = \frac{-I_B}{Q_{B,\max}}$	Exp. Lin. Diff. Eq.
Convert Elec. E. to Rot. E.	Constant torque to constant power transitional electrical machine model with linear proportional control	$\omega_T = \frac{P_{Max}\cdot V_S}{M_{Max}\cdot V_{Rated}}$	Exp. Alg. Eq.
		$if(\omega_M > \omega_T):$ $M_M - M_{Max}\cdot C_M = 0$ $else:$ $M_M - \frac{P_{Max}\cdot V_S \cdot C_M}{\omega_M \cdot V_{Rated}} = 0$	Hyb. Imp. Lin. Alg. Eq.
		$I_S - I_{Rated}\cdot C_M = 0$	Imp. Lin. Alg. Eq.
Distribute Rot. E.	Lossless post-transmission torque coupled rear wheel hybrid drivetrain with multi-ratio transmission for engine and fixed ratio motor reduction	$\omega_E = \omega_R \cdot X_G$	Exp. Alg. Eq.
		$\omega_M = \omega_R \cdot X_M$	Exp. Alg. Eq.
		$X_G = f(\omega_E)$	Exp. Alg. Eq.
		$M_M \cdot X_M + M_E \cdot X_G - M_R = 0$	Imp. Lin. Alg. Eq.
		$M_F = 0$	Imp. Lin. Alg. Eq.
		$M_{Rxn} - M_F - M_R = 0$	Imp. Lin. Alg. Eq.
Distribute Mech. E.	2D automotive chassis model with force and moment summations and no aerodynamic effects	$F_{F,z}\cdot(w_H - CG_x) - M_{F,y} - F_{R,z}$ $-M_{R,y} + M_{Rxn} + F_{R,x}\cdot CG_z + F_F$	Imp. Lin. Alg. Eq.
		$F_{F,x} + F_{R,x} - m_H \cdot \dot{v}_H = 0$	Imp. Lin. Diff. Eq.
		$-F_{F,z} - F_{R,z} + m_H \cdot g = 0$	Imp. Lin. Alg. Eq.

Table 3.10 Continued.

Function	Model Description	Relationship	Type
Transfer Mech. E.	2D paired wheel set with a Pacejka '96 longitudinal tire model and a linear proportional brake model	$\kappa_W = \frac{\omega_W \cdot r_W - v_W}{\lvert v_W \rvert}$	Exp. Alg. Eq.
		$F_{W,x} - 2 \cdot f\left(\kappa_W, \frac{F_{W,z}}{2}\right) = 0$	Imp. Non-Lin. Alg. Eq.
		$M_D - M_W - F_{W,x} \cdot r_W - J_W \cdot \dot{\omega}_W$	Imp. Lin. Diff. Eq.
		$M_W - M_{B,Max} \cdot C_B \cdot \mathrm{sgn}(\omega_W) = 0$	Imp. Non-Lin. Alg. Eq.
Process Control	Hybrid controller for RWD vehicle with regenerative braking and limited traction and anti-lock braking control. Control error is based on a set speed versus distance spline fit.	$v_T = f(d_H)$	Cubic Spline Fit
		$\dot{d}_H = v_H$	Exp. Ord. Diff. Eq.
		$if((v_T - v_H) > 0):$ $C_T = \min((v_T - v_H)g_C, 1)$ $else:$ $C_T = \max((v_T - v_H)g_C, -1)$	Hyb. Exp. Alg. Eq.
		$if(\kappa_R > \kappa_1 \;\; and \;\; \kappa_R < \kappa_2):$ $C_S = (\kappa_R - \kappa_1)/(\kappa_2 - \kappa_1)$ $else \;\; if(\kappa_R > \kappa_2):$ $C_S = 0$ $else;$ $C_S = 1$	Hyb. Exp. Alg. Eq.
		$if(C_T > 0):$ $C_E = C_T \cdot C_S$ $C_C = C_T \cdot C_S$ $C_B = 0$ $else:$ $C_E = C_T$ $C_C = 0$ $C_B = abs(C_T)$	Hyb. Exp. Alg. Eq.

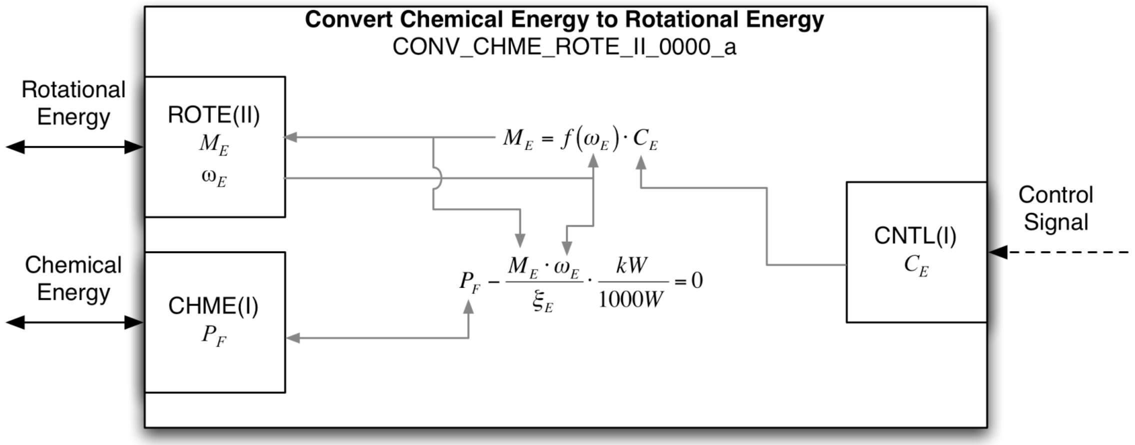

Convert Chemical Energy to Rotational Energy
CONV_CHME_ROTE_II_0000_a
Rotational Energy
ROTE(II)
M_E
ω_E
Chemical Energy
CHME(I)
P_F
$M_E = f(\omega_E) \cdot C_E$
$P_F - \frac{M_E \cdot \omega_E}{\xi_E} \cdot \frac{kW}{1000W} = 0$
CNTL(I)
C_E
Control Signal

Fig. 3.12. Convert Chemical Energy to Rotational Energy Type II Model Element

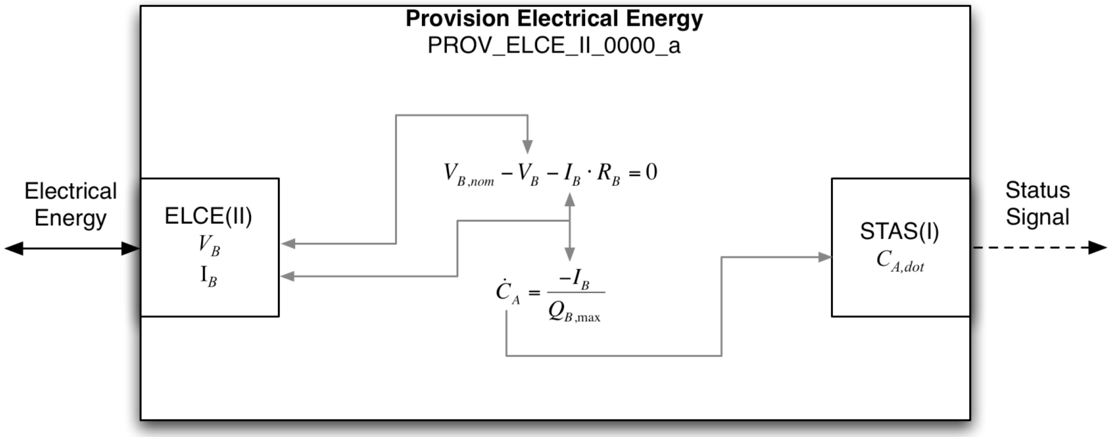

Provision Electrical Energy
PROV_ELCE_II_0000_a
Electrical Energy
ELCE(II)
V_B
I_B
$V_{B,nom} - V_B - I_B \cdot R_B = 0$
$\dot{C}_A = \frac{-I_B}{Q_{B,\max}}$
STAS(I)
$C_{A,dot}$
Status Signal

Fig. 3.13. Provision Electrical Energy Type II Model Element

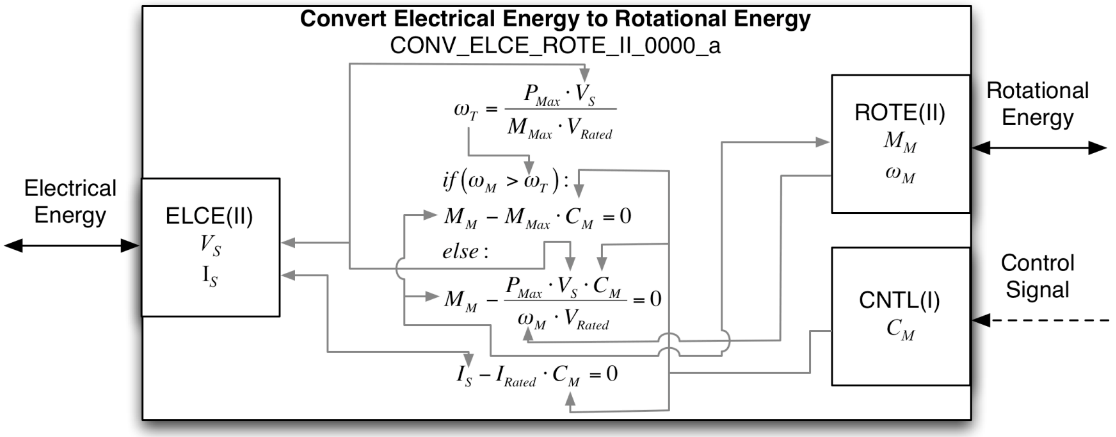

Convert Electrical Energy to Rotational Energy
CONV_ELCE_ROTE_II_0000_a
Electrical Energy
ELCE(II)
V_S
I_S
$\omega_T = \frac{P_{Max} \cdot V_S}{M_{Max} \cdot V_{Rated}}$
$if(\omega_M > \omega_T):$
$M_M - M_{Max} \cdot C_M = 0$
$else:$
$M_M - \frac{P_{Max} \cdot V_S \cdot C_M}{\omega_M \cdot V_{Rated}} = 0$
$I_S - I_{Rated} \cdot C_M = 0$
ROTE(II)
M_M
ω_M
Rotational Energy
CNTL(I)
C_M
Control Signal

Fig. 3.14. Convert Electrical Energy to Rotational Energy Type II Model Element

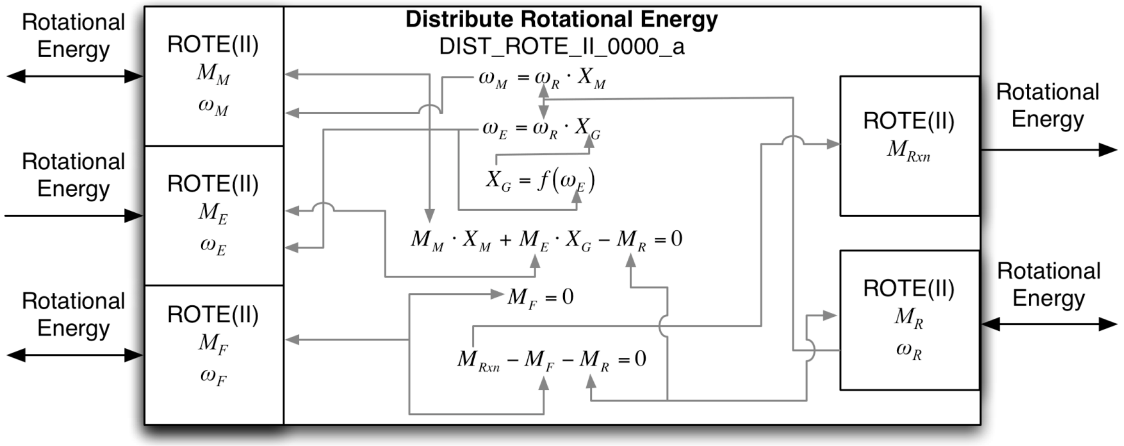

Distribute Rotational Energy
DIST_ROTE_II_0000_a
Rotational Energy
ROTE(II) M_M ω_M
Rotational Energy
ROTE(II) M_E ω_E
Rotational Energy
ROTE(II) M_F ω_F
$\omega_M = \omega_R \cdot X_M$
$\omega_E = \omega_R \cdot X_G$
$X_G = f(\omega_E)$
$M_M \cdot X_M + M_E \cdot X_G - M_R = 0$
$M_F = 0$
$M_{Rxn} - M_F - M_R = 0$
ROTE(II) M_{Rxn}
Rotational Energy
ROTE(II) M_R ω_R
Rotational Energy

Fig. 3.15. Distribute Rotational Energy Type II Model Element

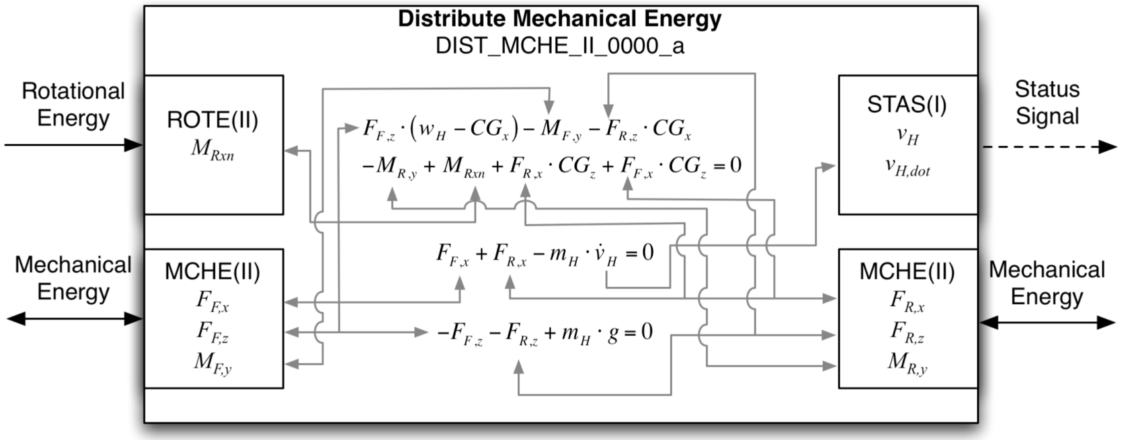

Distribute Mechanical Energy
DIST_MCHE_II_0000_a
Rotational Energy
ROTE(II) M_{Rxn}
Mechanical Energy
MCHE(II) $F_{F,x}$ $F_{F,z}$ $M_{F,y}$
$F_{F,z} \cdot (w_H - CG_x) - M_{F,y} - F_{R,z} \cdot CG_x$
$-M_{R,y} + M_{Rxn} + F_{R,x} \cdot CG_z + F_{F,x} \cdot CG_z = 0$
$F_{F,x} + F_{R,x} - m_H \cdot \dot{v}_H = 0$
$-F_{F,z} - F_{R,z} + m_H \cdot g = 0$
STAS(I) v_H $v_{H,dot}$
Status Signal
MCHE(II) $F_{R,x}$ $F_{R,z}$ $M_{R,y}$
Mechanical Energy

Fig. 3.16. Distribute Mechanical Energy Type II Model Element

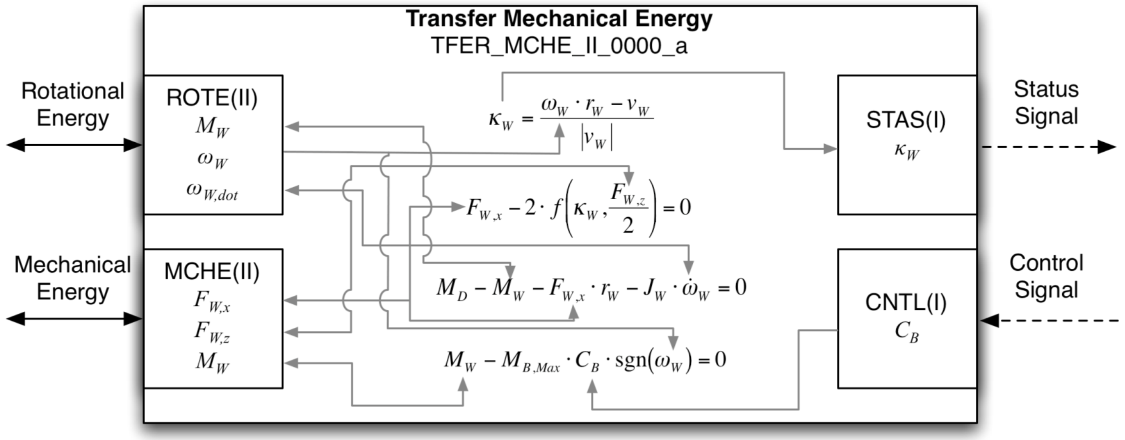

Transfer Mechanical Energy
TFER_MCHE_II_0000_a
Rotational Energy
ROTE(II) M_W ω_W $\omega_{W,dot}$
Mechanical Energy
MCHE(II) $F_{W,x}$ $F_{W,z}$ M_W
$\kappa_W = \frac{\omega_W \cdot r_W - v_W}{|v_W|}$
$F_{W,x} - 2 \cdot f\left(\kappa_W, \frac{F_{W,z}}{2}\right) = 0$
$M_D - M_W - F_{W,x} \cdot r_W - J_W \cdot \dot{\omega}_W = 0$
$M_W - M_{B,Max} \cdot C_B \cdot \text{sgn}(\omega_W) = 0$
STAS(I) κ_W
Status Signal
CNTL(I) C_B
Control Signal

Fig. 3.17. Transfer Mechanical Energy Type II Model Element

3.3.4. Model Solution

For both sets of models, the resulting equations form a set of hybrid differential algebraic equations (this is the usual case for a component-based modeling approach applied to an engineering system). There are multiple options for solving such problems. A direct approach to numerically solving such systems without preprocessing or an alternative representation generally fails [44]. Alternative representations include formatting the model elements to the standards of the Modelica [7] language and using a Modelica compiler and solver (such as Dymola) or reformatting the models as causal block diagram elements and implementing them in MATLAB/Simulink. Alternatively, the model elements could be analytically combined to produce a reduced set of equations that could then be implemented in an analytical solution. The appropriate representation and solution method depends on the intended use of the models. In the case of the hybrid powertrain model elements an extensive investigation of various powertrain configurations and an intensive optimization of powertrain performance with respect to concept selection and parameterization was required. As a result, a highly robust, highly modular, extremely computationally efficient and flexible model representation, assembly and solution approach was required. As a result, a custom model representation and assembly format was developed using C++ classes for the model elements and efficient parallel non-linear system solvers along with a Runge-Kutta 4 integrator. This approach was selected due to the ability to control every aspect of the solution and allow a completely modular and automated approach to investigating both parametric effects and model element choice and was possible due to the nature of the equations (index 0 non-linear DAEs). As the method proposed in this work primarily involves the relationship between functionality and behavioral model development, the details of the solution approach used are outside of the scope of this work and not included.

3.3.5. Model Results

For reference, results from an analysis of a single system configuration for both the Type I and II models are presented next. To use the models developed earlier, a full parameterization and model use case had to be developed for each. For each parameter shown for each model element in both sets of models, values were selected based on estimates of efficiencies (for the Type I models) and the availability of components (for the Type II models). These parameters are shown in Table 3.11 for the Type I elements and Table 3.12 for the Type II elements.

For each model, a simulated acceleration run from a standing start was performed for both electric only and electric plus internal combustion configurations of each (both represent Formula Hybrid events). In these events, time to complete the 75m run was recorded and the runs were terminated when vehicle displacement reached 75m. Autocross performance was also tested with the models. To simulate the powertrain demands for an autocross event, a data set from Texas A&M's 2008 Formula SAE entry over a complete autocross lap was used to generate a target speed profile (it was assumed the an optimal Formula Hybrid car would match the powertrain performance of a Formula SAE car). The powertrain models were then simulated using this target speed profile.

The results of the acceleration testing are shown along with the charge remaining in the powertrain's energy accumulators in Table 3.13. The autocross results are shown in Table 3.14. In this table, runs are shown with and without regenerative braking along with the actual time recorded from the 2008 TAMU car. Speed profiles from each of these autocross runs superimposed over the target speed profile appear in Fig. 3.18.

Table 3.11. Type I Model Parameters

Function	**Symbol**	**Value**	**Units**	**Description**
Convert Chem. E. to Rot. E.	ξ_E	0.30	[0,1]	Conversion Efficiency
	$P_{E,max}$	20.0	kW	Maximum Conversion Power
Provision Elec. E.	E_{max}	4000	kJ	Max. Energy Storage
	ξ_A	0.80	(0,1]	Provision Efficiency
Convert Elec. E. to Rot. E.	ξ_M	0.85	(0,1]	Conversion Efficiency
	$P_{M,max}$	10.0	kW	Maximum Power
Distribute Mech. E.	m_H	300.0	kg	Vehicle Mass
Transfer Mech. E.	$P_{B,max}$	120.0	kW	Max Braking Power
Process Control	g_C	10.0	s/m	Control Gain

Table 3.12. Type II Model Parameters

Function	Symbol	Value	Units	Description
Convert Chem. E. to Rot. E.	ξ_E	0.30	[0,1]	Conversion Efficiency
	F	Lookup	N*m	Based on WR250X Dyno
Provision Elec. E.	$V_{B,nom}$	72.0	V	Nominal Voltage
	R_B	0.1	□	Internal Resistance
	$Q_{B,max}$	60000.0	A*s	Battery Capacity
Convert Elec. E. to Rot. E.	V_{Rated}	48.0	V	Rated Voltage
	P_{Max}	13.41	kW	Maximum Power
	I_{Rated}	350.0	A	Rated Current
Distribute Rot. E.	f	Lookup	unitless	Fixed ratio gearbox model (5-speed)
Distribute Mech. E.	m_H	281	kg	Vehicle Mass
	CG_x	0.765	m	Vehicle CG Location, x
	CG_z	0.05	m	Vehicle CG Location, z
	w_H	1.7	m	Vehicle Wheelbase
	g	9.81	m/s^2	Gravitational Acceleration
Transfer Mech. E.	$M_{B,max}$	2000	N*m	Max Braking Moment
	r_W	0.254	m	Wheel Radius
	J_W	1.2	$kg*m^2$	Wheel Inertia, y
Process Control	g_C	1.0	s/m	Control Gain
	k_1	0.2	m/s	Slip Ratio Cut Start
	k_2	0.4	m/s	Slip Ratio Cut Stop

Table 3.13. Acceleration Results

Accel	Unrestricted		Electric Only	
Model	**Time**	**Charge**	**Time**	**Charge**
Type I	3.98s	99.1%	5.73s	98.6%
Type II	4.71s	97.25%	5.54s	96.8%

Table 3.14. Autocross Results

AutoX	With Regen.		No Regen.	
Model	**Time**	**Charge**	**Time**	**Charge**
FSAE	N/A	N/A	49.7s	N/A
Type I	50.64s	96.8%	50.64s	96.0%
Type II	51.87s	92.2%	51.87s	91.4%

To simulate an autocross start (a standing start), the initial speed for the simulated runs was set to 0m/s. Thus, a direct comparison between the hybrid cars and the convention car was not made but for reference, a simulated run of the car parameterized based on the 2008 TAMU car was performed using the Type II model with an initial speed that matched the initial speed from the target speed profile. The simulated car ran the lap in 49.84 seconds compared to the actual run of 49.7 seconds. It should be noted that the models, as developed, were strictly used for a powertrain capability analysis, for a full lap simulation analysis a more detailed model is required (a 3D type II model is appropriate for such an analysis and was developed using this approach in prior work for a conventional FSAE car [40]).

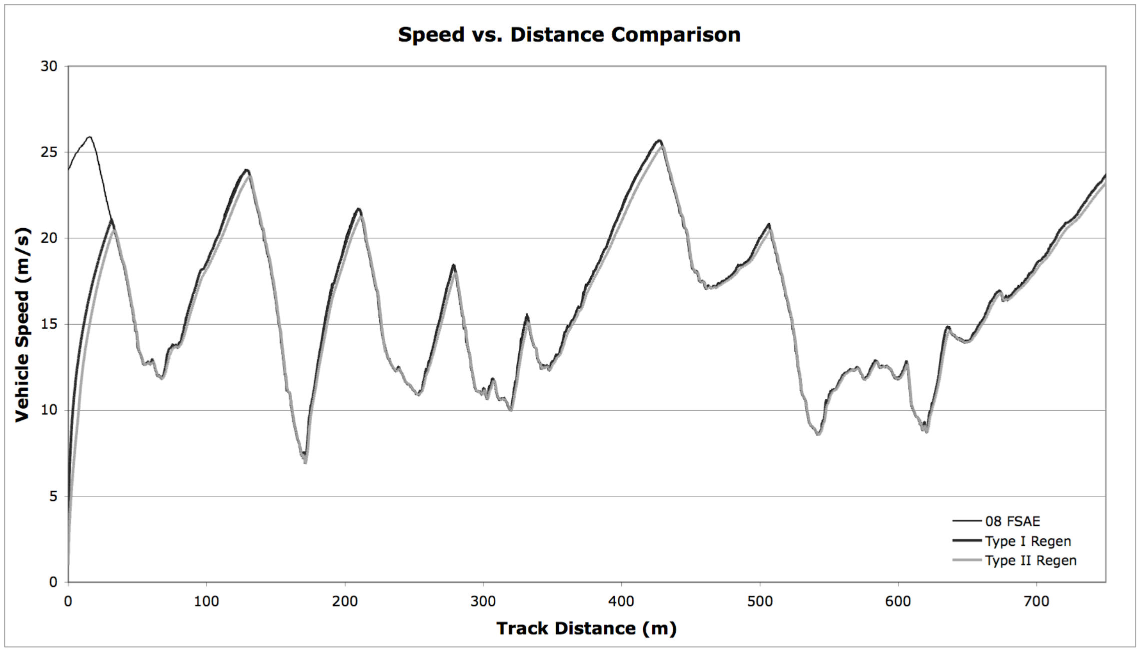


Fig. 3.18. Autocross Speed Profiles

By using a functional model to guide the behavioral modeling process for the hybrid powertrain system, it was possible to generate two models of very different fidelity within the same framework. The elements in the models had similar boundaries and flows from a functionality standpoint and used a typing system to define the variable set for each flow. The resulting model elements have a higher degree of modularity than a typical component based model. This modularity is improved through the use of common boundary flow types as well as the increased information on content and application resulting from the use of a formal typing system.

3.3.6. Discussion

The process of constructing a behavioral model based on a functional model is proposed to be completed in four steps: functional model creation, flow variable identification, model element identification and model solution along with any iterations required to arrive at the desired solution. The result of the application of this process is a behavioral model of a system that is tied to its functionality. Through example applications, driving the behavioral modeling of a system with a functional model has been found to provide the following contributions to system modeling:

- A link between behavioral model decomposition and functional decomposition,
- Potential reuse of behavioral model elements based on common functionality and established flow and model element types,
- Implementation of multi-fidelity modeling through the use of standard type definitions,
- Linking function-based component identification with quantitative component combination evaluation.

The first contribution results from the use of functional modeling to drive the behavioral modeling process. If implemented, the evolution of the functional model can in turn be used to assist the evolution of the representation of the system's behavior (as seen in the transition from the Type I to Type II model for the hybrid powertrain). The second contribution is enabled through the storage of behavioral model elements based on their associated function, form solution, and flow classification information. Once a complete behavioral model has been created for a system, model elements that were not re-used from prior solutions can be stored, along with their associated functions, form solutions and type information, plus the assumptions made during the model creation process, in a design repository such as the one demonstrated in [45]. When creating behavioral models for new systems, the functional model of the new system can be used to look up model elements and component solutions from prior designs based on common functionality and desired function and flow types. Even if the returned models are not compatible with the new system they may prove useful in assisting the model creation process for the new system.

The third contribution stems from the use of a formal typing system for flows and behavioral model elements. This typing system establishes common boundaries and a rough level of fidelity for a model element. These boundaries can then be used to develop higher or lower fidelity model elements within each function type given a set of connected flows and their corresponding types.

If a large set of existing model elements is available for the functions in a functional model and specific type level and the behavioral model elements for each function have compatible connecting flow types, the method allows the rapid development of component-based models for use in numerical explorations of the solution space for a system at varying levels of fidelity thus enabling the fourth contribution.

The method presented extends current component-based modeling methods by allowing an association between functionality and behavior through a structured process independent from the ultimate model representation and solution format. A major objective of the work is to not limit the approach to a specific model representation format and/or solution method. At a minimum, the approach provides a layer of functional abstraction over traditional component-based modeling techniques and assists the identification of variables, desired level of fidelity of elements and element re-use.

3.4. CONCLUSIONS

The objective of the work presented in the section was to investigate an approach for developing a complete behavioral model for a system based on a functional description. Specific benefits of the work include the assistance of concept selection through the use of a functional model as a unifying starting point for concept identification and evaluation, improved storage and re-use of behavioral model elements by primary association with functionality and typing information and integration of behavioral modeling decomposition with functional decomposition. Additionally, the method facilitates model element fidelity swapping and the storage of modeling knowledge.

The proposed method begins with functional modeling and includes flow variable identification and typing, model element identification and typing and finally model implementation. In the functional modeling step, the overall functionality and flows are identified along with a complete description of the system's detailed functionality.

During the flow variable identification step, potential model types are investigated and flow variables are identified based on the flows in the functional model and their selected types. Next, behavioral model elements are defined for each functional element in the functional model based on the intended use of the model and desired fidelity. The boundaries of these model elements correspond to those of the functions to which they are associated. The model elements themselves operate on the flow variables identified in the previous step. The models themselves form relationships between the input and output flow variables as identified from the flows in the functional model. The model can then be solved using the appropriate solution method given the type of models used and selected representation format. A complete example of the approach for two global levels of model fidelity was completed for a Formula Hybrid racecar powertrain. This example demonstrates the development of models within the approach at two levels of fidelity and proves that the approach can lead to the development of an accurate and useful model.

4. SENSITIVITY ANALYSIS IN EARLY DESIGN

During the conceptual design of engineering systems, investigating the sensitivity of a concept's performance with respect to parametric uncertainty is an important task. High sensitivity to parameters that are inherently noisy can lead to poor, or unexpected performance. However, parameters with high sensitivities can also be used as performance tuning variables later in the design process since changes in their values result in significant changes in performance. As a result, knowledge of the relative sensitivity of design parameters is as important as performance predictions when evaluating and selecting concepts during the early design process. Ideally, having this information as early as the conceptual design process can enable designers to make better choices during concept identification and selection. In order to quantify and mitigate (for noise) or utilize (for tuning) sensitivity during the design process, resources must be allocated to accurately identify and model its impact in a system. Identifying these effects as early as the conceptual design process allows better resource allocation throughout the entire design process. However, often during the conceptual design of a system, little is known about the potential physical forms of the solution. Without this information, it is difficult to define performance models and the probability distributions for the design parameters in these models.

The development of function-based behavioral modeling, as shown in Section 3, enables a function-based sensitivity analysis to be performed alongside the behavioral modeling of systems. In addition to the behavioral model re-use promoted by a functional decomposition, sensitivity information from previous design efforts can be re-used as well as providing the use appropriate sensitivity measures.

Another problem encountered in performing sensitivity analyses on models of complex systems during design is the associated computational burden. In general, local sensitivity analyses require at least two model evaluations per design variable in the model and global analyses using Monte Carlo methods require large numbers of evaluations (>>1000) to provide useful results.

In this work, it is proposed that the task of performing a sensitivity analysis during the design of a system be decomposed into two phases: a local sensitivity analysis for screening a large number of concepts during conceptual design and a global sensitivity analysis to be performed during the later stages of design. The approach is recommended due to the inherent differences in the information required to make decisions in early design versus the later stages of design. To facilitate the transition from a local analysis to a global analysis, it is recommended that a similar sensitivity analysis method be used in each.

Typically, a local analysis involves the use of derivative-based methods. For global analyses, variation-based methods are generally suggested [46-48]. In this work, a simplified local method similar to a variation-based approach is developed that allows a seamless transition from a local analysis of several concepts to a global analysis of a few. Since the problem of efficiently performing global variation-based sensitivity analyses has been well researched, most of the work presented here focuses on using a variation-like local method during the analysis of multiple conceptual solutions of a design problem. The primary novelty of this work lies in its use of a functional decomposition of the system, the behavioral model of the system and the associated sensitivity analysis.

The results of the work are presented in four subsections. The first sub section (4.1) details the reasoning behind and current methods for performing sensitivity analyses in design. The second sub-section (4.2) proposes a hybrid local sensitivity analysis derived from global variation-based methods. In the third sub-section (4.3), an example is presented that demonstrates how to apply the hybrid method to a functionally decomposed behavioral model (the method is certainly not exclusive to a functional decomposition but such a decomposition is helpful). Finally, conclusions and extensions of the work are presented in sub-section four (4.4).

4.1. SENSITIVITY ANALYSES IN ENGINEERING DESIGN

A sensitivity analysis is a study of how the variation of the inputs to a model create variation in the outputs of the model [47]. Such analyses can be qualitative or quantitative. A qualitative analysis identifies the relative importance of various design parameters to the overall sensitivity of the model. A quantitative analysis provides numerical measure of how sensitive the model is to variation of the design parameters. To perform a sensitivity analysis, a model of the system's behavior is required along with inputs (in the case of a design problem these are the design and noise parameters) and outputs (performance evaluations from the model).

Sensitivity analyses are used in a variety of fields in addition to engineering including the economic [49], environmental [50] and scientific [46] industries.

There are multiple methods available for performing a sensitivity analysis. A list of several methods currently used follows along with a characterization of these methods in Table 4.1.

Local Derivative (LD) - A sensitivity analysis approach characterized by the use of the local partial derivative of an output variable in response to an input variable of choice. The resulting measure carries units of the output variable divided by the input variable and does not require knowledge of input distribution or estimation of output distribution. Such approaches typically use two evaluation of the model for each parameter (at perturbed high and low values) to produce a second-order estimate of the local derivative [46].

Normalized Derivative (ND) – A normalization of a local derivative measure to the standard deviation of the input variable and output variable (other normalizations are sometimes used). This normalization produces a unitless measure and requires knowledge of the input variable's distribution and an estimation of output variable deviation. The results is a unitless signed measure [46].

Monte Carlo Regression (MCR) – A linear model fit to the results of a Monte Carlo simulation of the system. This approach requires full knowledge of input variable distributions and is significantly more computationally complex compared to a local derivative approach as it requires numerous evaluations of the model (>>1000) for each parameterization. The resulting measures provide a global breakdown of sensitivity contribution [46].

Variance-based (VB) – A sensitivity analysis performed using estimates of model variance and parametric contributions to variance. Analogous to MCR under certain conditions and has similar characteristics [46].

Simplified model fit (SMF) – A more computationally efficient model (such as a kriging model [51]) is fit to the system performance model. A sensitivity analysis is then performed on the resulting model. This approach shares characteristics of the variation-based analysis approach used but with a decrease in computational burden at the expense of an increase in model uncertainty (inherent in the model fitting process). In certain cases analytical solutions are possible [48].

Table 4.1. Characterization of Sensitivity Analysis Methods

	Method				
Characteristic	LD	ND	MCR	VB	SMF
Computationally efficient	X	X			X
Works directly on model	X	X	X	X	
Does not require add. info.	X				
Unitless		X	X	X	X
Contributive measure			X	X	X
Local/Global	L	L	G	G	G

During the design of engineering systems, the following consideration must be made when selecting a sensitivity analysis:

- Local information versus global information [48, 49, 52],
- Computational burden [48, 51, 52],
- Knowledge of design parameter distribution and distribution parameters,
- Usefulness of resulting knowledge,
- Modeling requirements.

In the context of conceptual design, behavior and sensitivity information can be used as selection criteria when evaluating various concepts. Within the context of engineering design, a sensitivity analysis can provide useful information in the form of sensitivity magnitude (quantitatively, how sensitive is a system?) and sensitivity contribution (what is the relative contribution of each variable to the system's sensitivity?). Variation-based sensitivity measures provide both pieces of information in the form of the overall variation of the model along with main effect contributions.

During conceptual design, limited information is available about potential concepts. This limited information may exclude knowledge of parametric distributions. As a result, it may not be possible to perform a full variation-based sensitivity analysis for these concepts. Additionally, such analyses are computationally expensive (even with the more efficient approaches discussed in [48, 52]). Local measures of sensitivity (derivative-based approaches) provide quantitative measures of each variable's effect on the system's performance at a single nominal operating point but cannot be used in a contributive manner or provide an overall sensitivity magnitude like a variation-based approach. However, unlike variation-based measures, local measures do not require specific information about parametric distributions and are generally much faster to evaluate computationally (typically, two performance evaluations are required for each variable considered in the sensitivity analysis). Since the results of a typical local analysis are derivatives (normalized or not) and the result of a variation-based analysis is a set of main effect contribution percentages and an overall variance, the results of the two analyses are not compatible.

To reconcile the various approaches to performing a sensitivity analysis in conceptual design, a hybrid local variation-based (HyVar) analysis is proposed. This analysis uses the basic mechanics and output information of a traditional sample-based variation-method and combines it with the cost of evaluation of a local derivative method. Essentially, this approach replaces a derivative-based local measure with a variation-based measure. In this approach, derivative information as well as a variance-like sensitivity magnitude along with main effect contributions are calculated for a nominal parameterization of each concept. The resulting measures provide the same results of a local derivative approach along with measures similar to that of a full variation-based analysis. The analysis is still a local analysis and does not replace full variation-based global sensitivity analysis but requires significantly less knowledge about parameter distributions and is suitable for screening a large number of parameterized concepts during the early stages of design.

The benefits of using the HyVar approach over local derivative methods are the compatibility of the results of the HyVar analysis with a full variation-based method while retaining a similar computational burden to a local method. Due to this reduced computational burden, it is feasible to operate on an existing model rather than use a fitting process (polynomial or kriging). Additionally, since the output parameters from the hybrid analysis have the same format as the output parameters of a full variation-based approach, the result of the two analyses can be directly compared. An example of where such a comparison would be useful is between a HyVar analysis performed during conceptual design versus a full variation-based approach performed on the same model during detailed design once parametric distributions are known.

The sensitivity measures provided by the proposed HyVar approach allow grouping of sensitivity contributions by direct addition of percentages. This is a primary reason for implementing them over a traditional derivative-based approach and allows the approach to be used in a functionality decomposed system behavior model. A derivation of these measures with an illustration of how the measures are combined to represent the sensitivity contribution of functions in presented in the next section.

4.2. HYBRID VARIATION-BASED LOCAL SENSITIVITY MEASURES

By applying the same basic calculations used in a traditional variation-based global sensitivity analysis to a local analysis, it is possible to get contribution measures of a system with the same computational burden as a traditional derivative based measure. While the approach still results in a local measure (distributions of parameters are not used) of sensitivity and carries the same caveats of a local approach, it does provide contribution measures that are not found with a derivative-based analysis. The format of these measures is compatible with those of a full variation-based approach (the output of both approaches is a set of sensitivity percentages that correspond to the contribution to total variation for each parameter).

In general, the variance of a finite population can be calculated using the formula shown in Equation 4.1. In the context of the results of a system performance model, Y_i would be the model output for a particular parameterization of the model and $\bar{Y}$ would be the mean value of the outputs of all parameterizations performed (up to N total).

$$\sigma^2 = \frac{1}{N}\sum_{r=1}^{N}\left(Y_r - \bar{Y}\right)^2 \quad (4.1)$$

For a complex system behavioral model that has been decomposed based on functionality, the input parameters will be grouped according to function and it is possible that the performance output might not be a single output variable (one model may produce several performance outputs). In this case, the output is a vector rather than a scalar and the parametric inputs to the model are a series of vectors (one for each function in the model). A mathematical representation of such a model appears in Equation 4.2 where Y_i is a specific row in the performance vector and $f_i\left(\vec{x}_1,\vec{x}_2\ldots\vec{x}_i\ldots\vec{x}_f\right)$ represents an evaluation of that performance by the set of input vectors $\vec{x}_i$ associated with each function up to a number of F total functions.

$$Y_i = f_i\left(\vec{x}_1,\vec{x}_2\ldots\vec{x}_i\ldots\vec{x}_F\right) \quad (4.2)$$

In the context of a design problem, a concept represents a unique parameterization of a model. For each parameter in the parameter vectors for each function in a concept, a slightly higher or lower value of the parameter can be substituted for the nominal value (as is done in partial differencing in a derivative-based local sensitivity analysis). For each case (high and low), the output predicted by the model can be compared to the nominal value of performance found using the nominal parameterization for the concept. This comparison can be made using the same basic calculation as the variance of a population where the population mean is replaced with the nominally predicted performance of the concept. This comparison can be performed for each parameter in each function for each performance output. The mathematical representation of this comparison appears in Equation 4.3. A discussion of the perturbation size and its affect on results appears in Section 4.3.

$$V_{i,j,k} = \frac{\left(Y_{i,x_{j,k+}} - Y_i\right)^2 + \left(Y_{i,x_{j,k-}} - Y_i\right)^2}{2} \quad (4.3)$$

In this equation, the index *i* represents the performance output, the index *j* represents the function and the index *k* represents the local parameter in the function *j*. Since two comparisons are made for each of the high and low values, the squared-deviation from the nominal performance is divided by 2. The result. V_{ijk}, is a measure of the average squared deviation from nominal performance found from perturbing the parameter of interest. For each function, these deviations can be summed as shown in Equation 4.4. In this equation, **P** is a vector that contains the number of local parameters for each function.

$$V_{i,j} = \sum_{k=1}^{P_j} V_{i,j,k} \quad (4.4)$$

The result is a measure of the deviation in performance produced by a single functional element of the model. These functional deviations can then be summed as well (Equation 4.5) to produce a total measure of deviation in the model resulting from the parametric perturbations.

$$V_i = \sum_{j=1}^{f} V_{i,j} \quad (4.5)$$

Like in a variation-based sensitivity analysis, the various deviations of each parameter and each function can then be divided by the total model variation to produce a percentage measure of contribution to the total deviation (Equations 4.6 and 4.7). The resulting measures, S_{ij} and S_{ijk} represent the relative contribution to the total deviation of the performance (i) for a specific function (j) and a parameter within a function (k). The measure S_{ijk} corresponds to the main effect sensitivity contribution as calculated in a full-variance based approach. The S_{ij} measure is a newly proposed measure that relates the role a single grouping of parameters based on the functional decomposition of the model. The approach can be applied to a system without a functional decomposition by assuming a single functional element (F=1) with all model parameters in that functional element.

$$S_{i,j} = \frac{V_{i,j}}{V_i} \quad (4.6)$$

$$S_{i,j,k} = \frac{V_{i,j,k}}{V_i} \quad (4.7)$$

If the functions in a system have a relatively equal contribution to the overall variation of the system, the *S* measure for each function should be approximately *1/F* where *F* is the number of functions in the system. As a result, a direct comparison between the sensitivity contribution of a function that appears in both large and small systems (large and small values of *F* respectively) cannot directly be made. To normalize the sensitivity contribution measures to allow such comparisons, the *S* measure should be multiplied by *F*, the number of functions in the system, to produce a normalized unitless ratio of sensitivity (Equation 4.8). A value of SR_{ij} equal to 1 indicates a sensitivity contribution of *1/F* for function *j* with respect to performance variable *i*. This indicates that the function has a sensitivity contribution equal to its functional contribution.

Values greater than 1 indicate a relatively higher contribution to sensitivity than the contribution to functionally. The opposite is true for values less than one. This measure allows a particular function's tendency to be over- or under-sensitive to be characterized outside of the context of the particular model or concept being studied. For systems with more than one performance variable, the sensitivity ratio can be averaged per Equation 4.9 where Z is the number of performance variables considered in the analysis.

$$SR_{i,j} = S_{i,j} \cdot f \quad (4.8)$$

$$SR_j = \sum_{i=1}^{Z} SR_{i,j} \quad (4.9)$$

The use of these measures on a functionally decomposed behavioral model along with a discussion of the results appears in the example presented in the following section.

4.3. EXAMPLE

To illustrate the application of the hybrid variation-based local (HyVar) sensitivity analysis, an example is presented based on a simple three-function system. The system represents a simple combination of an electrical power source, an electrical-to-rotational energy conversion and a rotational load. This system is represented with a formal functional model in Fig. 4.1. In this model, the three functional elements of the system are *provision electrical energy*, *convert electrical energy to rotational energy* and *export rotational energy*. The *provision* function includes a charge *status signal* output and the *convert* function includes a throttle *control signal*.

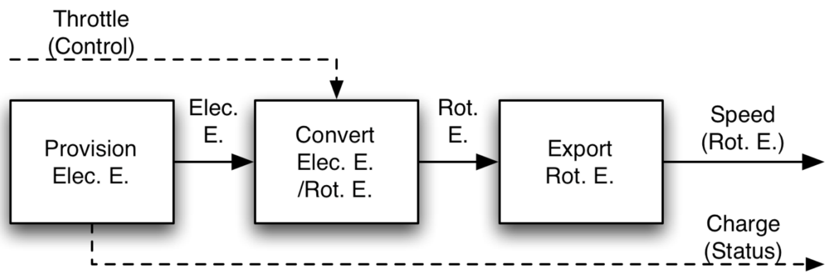


Fig. 4.1. Example Functional Model

As per the functionality-based behavioral modeling approach presented in Section 3, a flow routing map was created based on this functional model and is shown in Fig. 4.2. The first two functions are based on the hybrid automotive models appearing in Section 3.3.2 and the final function (*export rotational energy*) was created specifically for this example with a compatible interface to the energy conversion function.

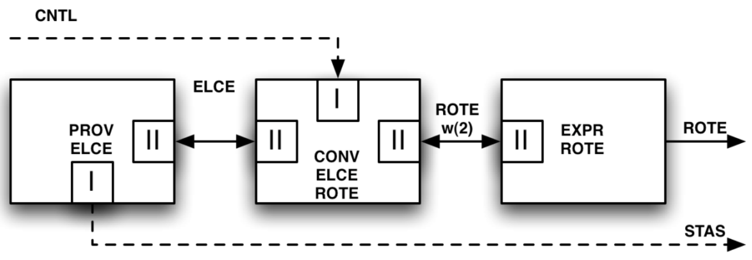


Fig. 4.2. Example Flow Routing and Type Model

The model elements selected for each functional element are of Type II (tied to form family solution) as described in 3.1.2.3. For reference, the FourCC codes used to briefly and uniquely described the functions and flows used in this work appear in Table 4.2. A description of the flow variables used in this example appears in Table 4.3 along with the mathematical relationships used in the behavioral model elements for each function in Table 4.4. Since this is a Type II analysis, form family solutions are required for each function. For the provision function, a chemical battery was selected; the energy conversion function was satisfied by an AC induction motor and the export function by a simple rotational load with inertia and friction. The motor and battery parameterizations and modeling are based on model elements used in the analysis of a hybrid Formula SAE racecar. The load model was derived and parameterized to interact with these components.

Table 4.2. Example FourCC Codes

Flow	FourCC	Function	FourCC
Control	CNTL	Convert	CONV
Electrical	ELCE	Distribute	DIST
Rotational	ROTE	Export	EXPR
Status	STAS		

Table 4.3. Model Flow Variables

Flow	Symbol	Type	Units	Description
Electrical Energy	$ELCE_{IIa,x,0}$	Voltage	V	Effort
	$ELCE_{IIa,x,1}$	Current	A	Flow
Rotational Energy	$ROTE_{IIa,x,0}$	Moment	N*m	Effort
	$ROTE_{IIa,x,1}$	Ang. Vel.	rad/s	Flow
Control Signal	$CNTL_{Ia,x,0}$	Analog	Varies	A single analog control signal
Status Signal	$STAS_{Ia,x,0}$	Analog	Varies	A single analog status signal

Table 4.4. Model Relationships

Function	Model Description	Relationship	Type
Provision Elec. E.	Linear battery model with internal resistance	$V_{B,nom} - V_B - I_B \cdot R_B = 0$	Imp. Lin. Alg. Eq.
		$\dot{C}_A = \frac{-I_B}{Q_{B,\max}}$	Exp. Lin. Diff. Eq.
Convert Elec. E. to Rot. E.	Constant torque to constant power transitional electrical machine model with linear proportional control	$\omega_T = \frac{P_{Max} \cdot V_S}{M_{Max} \cdot V_{Rated}}$	Exp. Alg. Eq.
		$if\left(\omega_M > \omega_T\right):$ $M_M - M_{Max} \cdot C_M = 0$ $else:$ $M_M - \frac{P_{Max} \cdot V_M \cdot C_M}{\omega_M \cdot V_{Rated}} = 0$	Hyb. Imp. Lin. Alg. Eq.
		$I_S - I_{Rated} \cdot C_M = 0$	Imp. Lin. Alg. Eq.
Export Rot. E.	Simple inertial and frictional load	$M_L - J_L \cdot \dot{\omega}_L - B_L \cdot \omega_L = 0$	Imp. Lin. Diff. Eq.

The function-based behavioral model assembly and solution process demonstrated in Section 3 was then used to create a well-posed model of the system that was capable of predicting the time required for the rotational load to reach a prescribed target speed from a standing start. Essentially, the model represented a battery, connected to a motor connected to a flywheel with a linear velocity-dependant friction force. Although this is a relatively simple system, it provides a useful illustration of how to apply the HyVar analysis presented earlier in the work. A more thorough example applied to a larger system is presented in the example shown in Section 6. Each functional element in the model has a set of parameters associated with it. These parameters are tied to the mathematical relationship selected to represent the function and are used to establish and distinguish various conceptual solutions to the system. A nominal set of parameters was selected to represent a single concept and is shown in Table 4.5.

Table 4.5. Model Parameters

Function	Symbol	Value	Units	Description
Provision Elec. E.	$V_{B,nom}$	72.0	V	Nominal Voltage
	R_B	0.1	W	Internal Resistance
	$Q_{B,max}$	60000.0	A*s	Battery Capacity
Convert Elec. E. to Rot. E.	V_{Rated}	48.0	V	Rated Voltage
	P_{Max}	13.41	kW	Maximum Power
	I_{Rated}	350.0	A	Rated Current
Export Rot. E.	B_{Load}	2.0	N*m*s/rad	Load Friction Constant
	J_{Load}	6.0	kg*m^2	Load Inertia

Based on this nominal parameterization, a nominal performance vector can be calculated from the assembled system behavioral model. In this example, one performance variable is considered (time for the flywheel to accelerated to 200 rad/s).

Using this model, a variety of sensitivity analyses can be performed. In this example, a derivative-based local analysis is performed along with the HyVar method presented in Section 4.2. Additionally, a full variation-based Monte Carlo sensitivity analysis is performed as well with an assumed set of parametric distributions for the variables in each function.

The results of these analyses appear in Tables 4.6, 4.7 and 4.8. Table 4.6 contains the results of three different applications of the HyVar method at three different variable perturbations (0.10%, 1.0% and 10%). Using a constant perturbation percentage for each variable represents a constant coefficient of variation in a full variation-based approach. Due to the significant reduction in evaluation cost of the HyVar analysis versus a full global analysis, it is recommended that a variety of perturbation steps be used. Smaller steps capture local effects better than larger steps but may not capture behavior that occurs further from the nominal performance. Performing three evaluations with three steps sizes varying by an order of magnitude provides a large range of coverage around each design variable. If significant variation in results occurs between the three step sizes, it is recommended to perform a full Monte Carlo sensitivity analysis. At each percentage, the sensitivity contribution, derivatives and normalized derivatives (normalized to the magnitude of the input variable perturbation) are tabulated along with the time required to perform the analysis and the nominal performance. As shown in this table, the most significant contribution to overall sensitivity in performance of the system is the nominal voltage of the battery. This is followed by the inertia of the load then the rated power of the motor, rated torque of the motor and finally the friction of the load. The results of this analysis are charted in Fig. 4.3.

Table 4.6. HyVar Results

	Parameter Offset Percentages								
Variables	0.10%			1%			10%		
Nominal Voltage	44.33%	-0.0239	-0.0034	44.36%	-0.0239	-0.0344	49.77%	-0.0260	-0.3738
Rated Power	16.94%	-3.11E-05	-0.0021	16.93%	-3.11E-05	-0.0213	16.15%	-3.20E-05	-0.2194
Rated Torque	8.42%	-0.0053	-0.0015	8.41%	-0.0053	-0.0150	7.92%	-0.0054	-0.1544
Load Friction	3.31%	0.2354	0.0009	3.31%	0.2352	0.0094	2.88%	0.2361	0.0945
Load Inertia	27.00%	0.2242	0.0027	26.99%	0.2240	0.0269	23.27%	0.2240	0.2688
Analysis Time (s)	0.0327			0.0338			0.0329		
Nominal Perf (s)	1.3338			1.3338			1.3338		

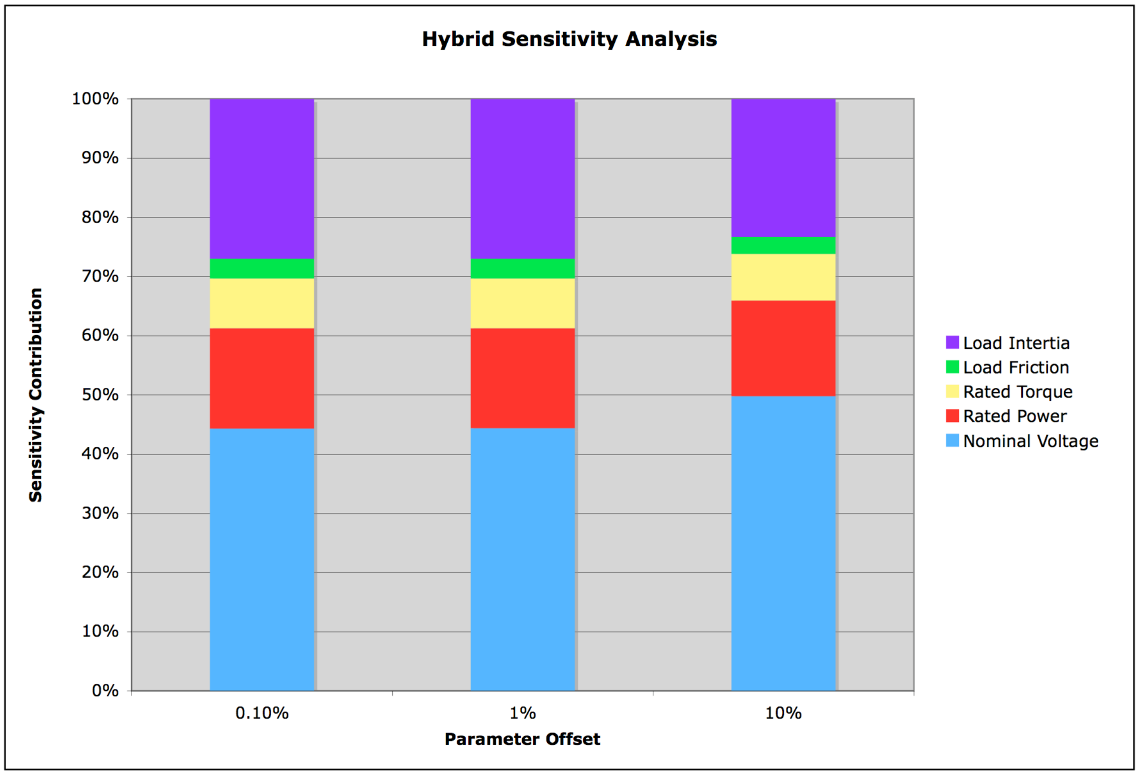


Fig. 4.3. HyVar Results Charted

Table 4.7 shows the tabulated results of the HyVar sensitivity analysis at a 1% perturbation using the nomenclature and grouping established in Section 4.2. As shown in this table, the *provision electrical energy* function contributes 44.4% to the overall sensitivity of the system followed by the *export rotational energy* function at 30.3% and then the *convert electrical energy to rotational energy* function at 25.3%. Once normalized to the number of functions in the system, the sensitivity ratios for these functions become 133.09%, 76.02% and 90.89% respectively. Thus, the *provision electrical energy* function provides a larger contribution to overall sensitivity than it does to functionality. The opposite is true with respect to the *convert electrical energy to rotational energy* and *export rotational energy* functions. A sensitivity contribution greater than one indicates that the parameters in the model of the functional element affect the performance of the system to a greater degree than the parameters in other functional elements in the system. Depending on the design problem, this information may be used for a variety of purposes. If the overall contribution of each functional element is desired to be equal, then the sensitivity ratios of each function should be driven to values around 1.0. If certain functions are desired to contribute more to the overall performance of a system, the sensitivity ratio of these elements should be driven to values greater than 1.0. If the opposite is true, and certain elements are desired to contribute less to the overall performance, the sensitivity ratios of these elements should be driven to values less than 1.0. The form of the measures (numbers with values around 1.0) allows their direct use in an optimization framework (a fitness function in a genetic algorithm for instance). Thus, if a specific sensitivity ratio profile is desired, a formal optimization problem can be set up to use the measures. This allows not only the optimization of performance (as typical in an engineering optimization problem) and overall sensitivity (total variation) but also the sensitivity profile of the functional elements in the system.

Table 4.7. Sensitivity Parameters

Contributor	Label	Value
Nominal Voltage	$S_{1,1,1}$	44.36%
Rated Power	$S_{1,2,1}$	16.93%
Rated Torque	$S_{1,2,2}$	8.41%
Load Friction	$S_{1,3,1}$	3.31%
Load Inertia	$S_{1,3,2}$	26.99%
Provision Elec. E.	$S_{1,1}$	44.36%
Convert EE to RE	$S_{1,2}$	25.34%
Export Rot. E.	$S_{1,3}$	30.30%
Provision Elec. E.	$SR_{1,1}$	133.09%
Convert EE to RE	$SR_{1,2}$	76.02%
Export Rot. E.	$SR_{1,3}$	90.89%

For reference, a full variation-based Monte Carlo sensitivity analysis was performed as well at a variety of sample sizes. The SA method presented in [47] was used to complete this analysis. In order to perform the analysis, parametric distributions for the variables in the model were required. As mentioned before, these distributions are not commonly available or applicable in early design but are necessary to apply a full global variation-based approach. As seen in Table 4.8, for small numbers of samples (<<1000) the results are unreliable (variations should not have negative values). At a sample size of 1000, the results begin to stabilize and after 10000 samples the precision of the analysis improves. However, this increased precision requires a significant computational burden as seen by the linearly increase in analysis time. The resulting sensitivity contribution measures are charted in Fig. 4.4. The relative contribution of each parameter differs from that of the HyVar analysis but that is to be expected as the two analyses are working on separate sources of input (the full analysis considers the actual distribution of each parameter versus the HyVar analysis that considers a perturbation of each parameter). The primary result of this analysis is the relative magnitude of analysis time required for each method. The HyVar method required 0.0338 seconds versus 184.2 seconds for a reliable full Monte-Carlo analysis. It should be noted that the HyVar method is not intended as a replacement for a global analysis but rather a complement for the early stages of design where such a large computational burden (even with efficiency improvements demonstrated by [48] and [52]) and additionally required information are not practical. As mentioned before, the HyVar analysis provides the same type of information as a global analysis (relative contribution) but uses a method that is feasible for use in early design when considering multiple concepts.

Table 4.8. Global Variation-based Results

	Monte Carlo Samples			
Variables	10	100	1000	10000
Nominal Voltage	87.18%	61.69%	62.02%	63.63%
Rated Power	-6.58%	-3.73%	1.04%	0.58%
Rated Torque	33.76%	4.41%	7.69%	8.20%
Load Friction	-7.83%	0.40%	8.63%	7.02%
Load Inertia	-28.66%	17.46%	25.34%	22.48%
Analysis Time (s)	0.1886	1.8039	18.20	184.2
Mean Perf. (s)	1.3498	1.3482	1.3467	1.3457
Variance (s^2)	0.0039	0.0078	0.0095	0.0093

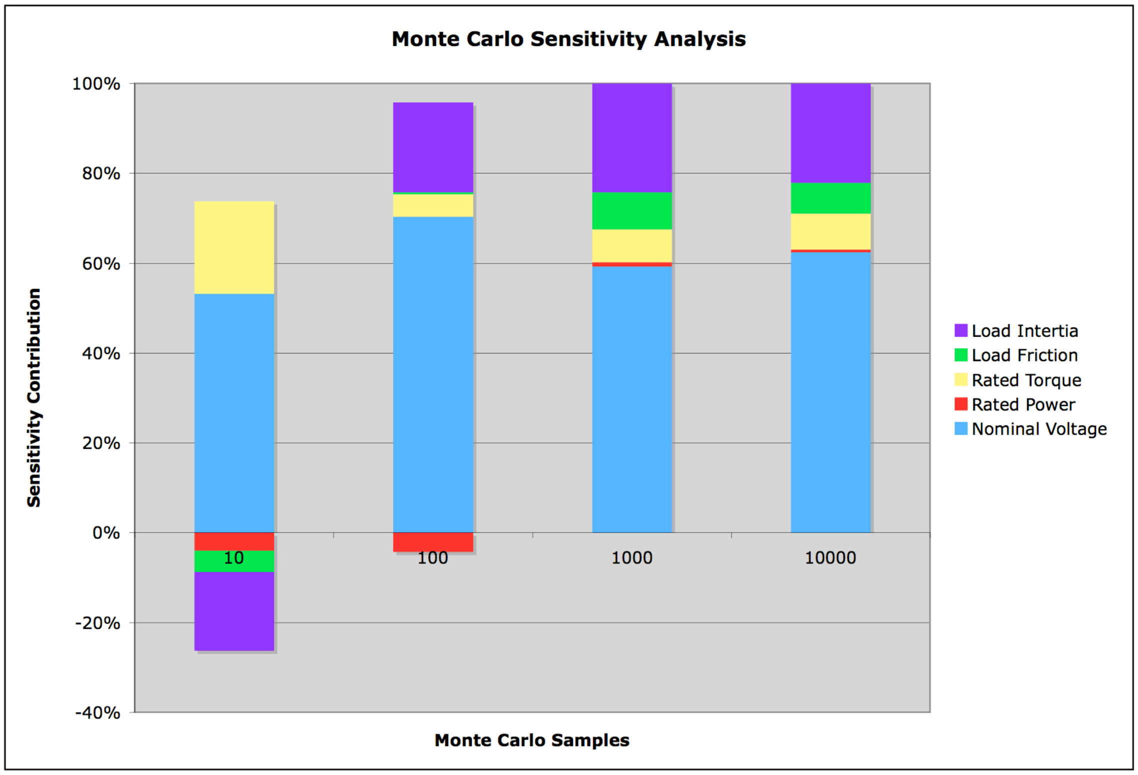


Fig. 4.4. Global Variation-based Results Charted

4.4. CONCLUSIONS

The HyVar sensitivity analysis approach presented in this work is a local sensitivity analysis method that uses an approach similar to a full variation-based analysis. The method provides the same qualitative measures as the full approach with the computational burden of a derivative-based local approach. This approach provides a bridge between a local analysis of many concepts during early design and a global analysis of a smaller set of concepts later in the design process. The approach does not require detailed information regarding the distributions of the various parameters used in a model of a concept and thus is appropriate for early concept analysis when this information may not be available. The method has a computational burden similar to a derivative-based approach and can be completed at the same time as such an approach with little overhead. As a result, it is recommended to augment current local derivative based sensitivity approaches with the HyVar analysis in order to obtain contribution measures along with the information typically provided by a derivative-based approach. The resulting contribution measures can then be used in the same manner as the contribution measures produced by a full variation-based analysis. If a full variation-based analysis is required, it should be performed once a relatively small set of feasible concepts have been developed and there is sufficient knowledge to establish parametric distributions. In this case, the measures from the full analysis can be directly superceded by the measures generated by the HyVar analysis.

The HyVar approach also allows direct addition of single parameter measures that provides a means of assessing the sensitivity contribution of a function in a functionally decomposed system behavioral model. The normalization of the measures to the functional size of system allows the measure to be used outside of the context of a particular system design and is conducive to design repository storage (see [45] for a discussion of a function-based design repository).

5. COMPUTATIONAL IMPLEMENTATION OF THE DESIGN TOOLS

The behavioral modeling and sensitivity analysis approaches presented in Sections 3 and 4 are conceptual in nature and require an actual implementation platform in order to be used in the context of a design problem. To this end, a software implementation of these methods was developed for use in completing examples and as an investigation of the requirements necessary for implementing the approaches. This implementation is detailed in this section and was used to generate the numerical results that appear throughout this work.

5.1. AUTOMATED MODEL ASSEMBLY AND SOLUTION IN PRACTICE

Component-based modeling of systems has been used extensively to solve industrial design problems. Probably the most widespread modeling tool that implements component-based modeling is the MATLAB/Simulink combination [8]. This modeling platform uses block diagram-based models to represent the constitutive elements of a system and has been used to model a variety of complex system. In addition to block-diagram based modeling tools, modeling languages such as Modelica have been developed that enable a more verbose description of the behavior of the components of a system. Modelica is an object-oriented modeling language derived from the early work of Elmqvist [35]. Tools such as Dymola [34] and OpenModelica [53] assemble and solve models represented in the Modelica language. Other simulation modeling frameworks [9] have been developed as well using similar philosophies.

The model development approach presented in Section 3 of this work is based on a functional decomposition rather than a component-based representation. In the functional decomposition approach, a function may be represented by a conceptual notion of how the function should be solved (a Type I model element), a model based on a family of solutions for the function (a Type II model element) or a very specific model developed based on a single particular solution (a Type III element). Additionally, a very specific model lexicon (the Functional Basis [10]) and development approach [29] is used in this modeling process. The use of such a lexicon promotes knowledge storage and reuse based on the concept of common functionality [45]. Existing modeling platforms, such as Simulink and Dymola with Modelica, were not developed around a functional decomposition. As a result, they are ill-suited for implementing such an approach.

Earlier work by the author implemented a functionality decomposed behavioral model in Simulink. In this work, the causal nature of the block diagram based models limited the potential reuse of model elements. Acausal model platforms, such as the Modelica language, overcome this limitation. However, the Modelica specification is currently being developed and refined and at the time this research originated there were limited Modelica compiler options. Currently, there is an open-source modeling platform being developed (OpenModelica) that implements the Modelica language. Since the Modelica specification and feature set has not stabilized and the OpenModelica suite (or another freely-available, multiplatform Modelica compiler) has not been sufficiently refined, its use in implementing the function-based modeling approach was not considered.

5.2. MODEL ASSEMBLY AND SOLUTION FRAMEWORK

The primary objectives in developing the model assembly and solution framework presented here were to create a useable implementation of the behavioral modeling and sensitivity work presented in Sections 3.1 and 3.2, to enable the application of these tools to the design of a complex system to demonstrate their effectiveness and to create a base of knowledge and software platform for future developments in the work.

Other objectives of the work included using a solution technique that allowed the example problems to be solved quickly and flexibly. To this end it was decided to use a general purpose, yet still relatively low-level programming language (C++) for model representation, assembly and solution.

The inputs to the modeling framework are the results of the application of the function-based behavioral modeling work shown in Section 3. The functional model along with the flow variable sets and model elements developed for each function represent the primary information required to be supplied by the modeler. To illustrate how the modeling framework was developed and provide a complete, yet tractable, example a simple three-function system (the work shown in Section 4) is presented along with the framework. The functional model for this system appears in Fig. 5.1 along with the flow routing information shown in Fig. 5.2. The flow variables used in this system are detailed in Table 5.1 along with the symbolic model elements in Table 5.2.

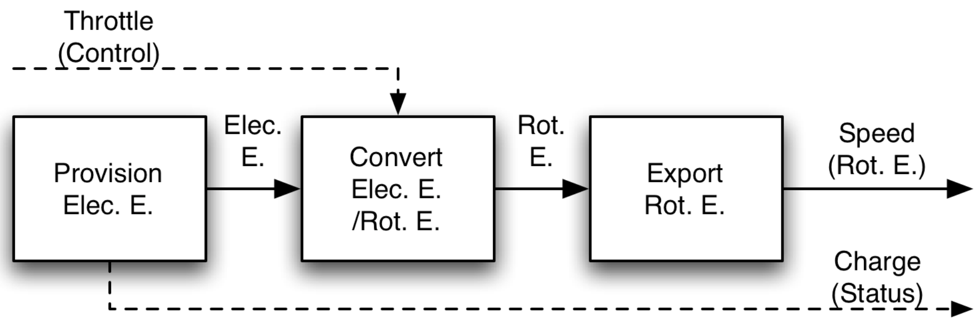


Fig. 5.1. Example System Functional Model

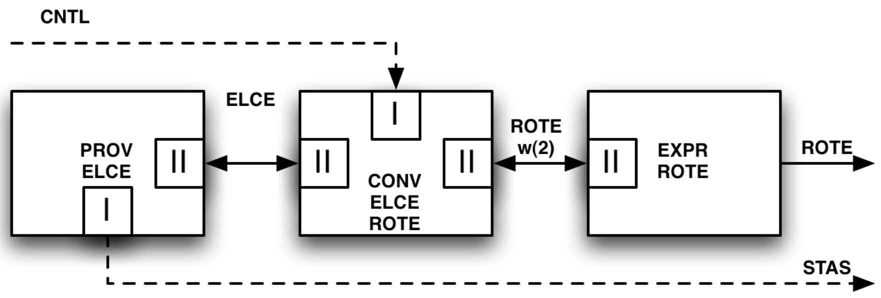


Fig. 5.2. Example System Flow Routing Model

Table 5.1. Three-function Model Flow Variables

Flow	Symbol	Type	Units	Description
Electrical Energy	$ELCE_{IIa,x,0}$	Voltage	V	Effort
	$ELCE_{IIa,x,1}$	Current	A	Flow
Rotational Energy	$ROTE_{IIa,x,0}$	Moment	N*m	Effort
	$ROTE_{IIa,x,1}$	Ang. Vel.	rad/s	Flow
Control Signal	$CNTL_{Ia,x,0}$	Analog	Varies	An analog control signal
Status Signal	$STAS_{Ia,x,0}$	Analog	Varies	A single analog status signal

Table 5.2. Three-function Model Elements

Function	Model Description	Relationship	Type
Provision Elec. E.	Linear battery model with internal resistance	$V_{B,nom} - V_B - I_B \cdot R_B = 0$	Imp. Lin. Alg. Eq.
		$\dot{C}_A = \frac{-I_B}{Q_{B,\max}}$	Exp. Lin. Diff. Eq.
Convert Elec. E. to Rot. E.	Constant torque to constant power transitional electrical machine model with linear proportional control	$\omega_T = \frac{P_{Max} \cdot V_S}{M_{Max} \cdot V_{Rated}}$	Exp. Alg. Eq.
		$if(\omega_M > \omega_T):$ $M_M - M_{Max} \cdot C_M = 0$ $else:$ $M_M - \frac{P_{Max} \cdot V_M \cdot C_M}{\omega_M \cdot V_{Rated}} = 0$	Hyb. Imp. Lin. Alg. Eq.
		$I_S - I_{Rated} \cdot C_M = 0$	Imp. Lin. Alg. Eq.
Export Rot. E.	Simple inertial and frictional load	$M_L - J_L \cdot \dot{\omega}_L - B_L \cdot \omega_L = 0$	Imp. Lin. Diff. Eq.

5.2.1. Functional Decompositions and the Object-Oriented Software Philosophy

To develop the model representation, assembly and solution framework, object-oriented programming practices were used. As introduced in Section 1, function-based design and object-oriented programming share the same philosophical underpinnings. Namely, the organization of the functional elements of a system into discrete elements with local information and interfaces for passing information to other functions in the system. However, this decomposition alone does not sufficiently describe function-based design (or object-oriented programming). Inheritance, polymorphism, encapsulation and abstraction are all important aspects of both approaches (and the keys to their similarity).

5.2.2. Inheritance and Instancing

Inheritance in object-oriented programming results from deriving new classes based on the structure of existing classes. Once defined, a class must be instantiated in order to be used. An instance of a class is a uniquely defined object that uses the representation provided by the class definition. In the realm of engineering design, the concept of inheritance relates to the hierarchy of definition used for functions and the solutions to these functions. For example, a *three-phase induction motor* inherits from the family of *AC motors*, which inherits from the family of *electrical machines*, which solve the function of *converting electrical energy to rotational energy* (and vice-versa). The class of *three-phase induction motor* may be instantiated by selecting or defining a full description of a specific motor in this context.

5.2.3. Polymorphism

Polymorphism allows an inherited class to override the behavior of its parent class. In the context of engineering design, a model may be made for a functional element of a system at a variety of levels. For example, a high-level model of a system that needs the functionality of an electrical machine may require a simplified model of this function's behavior during the early stages of design. However, later in the design process, a more detailed model for this functionality may be needed to provide the desired information from the system model. In this case, the behavior of the previous model must be replaced with the behavior of the newer model. Building polymorphism into a design framework allows this replacement of models to be performed seamlessly. A primary difference between the modeling approach presented in this work and prior efforts in integrating object-oriented programming philosophies into engineering design [35] is the formal inclusion and definition of functionality.

5.2.4. Encapsulation

Encapsulation involves the "hiding" of certain information within classes from other classes. In C++, the variables (members) of a class can be hidden from all other classes (private), exposed to inherited classes (protected) or can be exposed to all classes (public). In the engineering design domain, encapsulation is more strongly enforced in that the various components that comprise a system cannot "share" their internal properties at all. Energy, material and signal flows through the boundaries of the elements are the only method of communication a sub-system has with other sub-systems. However, in the context of a design problem, it can be useful to implement the programming perspective of encapsulation during modeling.

5.2.5. Abstraction

Abstraction is the primary difference between modular programming and true object-oriented programming. As stated by Bjarne Stroustrup, the creator of C++, a key aspect of object-oriented software development is to “make commonality explicit by using inheritance” [11]. Having a true object-oriented modeling framework allows an explicit representation of the abstraction of a design problem.

5.3. DEVELOPMENT OF A MODELING FRAMEWORK

To implement the function-based behavioral modeling tools presented in Sections 3.1 and 3.2, an object-oriented programming platform was used. Several programming languages support this paradigm and for this work C++ was selected. C++ is one of the most common languages used in modern application development [54] and supports the object-oriented design philosophy (OOP) while retaining and supporting most aspects of the C programming language. C++ compares favorably to many other languages that support OOP such as Java and scripting languages in terms of speed of execution at the expense of coding effort and complexity [55]. Since the author was familiar with C++ at the time of developing this framework, the speed benefit was chosen over the burden of becoming familiar with a different, but possibly simpler, language. Additionally, several numerical equation solvers are available for C (and hence C++) including the SUNDIALS package [41] that is used in this work.

5.3.1. Definition of Model Elements

The first task is creating an implementation of the behavioral modeling and sensitivity analysis framework was to create a class definition for a behavioral model element. Each function in the functional model requires such an element to express the mathematical relationships that represent behavior. Eight key functions were identified for a class that represents a behavioral model element including:

1. Initialization of the model element,
2. Loading of various parameterizations (to define concepts),
3. Allowing distributions for parameters and perturbations,

4. Interfacing with causal flow variables,
5. Interfacing with acausal flow variables,
6. Integration of flow and internal variables,
7. Local termination of the global system model,
8. Handling of local element members and methods.

These eight functions are implemented in a modeling framework that uses seven basic steps. This modeling process is outlined in the flow chart shown in Fig. 5.3 and each step is detailed next.

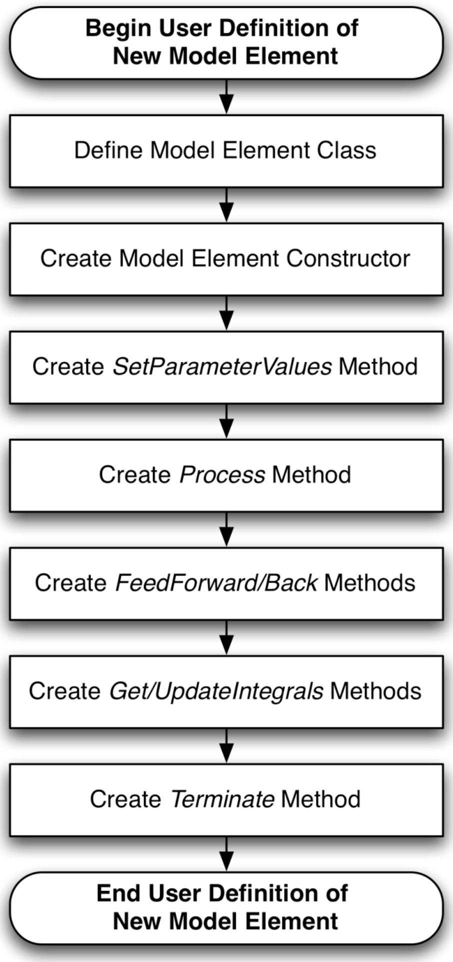
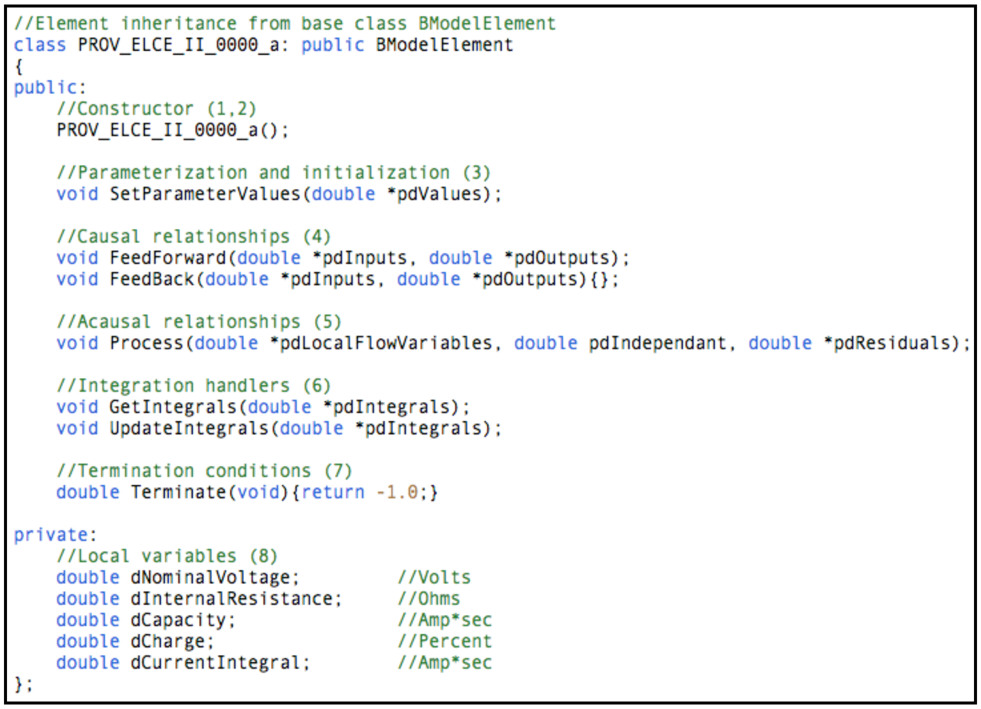

Begin User Definition of New Model Element
Define Model Element Class
Create Model Element Constructor
Create *SetParameterValues* Method
Create *Process* Method
Create *FeedForward/Back* Methods
Create *Get/UpdateIntegrals* Methods
Create *Terminate* Method
End User Definition of New Model Element

Fig. 5.3. User Definition of Model Element Flow Chart

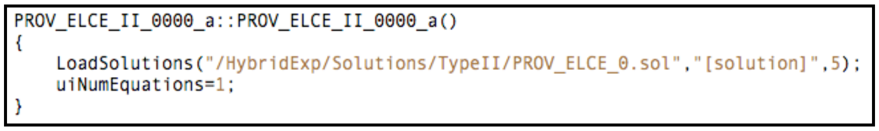
To this end, a virtual parent class (named **BModelElement**) was created that enabled each of these functionalities. A virtual class is incomplete and cannot be used in its defined form. A child class must be created that inherits from the virtual class and defines any missing functionality. An example of such a child class is the **PROV_ELCE_II_0000_a** class as defined in Fig. 5.4.



```
//Element inheritance from base class BModelElement
class PROV_ELCE_II_0000_a: public BModelElement
{
public:
    //Constructor (1,2)
    PROV_ELCE_II_0000_a();

    //Parameterization and initialization (3)
    void SetParameterValues(double *pdValues);

    //Causal relationships (4)
    void FeedForward(double *pdInputs, double *pdOutputs);
    void FeedBack(double *pdInputs, double *pdOutputs){};

    //Acausal relationships (5)
    void Process(double *pdLocalFlowVariables, double pdIndependant, double *pdResiduals);

    //Integration handlers (6)
    void GetIntegrals(double *pdIntegrals);
    void UpdateIntegrals(double *pdIntegrals);

    //Termination conditions (7)
    double Terminate(void){return -1.0;}

private:
    //Local variables (8)
    double dNominalVoltage;         //Volts
    double dInternalResistance;     //Ohms
    double dCapacity;               //Amp*sec
    double dCharge;                 //Percent
    double dCurrentIntegral;        //Amp*sec
};
```

Fig. 5.4. PROV_ELCE_II_0000_a Class Definition

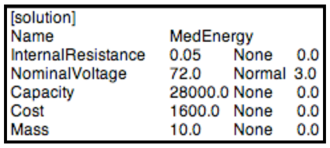
As shown in Fig. 5.4. The **PROV_ELCE_II_0000_a** class inherits from **BModelElement** and defines the eight characteristics of this behavioral model element. The naming convention used for this class includes the FourCC codes presented earlier in the work (PROV for provision and ELCE for electrical energy) along with a model element type definition (II), a number indicating the specific model (in this case 0000) and a variant (a). All model elements used in the remainder of this work conform to this convention. The definition of each of the eight characteristics is noted in the figure. The constructor (a method that is called each time the class is instantiated) initializes the model element and loads a set of parameterized concepts for the element as shown in the code in Fig. 5.5.



```
PROV_ELCE_II_0000_a::PROV_ELCE_II_0000_a()
{
    LoadSolutions("/HybridExp/Solutions/TypeII/PROV_ELCE_0.sol","[solution]",5);
    uiNumEquations=1;
}
```

Fig. 5.5. PROV_ELCE_II_0000_a Constructor

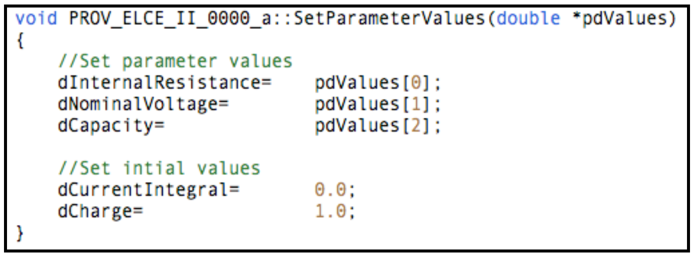
The code shown in Fig. 5.5 opens a solution file (a text file like that shown in Fig. 5.6) and loads five parameter values along with their distributions for each concept (noted with a [solution] tag in the text file). Currently, the modeling framework accepts uniform and normal distributions with a single parameter each. Due to the object-oriented design of the model assembly code, it a simple matter to extend the method to handle additional distributions that require more than one parameter if necessary.



```
[solution]
Name                MedEnergy
InternalResistance  0.05    None    0.0
NominalVoltage      72.0    Normal  3.0
Capacity            28000.0 None    0.0
Cost                1600.0  None    0.0
Mass                10.0    None    0.0
```

Fig. 5.6. PROV_ELCE_II_0000_a Parameter Sample

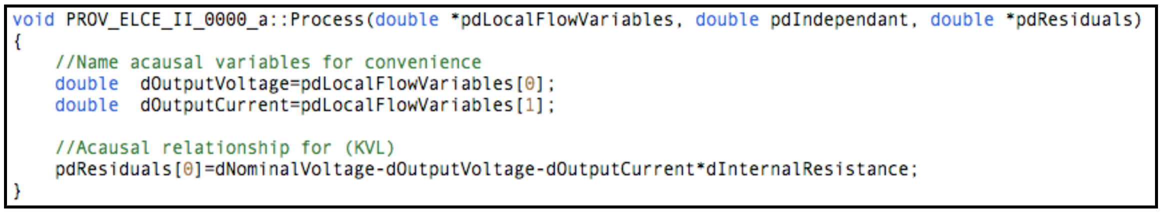
The third aspect of functionality, allowing distributions and perturbations of parameter values for a sensitivity analysis, is implemented for an element as shown in the code in Fig. 5.7. In this code, the local parameter values can be set based on a call to the **SetParameterValues** method with an input of an array containing variable values. Additionally, initial values are then set (such as the charge and charge integral variables for the element shown).



```
void PROV_ELCE_II_0000_a::SetParameterValues(double *pdValues)
{
    //Set parameter values
    dInternalResistance=    pdValues[0];
    dNominalVoltage=        pdValues[1];
    dCapacity=              pdValues[2];

    //Set intial values
    dCurrentIntegral=       0.0;
    dCharge=                1.0;
}
```

Fig. 5.7. PROV_ELCE_II_0000_a SetParameterValues Method

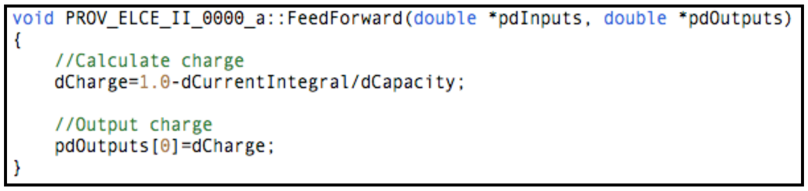
The acausal relationships for each model element are defined using a set of residual functions. Acausal relationships take the form of implicit equations (f(x)=0) rather than explicit equation (y=f(x)). For this reason, they are generally referred to as constraints or relationships rather than equations. For the local acausal constraints to be satisfied, the values of these residual functions must be zero. The code shown in Fig. 5.8 implements these functions for the **PROV_ELCE_II_0000_a** element. As shown in this figure, two flow variables are sent to the **Process** method in the **pdLocalFlowVariables** array along with the value of the independent variable and an array containing the values of the residuals. The first two lines of code in the **Process** method create named variables for the flow variables for convenience. The last line defined is the single residual function for the element. The function is based on the mathematical relationship defined for this element as shown in Fig. 3.13.



```
void PROV_ELCE_II_0000_a::Process(double *pdLocalFlowVariables, double pdIndependant, double *pdResiduals)
{
    //Name acausal variables for convenience
    double  dOutputVoltage=pdLocalFlowVariables[0];
    double  dOutputCurrent=pdLocalFlowVariables[1];

    //Acausal relationship for (KVL)
    pdResiduals[0]=dNominalVoltage-dOutputVoltage-dOutputCurrent*dInternalResistance;
}
```

Fig. 5.8. PROV_ELCE_II_0000_a Process Method

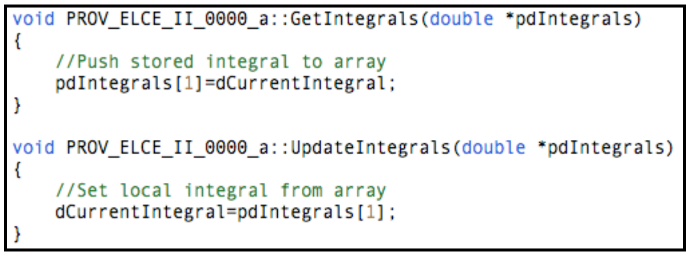
The **FeedForward** method shown in Fig. 5.9. implements the causal modeling functionality of the element. For this method, a set of causal inputs are provided (from the causal outputs of connected elements upstream) and a set of causal outputs are computed (and sent to the inputs of connected causal elements downstream). In the case of the **PROV_ELCE_II_0000_a** element, the charge in the element is computed and populated into the output vector.



```
void PROV_ELCE_II_0000_a::FeedForward(double *pdInputs, double *pdOutputs)
{
    //Calculate charge
    dCharge=1.0-dCurrentIntegral/dCapacity;

    //Output charge
    pdOutputs[0]=dCharge;
}
```

Fig. 5.9. PROV_ELCE_II_0000_a FeedForward Method

An interface to integration is provided by two methods: **GetIntegrals** and **UpdateIntegrals** respectively. The first method takes the locally stored value for an integral and populates it into an array. The second method sets the local integral value based on an incoming array. The actual integration is handled by the **BModel** class (which is detailed later in this work). The integration methods used are shown in Fig. 5.10.



```
void PROV_ELCE_II_0000_a::GetIntegrals(double *pdIntegrals)
{
    //Push stored integral to array
    pdIntegrals[1]=dCurrentIntegral;
}

void PROV_ELCE_II_0000_a::UpdateIntegrals(double *pdIntegrals)
{
    //Set local integral from array
    dCurrentIntegral=pdIntegrals[1];
}
```

Fig. 5.10. PROV_ELCE_II_0000_a Integration Methods

For each element, a set of local variables can be established in the class definition along with a custom local termination function that allows the global solution to be terminated and interpolated based on local termination criteria. For the **PROV_ELCE_II_0000_a** element, the local variables are shown in Fig. 5.4. and the **Terminate** method is set to simply return the value -1.0. Return values of the **Terminate** method less than one indicate that the solution should not be terminated based on local conditions while values between 0.0 and 1.0 indicate that the solution should terminate and linearly interpolate the performance and output variables between the last integration step (0.0) and the current integration step (1.0).

The code shown in Figs. 5.4-5.10 represents all the code necessary to completely represent a single model element. The actual assembly and solution of model elements is handled by the **BModelElement** parent class along with a **BModel** class that contains instances of the individual model elements included in the model. For each system model, a new class should be created that inherits from the class **BModel**. The **BModel** class handles all of the automated assembly and solution code and, like the **BModelElement** class, is virtual and requires the definition of a few methods in order to be used. A flow chart outlining the actions that must be performed by a modeler implementing a new child class of **BModel** appears in Fig. 5.11.

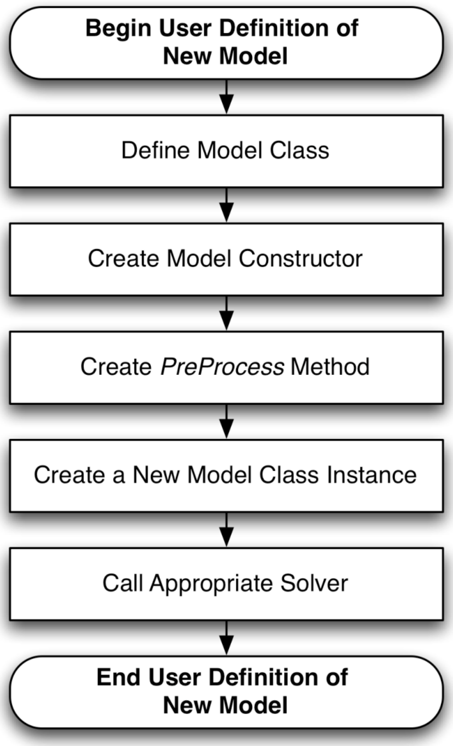
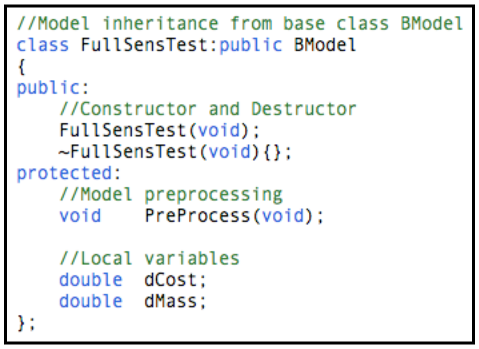

Begin User Definition of New Model
Define Model Class
Create Model Constructor
Create *PreProcess* Method
Create a New Model Class Instance
Call Appropriate Solver
End User Definition of New Model

Fig. 5.11. User Definition of New Model Flow Chart

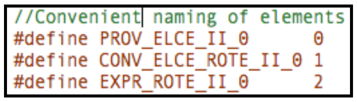
A sample of a useable class inherited from **BModel** is shown in Fig. 5.12. In this code, a new model class called *FullSensTest* is defined from the parent class of **BModel**. This class was used in the sensitivity analysis performed in Section 4.3. For this new class, a constructor is defined along with an empty destructor (no operations are performed upon de-allocation of instances of this class). Additionally, a **PreProcess** method is defined that handles any custom computations required before an attempted solution of the model. As with the model elements, local variables can be defined for the model as a whole (in this case cost and mass variables for the entire system).



```
//Model inheritance from base class BModel
class FullSensTest:public BModel
{
public:
    //Constructor and Destructor
    FullSensTest(void);
    ~FullSensTest(void){};
protected:
    //Model preprocessing
    void    PreProcess(void);

    //Local variables
    double  dCost;
    double  dMass;
};
```

Fig. 5.12. FullSensTest Class Definition

In the developed modeling framework, each model element (corresponding to a function in the functional model) is added to a vector. When adding a new element, it is useful to create a descriptive name for the index of the newly added element. To this end, **#define** statements can be used to implement such naming as shown in Fig. 5.13. In this code, the elements are named using the FourCC convention along with the model element type and a unique number for each identical function name (since each function appears only once in this example, each is numbered 0).



```
//Convenient naming of elements
#define PROV_ELCE_II_0      0
#define CONV_ELCE_ROTE_II_0 1
#define EXPR_ROTE_II_0      2
```

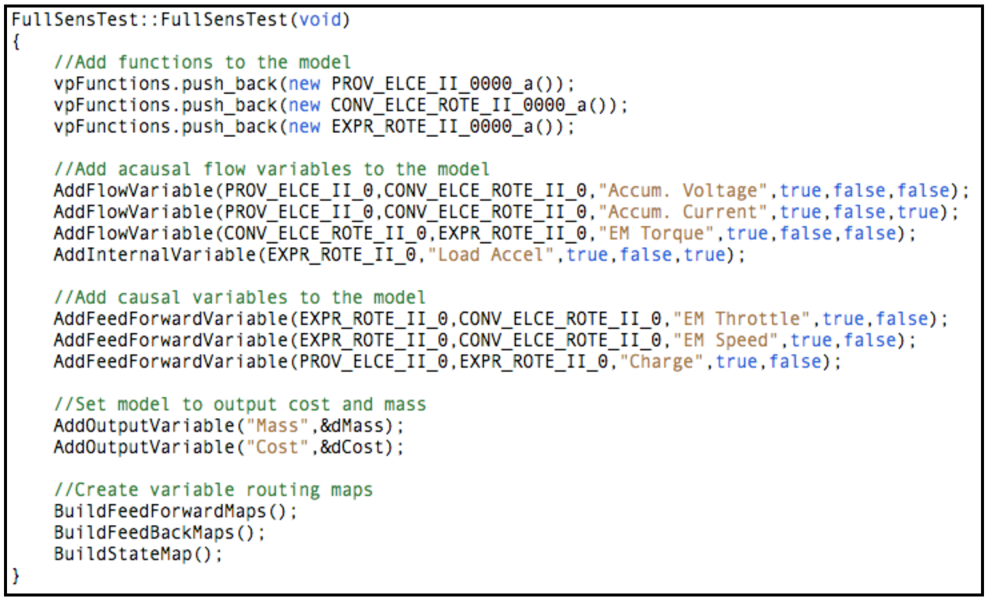
Fig. 5.13. FullSensTest Naming Conventions

Now that a class definition has been made along with naming conventions, the methods for the derived *FullSensTest* class can now be defined. First, a constructor is needed to add model elements and establish the flow routing in the system. The code for this method appears in Fig. 5.14. The first three lines of code create new instances of each element and push each of the elements to a vector. The following lines of code add the causal and acausal flow variables and any necessary internal variables to the system.

The first flow added is the voltage from the *PROV_ELCE_II_0* element to the *CONV_ELCE_ROTE_II_0* element. As seen in the code that adds this flow, a descriptive name (Accum. Voltage) and three Boolean flags used. The first flag (true) indicates that this variable should be logged to a data file during the solution. The second flag (false) indicates that this variable is not a performance (output) variable. Finally, the third flag (false) indicates that this variable does not need to be integrated during the solution. The remaining flow variables in the system are added in the following two lines of code along with an internal variable that represents the acceleration and its integral (the third flag is true, meaning this variable must be integrated).

The causal variables are then added to the model. In the case of the *FullSensTest* model, three causal variables are used including the throttle control signal, a speed status signal and a charge status signal. The order of elements shown in the **AddFeedForwardVariable** method represents a strict order from origin element to destination element. Additionally, a descriptive name and logging and performance flags are included. In the developed framework, causal variables were not allowed to be integrated (if integration of a causal is necessary it can be implemented with an internal flow variable such as the “Load Accel” variable).

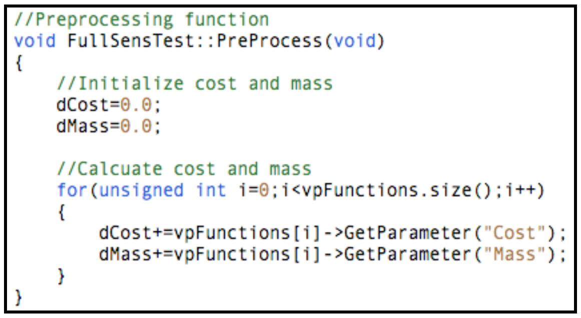
The final lines of code shown in Fig. 5.14 add two output variables to the model: mass and cost. These variables are computed and then output to a performance file at the completion of a solution. Finally, three methods are called to build the maps necessary to assemble and solve the model. These functions are defined in the **BModel** class and are described later in this work.



```
FullSensTest::FullSensTest(void)
{
    //Add functions to the model
    vpFunctions.push_back(new PROV_ELCE_II_0000_a());
    vpFunctions.push_back(new CONV_ELCE_ROTE_II_0000_a());
    vpFunctions.push_back(new EXPR_ROTE_II_0000_a());

    //Add acausal flow variables to the model
    AddFlowVariable(PROV_ELCE_II_0,CONV_ELCE_ROTE_II_0,"Accum. Voltage",true,false,false);
    AddFlowVariable(PROV_ELCE_II_0,CONV_ELCE_ROTE_II_0,"Accum. Current",true,false,true);
    AddFlowVariable(CONV_ELCE_ROTE_II_0,EXPR_ROTE_II_0,"EM Torque",true,false,false);
    AddInternalVariable(EXPR_ROTE_II_0,"Load Accel",true,false,true);

    //Add causal variables to the model
    AddFeedForwardVariable(EXPR_ROTE_II_0,CONV_ELCE_ROTE_II_0,"EM Throttle",true,false);
    AddFeedForwardVariable(EXPR_ROTE_II_0,CONV_ELCE_ROTE_II_0,"EM Speed",true,false);
    AddFeedForwardVariable(PROV_ELCE_II_0,EXPR_ROTE_II_0,"Charge",true,false);

    //Set model to output cost and mass
    AddOutputVariable("Mass",&dMass);
    AddOutputVariable("Cost",&dCost);

    //Create variable routing maps
    BuildFeedForwardMaps();
    BuildFeedBackMaps();
    BuildStateMap();
}
```

Fig. 5.14. FullSensTest Constructor

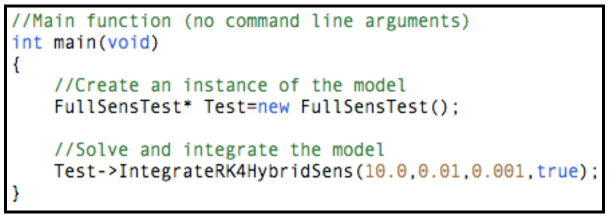
For each model, a **PreProcess** method is defined to allow any computations necessary before a solution is attempted. In the case of the *FullSensTest* model, the cost and mass variables are calculated by getting the values of cost and mass for each element and summing them. Since **PreProcess** is a virtual method and is user-defined, any code may be included in this method. This represents a primary difference between a fixed simulation modeling environment or modeling language and the framework presented here. Since the model representation code is C++, any C or C++ code can be incorporated into the virtual methods built into the framework. This enables access to any number of external sources of data or other programs (code could be added to access a spreadsheet, or the results of simulation program, etc.). Additionally, the use of the object-oriented design philosophy enables developers to create derived classes from **BModel** that include such external accessibility tools in their definition, thus alleviating the need of the modeler to custom create such tools. Essentially, using C++ and object-oriented design allows a modeler freedom to do as much (custom coded access tools) or as little (using someone else's custom-coded access tools) as desired (or necessary) while having full access to all of the resources available with a general purpose programming language. The **PreProcess** method for the *FullSensTest* class appears in Fig. 5.15.



```
//Preprocessing function
void FullSensTest::PreProcess(void)
{
    //Initialize cost and mass
    dCost=0.0;
    dMass=0.0;

    //Calcuate cost and mass
    for(unsigned int i=0;i<vpFunctions.size();i++)
    {
        dCost+=vpFunctions[i]->GetParameter("Cost");
        dMass+=vpFunctions[i]->GetParameter("Mass");
    }
}
```

Fig. 5.15. FullSensTest PreProcess Method

Once a **BModel** derived class has been defined and the various virtual methods developed, the resulting model can then be solved. A number of integration and sensitivity analysis methods were built into the **BModel** base class. For the *FullSensTest* model, a fixed-step Runge-Kutta 4 integration [56] is used along with the hybrid sensitivity analysis presented in Section 4. The complete code required to create a new instance of the *FullSensTest* model and solve it is shown in Fig. 5.16. As seen in this figure, only two lines of code are needed to create a new instance of the model and solve it. This represents another strength of the object-oriented design philosophy. Once a class has been fully defined, the actual implementation of the class can be simply performed. In the code shown in Fig. 5.16, an instance (*Test*) of the *FullSensTest* class is created. Upon creation, all of the code listed in the constructor for this class (Fig. 5.4) is called and the behavioral model for the system is built. Next, a call to the **IntegrateRK4HybridSens** method is made (this method is defined in the **BModel** class) along with integration parameters. The first parameter is the length of integration (10.0 seconds). Next is the integration step (0.01 seconds) followed by the perturbation to be used in the sensitivity analysis (0.1%). Finally, there is a flag to indicate whether or not each run of the sensitivity analysis should be logged separately (in this case *true* for yes).



```
//Main function (no command line arguments)
int main(void)
{
    //Create an instance of the model
    FullSensTest* Test=new FullSensTest();

    //Solve and integrate the model
    Test->IntegrateRK4HybridSens(10.0,0.01,0.001,true);
}
```

Fig. 5.16. Example Main Function

5.3.2. Representation, Assembly and Solution

The code shown in the previous sections represents the user-supplied code necessary to define and implement a behavioral model in the developed framework. The actual pre-defined code that allows this definition and actually performs the assembly, solution and integration of the model is presented next. This code is defined in the **BModel** and **BModelElement** classes and is transparently included when a class is derived from either of these parent classes. A flow chart outlining the model assembly process appears in Fig. 5.17.

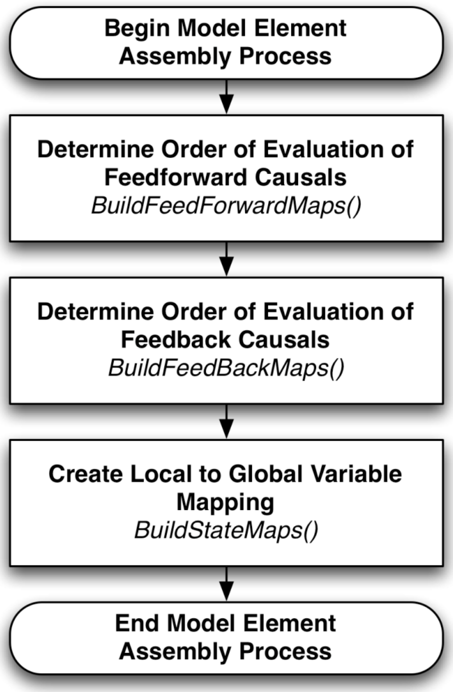

Begin Model Element Assembly Process
Determine Order of Evaluation of Feedforward Causals
BuildFeedForwardMaps()
Determine Order of Evaluation of Feedback Causals
BuildFeedBackMaps()
Create Local to Global Variable Mapping
BuildStateMaps()
End Model Element Assembly Process

Fig. 5.17. Model Element Assembly Flow Chart

The determination of causal model evaluation order and variable routing is performed in five basic steps. These steps are outlined in Fig. 5.18 and are detailed next.

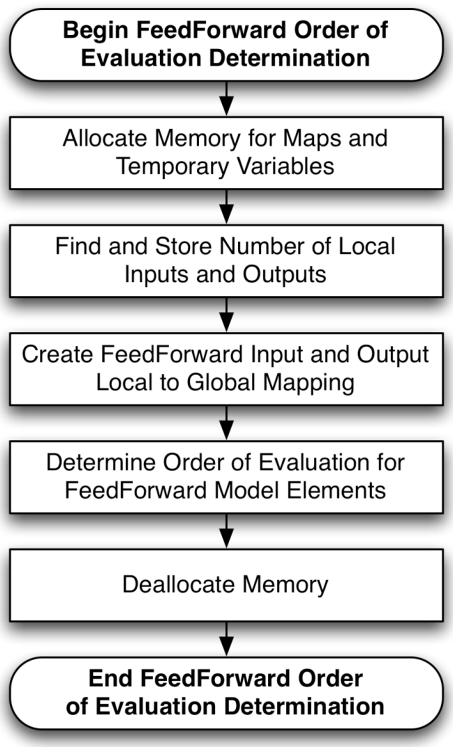
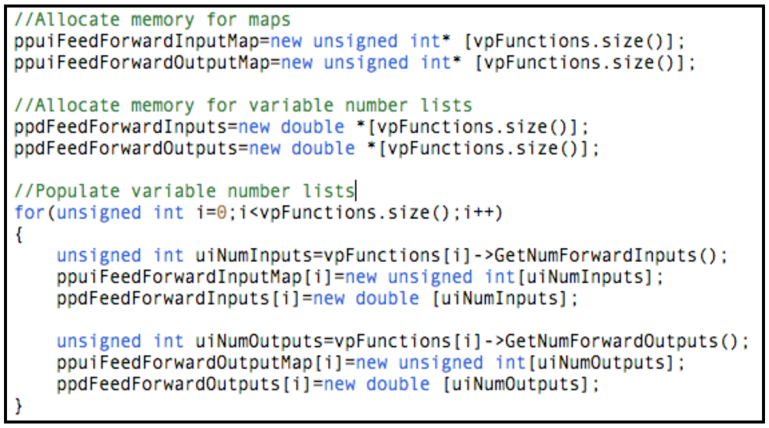

Begin FeedForward Order of Evaluation Determination
Allocate Memory for Maps and Temporary Variables
Find and Store Number of Local Inputs and Outputs
Create FeedForward Input and Output Local to Global Mapping
Determine Order of Evaluation for FeedForward Model Elements
Deallocate Memory
End FeedForward Order of Evaluation Determination

Fig. 5.18. FeedForward Order of Evaluation Determination Flow Chart

The first method presented is **BuildFeedForwardMaps**. This method sorts the feedforward causal variables for processing during solution and integration and is necessary to sequence the casuals so that they can be processed in order (casual variables and relationships may be added to the system model out of the sequence in which they must be evaluated). For example, a motor controller needs to know the charge of the batteries connected to it before it can calculate the throttle to be provided to the motor. This set of operations can be defined using strictly acausal variables and relationships (such as used in the Modelica language) or strictly causal (such as in a block diagram model like those used in Simulink) but a mixed approach, allowing both causal and acausal modeling is useful from the standpoint of the modeler. For example, logical operations can be implemented in causal modeling that would be prohibitively difficult to solve in acausal modeling (requiring a highly sophisticated hybrid DAE solution). However, some relationships are difficult to express in a causal manner during the modeling of elements [35, 38] and necessitate an acausal approach.

From an assembly standpoint, causal modeling is actually more difficult than acausal modeling as the sequencing of equations must be performed (order is not important in acausal modeling). However, the solution of the resulting model is much easier for causal modeling as the elements can be evaluated in isolation and outputs can be passed to the next element's inputs. Acausal model solution is extremely difficult and generally requires symbolic manipulation and state identification (usually performed with Pantelides' [57] algorithm) along with a robust DAE solver (this is the approach used in Modelica compilers). However, if most of the causal relationships are extracted and explicitly defined by the modeler leaving only a minimum of acausal relationships for each element and higher-index constraints are not used, an acausal solution can be attempted using an non-linear solution method along with a standard integration approach (such as a Runge-Kutta 4 method). Essentially, such a modeling approach reduces to solving a set of non-linear equations at each integration step and then integrating the system as a set of ODEs. This approach is sufficient for modeling the systems considered in this work and many systems encountered in the engineering modeling domain (most engineers with undergraduate educations are not even exposed to the formal concept of a DAE, let alone hybrid DAEs with high-index constraints).

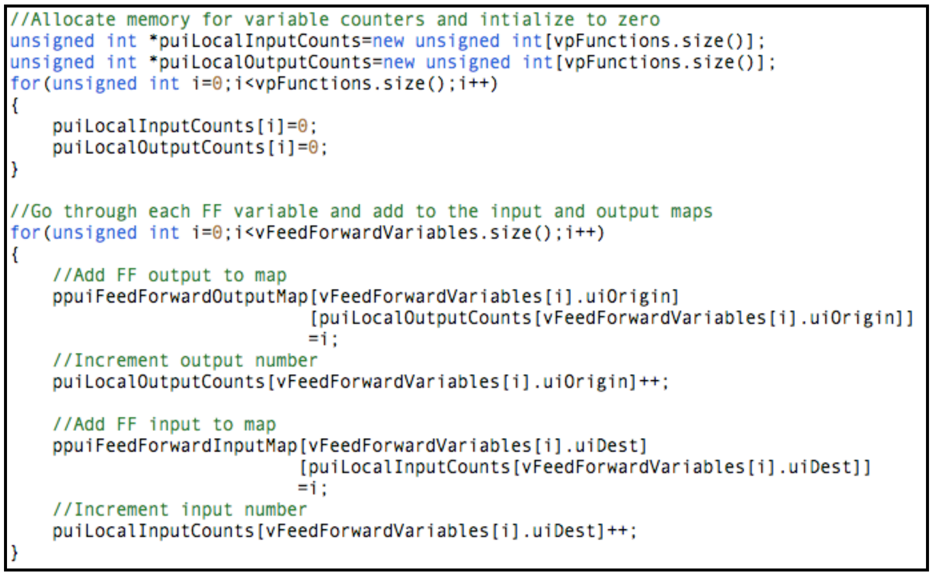
To implement causal modeling in the developed modeling framework, a set of input/output maps are used to determine the order in which model elements are evaluated and how causal variables should be passed from element to element. The code for this implementation is shown in Figs. 5.19 through 5.22 and begins with the definition of input and output maps as shown in Fig. 5.19. Each map is a two-dimensional array. The first index of the array is the element, the second index is the local causal input (for the input map) or the output (for the output map). Each element of the array (keyed by element and local variable) is the global identifier for that causal variable. This global identity is established in the order the casuals are added to the model in the constructor and starts with an unsigned integer value of 0.



```
//Allocate memory for maps
ppuiFeedForwardInputMap=new unsigned int* [vpFunctions.size()];
ppuiFeedForwardOutputMap=new unsigned int* [vpFunctions.size()];

//Allocate memory for variable number lists
ppdFeedForwardInputs=new double *[vpFunctions.size()];
ppdFeedForwardOutputs=new double *[vpFunctions.size()];

//Populate variable number lists
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    unsigned int uiNumInputs=vpFunctions[i]->GetNumForwardInputs();
    ppuiFeedForwardInputMap[i]=new unsigned int[uiNumInputs];
    ppdFeedForwardInputs[i]=new double [uiNumInputs];

    unsigned int uiNumOutputs=vpFunctions[i]->GetNumForwardOutputs();
    ppuiFeedForwardOutputMap[i]=new unsigned int[uiNumOutputs];
    ppdFeedForwardOutputs[i]=new double [uiNumOutputs];
}
```

Fig. 5.19. BuildFeedForwardMaps Allocation Code

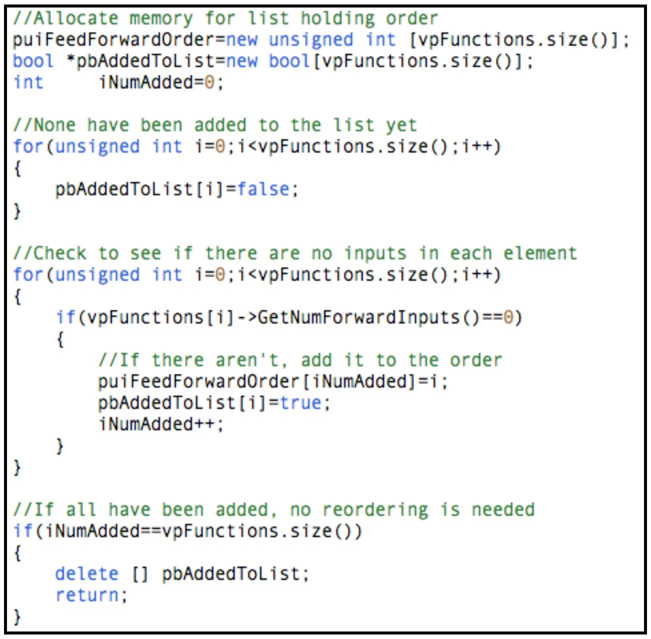
The code shown in Fig. 5.20 populates the input and output maps by going through each feedforward variable and adding its origin and destination elements. An example of the population of these maps is shown in Section 5.4.



```
//Allocate memory for variable counters and intialize to zero
unsigned int *puiLocalInputCounts=new unsigned int[vpFunctions.size()];
unsigned int *puiLocalOutputCounts=new unsigned int[vpFunctions.size()];
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    puiLocalInputCounts[i]=0;
    puiLocalOutputCounts[i]=0;
}

//Go through each FF variable and add to the input and output maps
for(unsigned int i=0;i<vFeedForwardVariables.size();i++)
{
    //Add FF output to map
    ppuiFeedForwardOutputMap[vFeedForwardVariables[i].uiOrigin]
                            [puiLocalOutputCounts[vFeedForwardVariables[i].uiOrigin]]
                            =i;
    //Increment output number
    puiLocalOutputCounts[vFeedForwardVariables[i].uiOrigin]++;

    //Add FF input to map
    ppuiFeedForwardInputMap[vFeedForwardVariables[i].uiDest]
                           [puiLocalInputCounts[vFeedForwardVariables[i].uiDest]]
                           =i;
    //Increment input number
    puiLocalInputCounts[vFeedForwardVariables[i].uiDest]++;
}
```

Fig. 5.20. BuildFeedForwardMaps Generation Code (1)

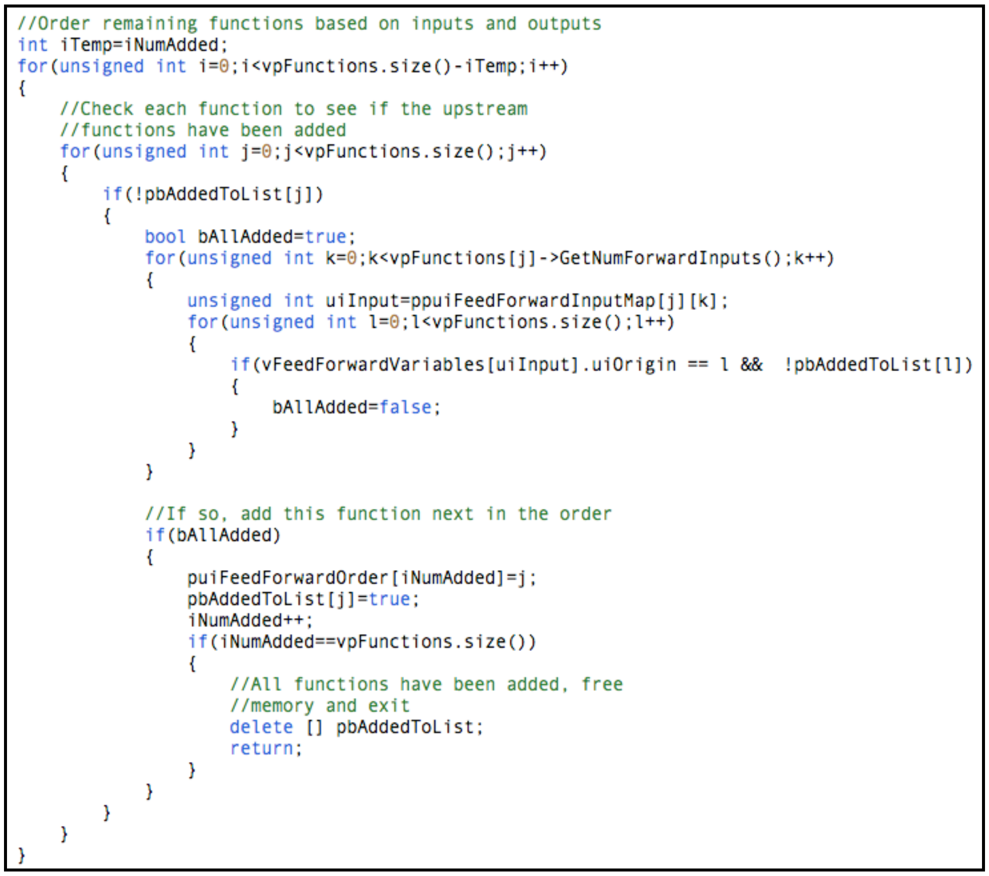
Once the input and output maps are established, they can be used to determine the order in which the model elements need to be evaluated. To perform this ordering, an order vector (**puiFeedForwardOrder**) must be created and initialized. This is performed using the code shown in Fig. 5.21. In this code, each element is checked for input variables. If there are no input variables, it is added to the order vector (if there are no inputs the element is not dependant on the evaluation of prior elements). If all elements have been added at the completion of this step, no sorting of elements is required and the method returns.



```
//Allocate memory for list holding order
puiFeedForwardOrder=new unsigned int [vpFunctions.size()];
bool *pbAddedToList=new bool[vpFunctions.size()];
int     iNumAdded=0;

//None have been added to the list yet
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    pbAddedToList[i]=false;
}

//Check to see if there are no inputs in each element
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    if(vpFunctions[i]->GetNumForwardInputs()==0)
    {
        //If there aren't, add it to the order
        puiFeedForwardOrder[iNumAdded]=i;
        pbAddedToList[i]=true;
        iNumAdded++;
    }
}

//If all have been added, no reordering is needed
if(iNumAdded==vpFunctions.size())
{
    delete [] pbAddedToList;
    return;
}
```

Fig. 5.21. BuildFeedForwardMaps Generation Code (2)

If some of the elements in the model require inputs from other elements, the order in which the elements should be evaluated must be determined. This determination is performed in the code shown in Fig. 5.22. For each element not already added to the evaluation order, a check is performed versus all other elements to determine if it can be added to the order. If all elements providing inputs to the element being considered are already in the order, the element can be added. If not, then another element is checked using the same criteria. This reordering is similar to a traditional sort algorithm and completes once all elements have been added. If all elements cannot be added, an order could not be established and the system is unsolvable.



```
//Order remaining functions based on inputs and outputs
int iTemp=iNumAdded;
for(unsigned int i=0;i<vpFunctions.size()-iTemp;i++)
{
    //Check each function to see if the upstream
    //functions have been added
    for(unsigned int j=0;j<vpFunctions.size();j++)
    {
        if(!pbAddedToList[j])
        {
            bool bAllAdded=true;
            for(unsigned int k=0;k<vpFunctions[j]->GetNumForwardInputs();k++)
            {
                unsigned int uiInput=ppuiFeedForwardInputMap[j][k];
                for(unsigned int l=0;l<vpFunctions.size();l++)
                {
                    if(vFeedForwardVariables[uiInput].uiOrigin == l &&  !pbAddedToList[l])
                    {
                        bAllAdded=false;
                    }
                }
            }

            //If so, add this function next in the order
            if(bAllAdded)
            {
                puiFeedForwardOrder[iNumAdded]=j;
                pbAddedToList[j]=true;
                iNumAdded++;
                if(iNumAdded==vpFunctions.size())
                {
                    //All functions have been added, free
                    //memory and exit
                    delete [] pbAddedToList;
                    return;
                }
            }
        }
    }
}
```

Fig. 5.22. BuildFeedForwardMaps Generation Code (3)

The same method is applied to feedback variables (feedforward variables are computed first, then feedback variable are computed based on the result of the feedforward analysis). This enables two paths of causal execution before the acausal elements are solved. Once ordered, the actual calculation of the causal variables is performed in the elements themselves and the local inputs and outputs are routed to the global causal variable list based on the input and output maps.

Four basic steps are used to assemble the local acausal relationships and relate local flow variables to global flow variables. These steps are outlined in Fig. 5.23 and are detailed next.

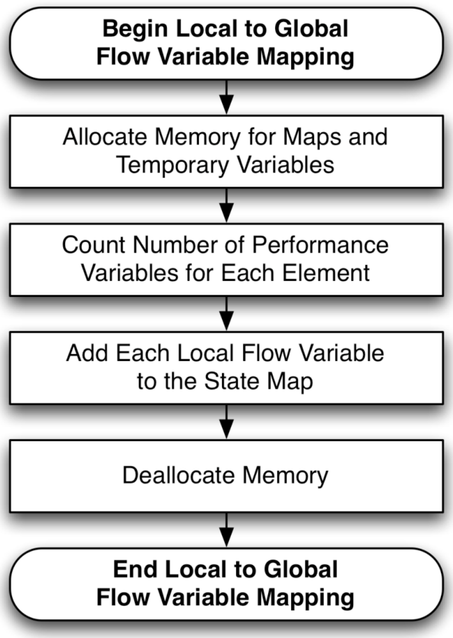
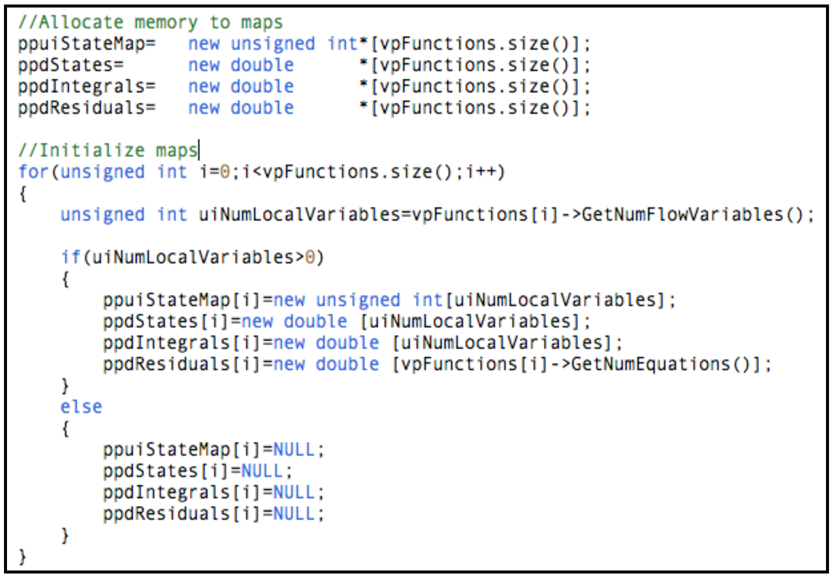

Begin Local to Global Flow Variable Mapping
Allocate Memory for Maps and Temporary Variables
Count Number of Performance Variables for Each Element
Add Each Local Flow Variable to the State Map
Deallocate Memory
End Local to Global Flow Variable Mapping

Fig. 5.23. Local to Global Flow Variable Mapping Flow Chart

For the acausal variables, order of evaluation is not important. The primary difficulty in assembling the elements operating on these variables is the routing of global to local information and vice versa. To accomplish this task, a set of maps, similar to the input and output maps used in the causal analysis, are utilized. The method that performs this mapping is **BuildStateMap** and must be called in the constructor of a class derived from **BModel**. The code for this method is detailed in Figs. 5.24 through 5.26. The primary map used is the **ppuiStateMap** shown in Fig. 5.24. This map is a two-dimensional array that contains references to the global identifier of each flow and internal variable added to the model (starting at an integer value of 0). The first index of this array is the element in the system and the second index is the local variable identifier. Thus, for a given element identifier and a local variable identifier, a global variable identifier is stored. This allows the quick mapping of local variable values to global variable values.

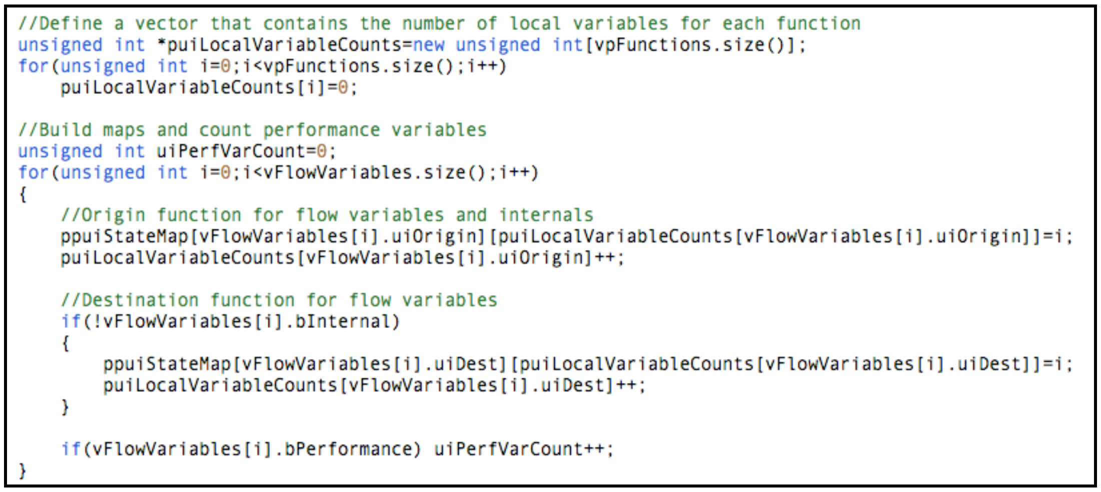
These values are routed using the **ppdStates** and **ppdIntegrals** arrays. These arrays hold the actual numerical values of flow variables as they are passed from the global list of variables to local elements during the solution of the model. The **ppdResiduals** array holds the residual evaluations for each element. The definition and initialization of these arrays is shown in Fig. 5.24.



```
//Allocate memory to maps
ppuiStateMap=   new unsigned int*[vpFunctions.size()];
ppdStates=      new double      *[vpFunctions.size()];
ppdIntegrals=   new double      *[vpFunctions.size()];
ppdResiduals=   new double      *[vpFunctions.size()];

//Initialize maps
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    unsigned int uiNumLocalVariables=vpFunctions[i]->GetNumFlowVariables();

    if(uiNumLocalVariables>0)
    {
        ppuiStateMap[i]=new unsigned int[uiNumLocalVariables];
        ppdStates[i]=new double [uiNumLocalVariables];
        ppdIntegrals[i]=new double [uiNumLocalVariables];
        ppdResiduals[i]=new double [vpFunctions[i]->GetNumEquations()];
    }
    else
    {
        ppuiStateMap[i]=NULL;
        ppdStates[i]=NULL;
        ppdIntegrals[i]=NULL;
        ppdResiduals[i]=NULL;
    }
}
```

Fig. 5.24. BuildStateMap Initialization Code

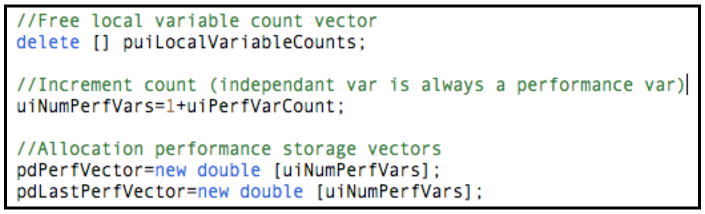
To populate the state map, each flow variable (including the internal variables) is inspected for its origin function and destination. Each time a flow variable originates or terminates at a particular function, the local variable identifier for that function is incremented and the global identifier of that variable is recorded in the variable map. Additionally, this code records each performance variable for use later on. The variable map generation code appears in Fig. 5.25.



```
//Define a vector that contains the number of local variables for each function
unsigned int *puiLocalVariableCounts=new unsigned int[vpFunctions.size()];
for(unsigned int i=0;i<vpFunctions.size();i++)
    puiLocalVariableCounts[i]=0;

//Build maps and count performance variables
unsigned int uiPerfVarCount=0;
for(unsigned int i=0;i<vFlowVariables.size();i++)
{
    //Origin function for flow variables and internals
    ppuiStateMap[vFlowVariables[i].uiOrigin][puiLocalVariableCounts[vFlowVariables[i].uiOrigin]]=i;
    puiLocalVariableCounts[vFlowVariables[i].uiOrigin]++;

    //Destination function for flow variables
    if(!vFlowVariables[i].bInternal)
    {
        ppuiStateMap[vFlowVariables[i].uiDest][puiLocalVariableCounts[vFlowVariables[i].uiDest]]=i;
        puiLocalVariableCounts[vFlowVariables[i].uiDest]++;
    }

    if(vFlowVariables[i].bPerformance) uiPerfVarCount++;
}
```

Fig. 5.25. BuildStateMap Generation Code

The remaining code in the **BuildStateMap** method appears in Fig. 5.26. This code cleans up memory and allocates vectors for storing the performance evaluations of the current and last integration steps. These vectors are used to interpolate the performances of models that terminate between integration steps.



```
//Free local variable count vector
delete [] puiLocalVariableCounts;

//Increment count (independant var is always a performance var)
uiNumPerfVars=1+uiPerfVarCount;

//Allocation performance storage vectors
pdPerfVector=new double [uiNumPerfVars];
pdLastPerfVector=new double [uiNumPerfVars];
```

Fig. 5.26. BuildStateMap Clean Up Code

Once the casual order and acausal variable routing maps are established, a solution can be attempted for the global model. The **IntegrateRK4HybridSens** method called in Fig. 5.16 for the *FullSensTest* derived model represents one of the solvers developed for the modeling approach. This method is detailed in the following section. Figs. 5.28 through 5.36 contain the code for this method. Fig. 5.27 outlines each step executed in the **IntegrateRK4HybridSens** method.

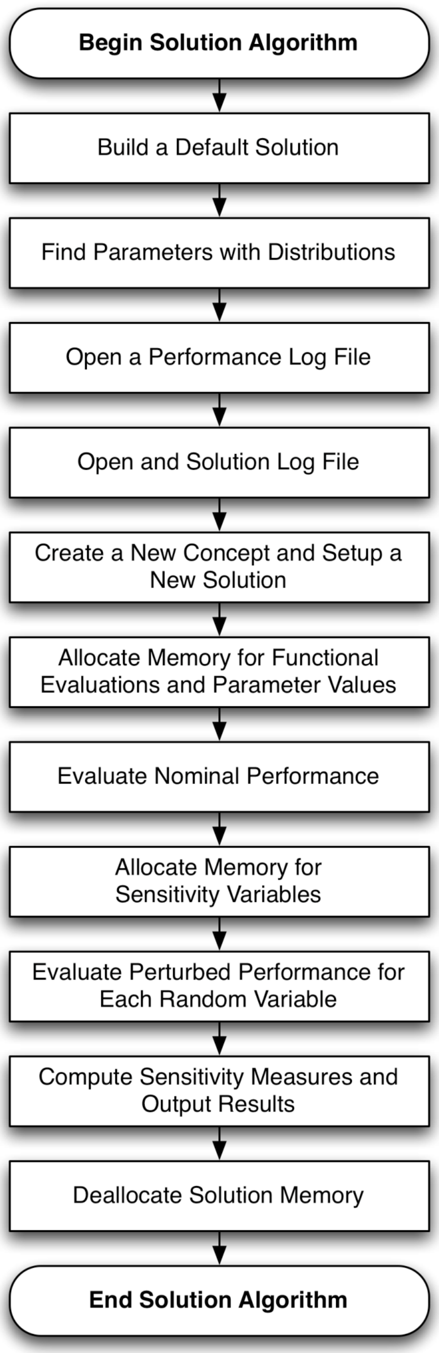
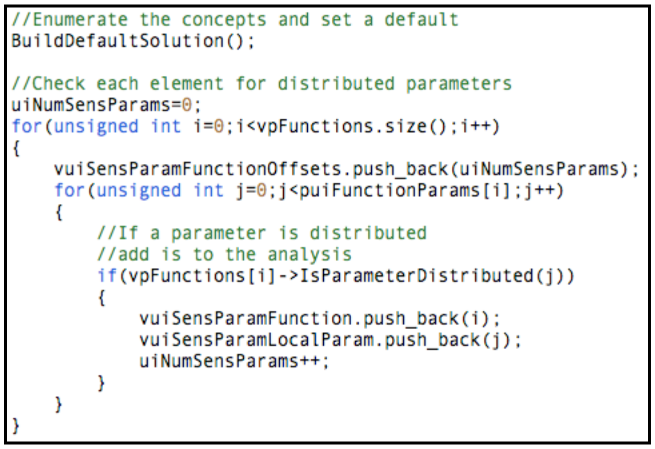

Begin Solution Algorithm
Build a Default Solution
Find Parameters with Distributions
Open a Performance Log File
Open and Solution Log File
Create a New Concept and Setup a New Solution
Allocate Memory for Functional Evaluations and Parameter Values
Evaluate Nominal Performance
Allocate Memory for Sensitivity Variables
Evaluate Perturbed Performance for Each Random Variable
Compute Sensitivity Measures and Output Results
Deallocate Solution Memory
End Solution Algorithm

Fig. 5.27. Model Solution Algorithm Flow Chart

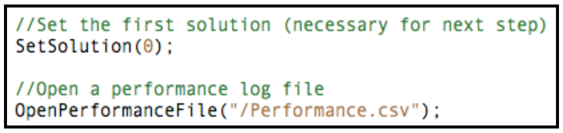
The **IntegrateRK4HybridSens** method integrates the model using a fixed time step and a Runge-Kutta 4 integration method and performs the hybrid sensitivity analysis method shown in Section 4. The first task in solving the system model is to build a nominal configuration for the system and enumerate the number of total available concepts based on the number of configurations loaded for each element. For example, if there are three elements in a system and each element has two values, the total number of configurations is 2*2*2 = 8. Each of these configurations is assigned a unique integer identifier in sequence within the **BuildDefaultSolution** method. Next, the number of parameters to be included in the sensitivity analysis is computed. This is done by looping through each element and checking to see which local parameters have been flagged as having distributions (this is defined in the text input file as seen in Fig. 5.6). The code that builds the default solution and counts parameters appears in Fig. 5.28.



```
//Enumerate the concepts and set a default
BuildDefaultSolution();

//Check each element for distributed parameters
uiNumSensParams=0;
for(unsigned int i=0;i<vpFunctions.size();i++)
{
    vuiSensParamFunctionOffsets.push_back(uiNumSensParams);
    for(unsigned int j=0;j<puiFunctionParams[i];j++)
    {
        //If a parameter is distributed
        //add is to the analysis
        if(vpFunctions[i]->IsParameterDistributed(j))
        {
            vuiSensParamFunction.push_back(i);
            vuiSensParamLocalParam.push_back(j);
            uiNumSensParams++;
        }
    }
}
```

Fig. 5.28. IntegrateRK4HybridSens Initialization Code (1)

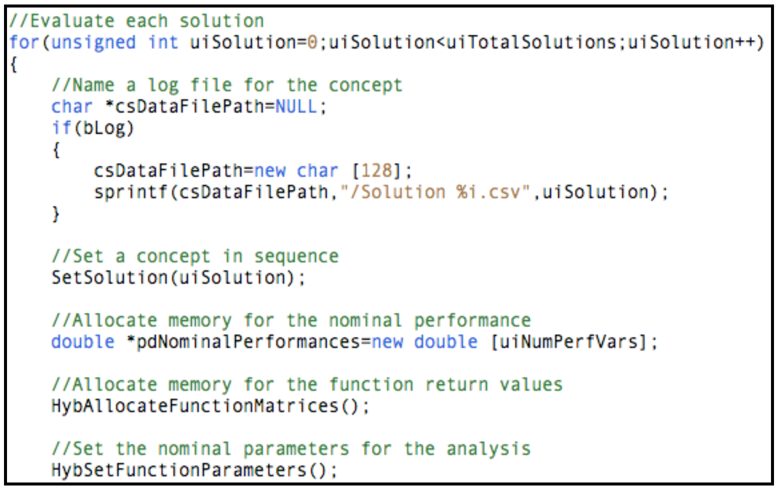
Next, the first enumerated solution is set as the current working solution and a performance file is opened. A solution must be set to open the file in order to generate the file's header (the performance log is a comma-delimited file that can be opened with any common spreadsheet program). The code for these functions is shown in Fig. 5.29.



```
//Set the first solution (necessary for next step)
SetSolution(0);

//Open a performance log file
OpenPerformanceFile("/Performance.csv");
```

Fig. 5.29. IntegrateRK4HybridSens Initialization Code (2)

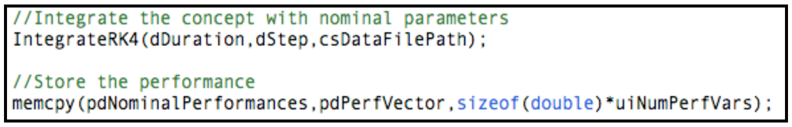
For each enumerated configuration, the nominal performance and sensitivity measures are then computed. The code for these computations is shown in Fig. 5.30. The first step is to create a log file for the results of the solution for the current active configuration. Next, the active configuration is set using the **SetSolution** method. A vector is then created to hold the performance evaluations to be generated from the model solution. Methods are then called to allocate memory to a set of matrices that hold the performance evaluations used in the sensitivity analysis and the values of the parameters used in the analysis.



```
//Evaluate each solution
for(unsigned int uiSolution=0;uiSolution<uiTotalSolutions;uiSolution++)
{
    //Name a log file for the concept
    char *csDataFilePath=NULL;
    if(bLog)
    {
        csDataFilePath=new char [128];
        sprintf(csDataFilePath,"/Solution %i.csv",uiSolution);
    }

    //Set a concept in sequence
    SetSolution(uiSolution);

    //Allocate memory for the nominal performance
    double *pdNominalPerformances=new double [uiNumPerfVars];

    //Allocate memory for the function return values
    HybAllocateFunctionMatrices();

    //Set the nominal parameters for the analysis
    HybSetFunctionParameters();
```

Fig. 5.30. IntegrateRK4HybridSens Solution Loop (1)

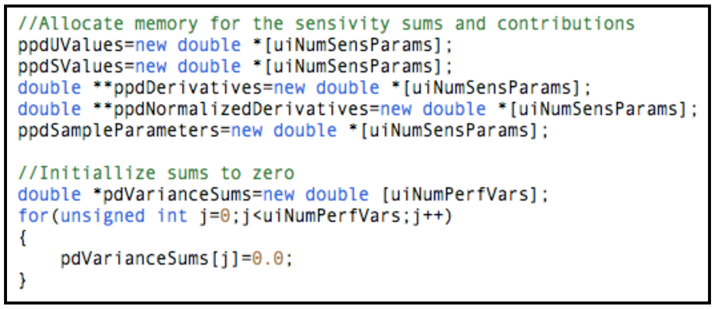
The next step in the solution is to integrate the system for the nominal parameterization of the current configuration. The results of this integration are then stored in the nominal performance vector. These steps are shown in the code in Fig. 5.31.



```
//Integrate the concept with nominal parameters
IntegrateRK4(dDuration,dStep,csDataFilePath);

//Store the performance
memcpy(pdNominalPerformances,pdPerfVector,sizeof(double)*uiNumPerfVars);
```

Fig. 5.31. IntegrateRK4HybridSens Solution Loop (2)

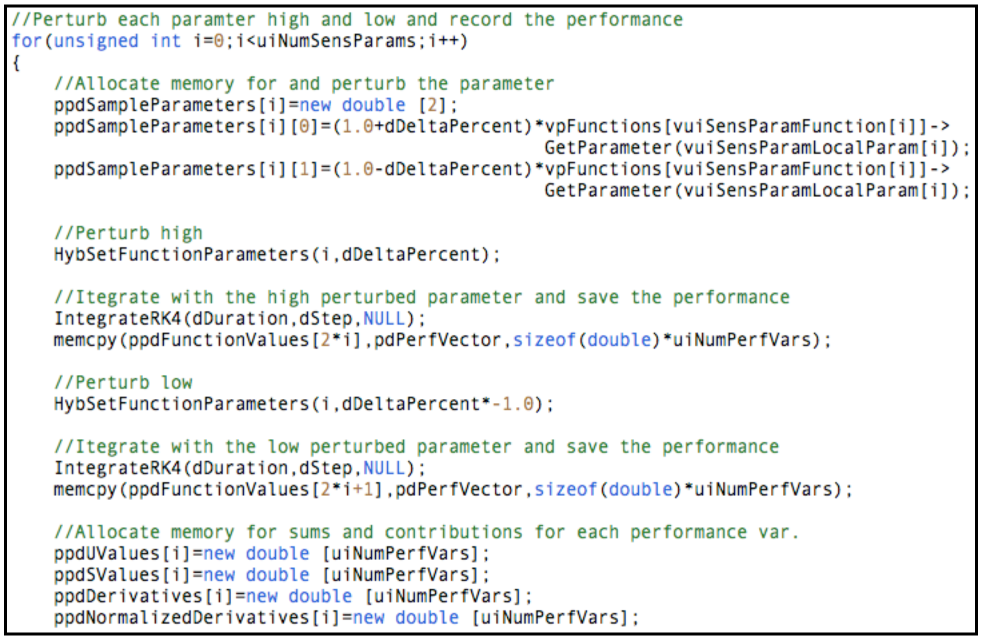
After the calculation of nominal performance, a sensitivity analysis is performed. For this analysis, arrays are created to hold the contributions and contribution sums for the analysis. The creation and initialization of these arrays is shown in Fig. 5.32.



```
//Allocate memory for the sensivity sums and contributions
ppdUValues=new double *[uiNumSensParams];
ppdSValues=new double *[uiNumSensParams];
double **ppdDerivatives=new double *[uiNumSensParams];
double **ppdNormalizedDerivatives=new double *[uiNumSensParams];
ppdSampleParameters=new double *[uiNumSensParams];

//Initiallize sums to zero
double *pdVarianceSums=new double [uiNumPerfVars];
for(unsigned int j=0;j<uiNumPerfVars;j++)
{
    pdVarianceSums[j]=0.0;
}
```

Fig. 5.32. IntegrateRK4HybridSens Solution Loop (3)

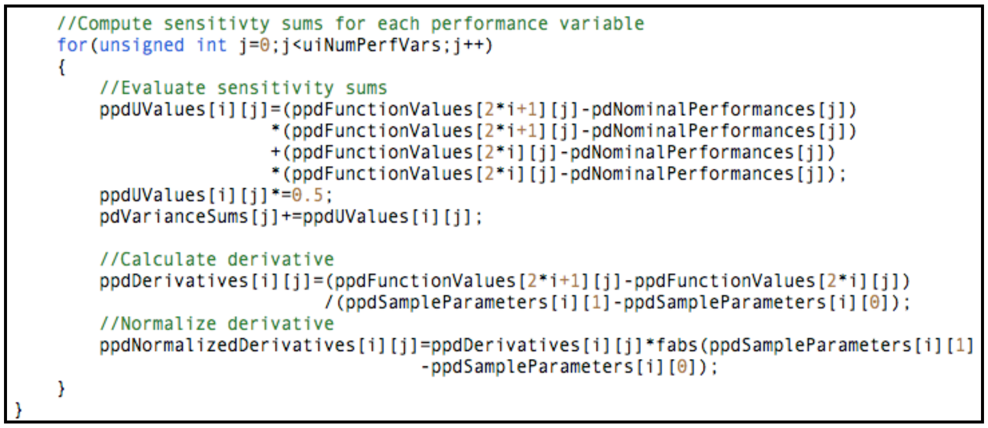
The actual sensitivity analysis is performed in the code shown in Fig. 5.33. For each parameter in the analysis, a high and low perturbation is generated and the system's performance is evaluated at each of the perturbed values. The performances are stored and memory is allocated to store the contributions and contribution sums for each performance variable along with derivative information.



```
//Perturb each paramter high and low and record the performance
for(unsigned int i=0;i<uiNumSensParams;i++)
{
    //Allocate memory for and perturb the parameter
    ppdSampleParameters[i]=new double [2];
    ppdSampleParameters[i][0]=(1.0+dDeltaPercent)*vpFunctions[vuiSensParamFunction[i]]->
                                              GetParameter(vuiSensParamLocalParam[i]);
    ppdSampleParameters[i][1]=(1.0-dDeltaPercent)*vpFunctions[vuiSensParamFunction[i]]->
                                              GetParameter(vuiSensParamLocalParam[i]);

    //Perturb high
    HybSetFunctionParameters(i,dDeltaPercent);

    //Itegrate with the high perturbed parameter and save the performance
    IntegrateRK4(dDuration,dStep,NULL);
    memcpy(ppdFunctionValues[2*i],pdPerfVector,sizeof(double)*uiNumPerfVars);

    //Perturb low
    HybSetFunctionParameters(i,dDeltaPercent*-1.0);

    //Itegrate with the low perturbed parameter and save the performance
    IntegrateRK4(dDuration,dStep,NULL);
    memcpy(ppdFunctionValues[2*i+1],pdPerfVector,sizeof(double)*uiNumPerfVars);

    //Allocate memory for sums and contributions for each performance var.
    ppdUValues[i]=new double [uiNumPerfVars];
    ppdSValues[i]=new double [uiNumPerfVars];
    ppdDerivatives[i]=new double [uiNumPerfVars];
    ppdNormalizedDerivatives[i]=new double [uiNumPerfVars];
```

Fig. 5.33. IntegrateRK4HybridSens Solution Loop (4)

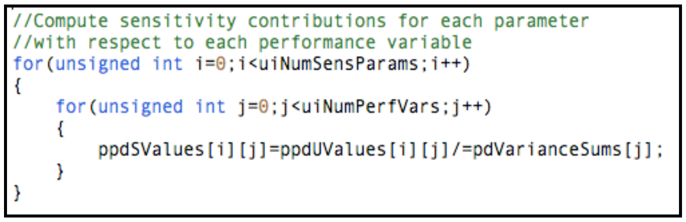
Next, sensitivity contribution sums are computed and stored in the **ppdUValues** array. This array is indexed by parameter then performance variable. Derivatives are computed as well and normalized. The code for these computations is shown in Fig. 5.34.



```
    //Compute sensitivty sums for each performance variable
    for(unsigned int j=0;j<uiNumPerfVars;j++)
    {
        //Evaluate sensitivity sums
        ppdUValues[i][j]=(ppdFunctionValues[2*i+1][j]-pdNominalPerformances[j])
                        *(ppdFunctionValues[2*i+1][j]-pdNominalPerformances[j])
                        +(ppdFunctionValues[2*i][j]-pdNominalPerformances[j])
                        *(ppdFunctionValues[2*i][j]-pdNominalPerformances[j]);
        ppdUValues[i][j]*=0.5;
        pdVarianceSums[j]+=ppdUValues[i][j];

        //Calculate derivative
        ppdDerivatives[i][j]=(ppdFunctionValues[2*i+1][j]-ppdFunctionValues[2*i][j])
                             /(ppdSampleParameters[i][1]-ppdSampleParameters[i][0]);
        //Normalize derivative
        ppdNormalizedDerivatives[i][j]=ppdDerivatives[i][j]*fabs(ppdSampleParameters[i][1]
                                        -ppdSampleParameters[i][0]);
    }
}
```

Fig. 5.34. IntegrateRK4HybridSens Solution Loop (5)

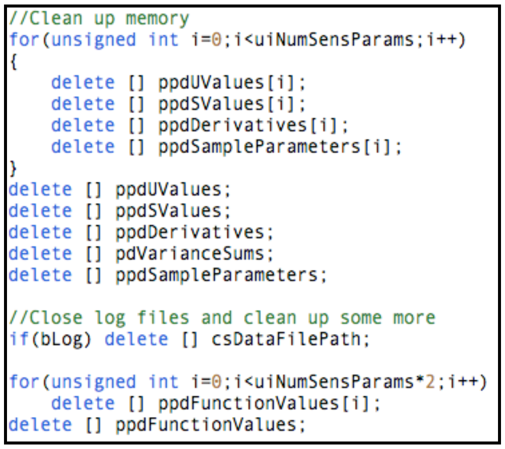
The sensitivity contribution sums are then normalized to produce sensitivity contribution measures as shown in Fig. 5.35.



```
//Compute sensitivity contributions for each parameter
//with respect to each performance variable
for(unsigned int i=0;i<uiNumSensParams;i++)
{
    for(unsigned int j=0;j<uiNumPerfVars;j++)
    {
        ppdSValues[i][j]=ppdUValues[i][j]/=pdVarianceSums[j];
    }
}
```

Fig. 5.35. IntegrateRK4HybridSens Solution Loop (6)

After the computation of the sensitivity measures for each parameter, memory is cleaned up and the log files are closed as shown in Fig. 5.36.



```
//Clean up memory
for(unsigned int i=0;i<uiNumSensParams;i++)
{
    delete [] ppdUValues[i];
    delete [] ppdSValues[i];
    delete [] ppdDerivatives[i];
    delete [] ppdSampleParameters[i];
}
delete [] ppdUValues;
delete [] ppdSValues;
delete [] ppdDerivatives;
delete [] pdVarianceSums;
delete [] ppdSampleParameters;

//Close log files and clean up some more
if(bLog) delete [] csDataFilePath;

for(unsigned int i=0;i<uiNumSensParams*2;i++)
    delete [] ppdFunctionValues[i];
delete [] ppdFunctionValues;
```

Fig. 5.36. IntegrateRK4HybridSens Solution Loop (7)

The analysis shown in Figs. 5.30 through 5.36 is completed for each configuration of the system and saved to the performance log file. At the completion of the solution of all concepts, the method cleans up memory, closes the log file and returns.

Once assembled, the actual system of equations resulting at each time step is solved using the KINSOL portion of the SUNDIALS numerical solution package. SUNDIALS is a set of equation solvers developed at the Lawrence Livermore National Laboratory [41]. KINSOL is C-based solver for systems of non-linear equations based on Newton-Krylov method. To implement KINSOL in the developed modeling framework, a wrapper class was developed using C++. This wrapper class, **Solver**, is initialized with a pointer (a reference) to the model being solved. The **BModel** base class contains the methods necessary to format and pass the solution vector and residuals to the solver in the format required by KINSOL.

5.4. FRAMEWORK EXAMPLE

To illustrate the operations performed during the model assembly and solution process, the mapping process of the example outlined in Section 5.2 is included. As shown in the code in Figure 5.14, the *FullSensTest* model uses three model elements and a number of causal and acausal variables. The details of each of these elements, including number of local equations, number of local flow variables, number of causal inputs and outputs and the global identifier that would be assigned by the model assembler is shown in Table

5.3. As seen in this table, there are four total acausal equations, seven total acausal flow variables and three causal variables (each has an input and output). Each model element is assigned a global ID starting with 0 (BME_x).

Table 5.3. Model Element Descriptions

BModelElement	Eqs.	Flow Vars.	FF Inputs	FF Outputs	Global ID
PROV_ELCE_II_0000_a	1	2	0	1	BME_0
CONV_ELCE_ROTE_II_0000_a	2	3	2	0	BME_1
EXPR_ROTE_II_0000_a	1	2	1	2	BME_2

Each flow variable added to the model is listed in Table 5.4. In this table, the flow variable name (using the FourCC naming convention), a descriptive name, origin element, destination element and Global ID are shown. There are four total flow variables in the model, the last flow variable shown, ROTE_II_2a_0_1, is an internal variable in this model (it is not passed from one element to another but is internally needed for integration). Each flow variable is assigned a Global ID in the form of GFV_x starting with x=0.

Table 5.4. Flow Variable Descriptions

Flow Variables	Description	Origin	Destination	Global ID
ELCE_II_2a_0_0	Accumulator Voltage	PROV_ELCE_II_0000_a	CONV_ELCE_ROTE_II_0000_a	GFV_0
ELCE_II_2a_0_1	Accumulator Current	PROV_ELCE_II_0000_a	CONV_ELCE_ROTE_II_0000_a	GFV_1
ROTE_II_2a_0_0	Load Torque	CONV_ELCE_ROTE_II_0000_a	EXPR_ROTE_II_0000_a	GFV_2
ROTE_II_2a_0_1	Load Accel	EXPR_ROTE_II_0000_a	EXPR_ROTE_II_0000_a	GFV_3

The causal variables are defined in a similar manner to the acausal flow variables. These variables are described in Table 5.5 along with a Global ID in the GFF_x format where x again starts at 0.

Table 5.5. Causal Variable Descriptions

FF Causal Var.	Description	Origin	Destination	Global ID
CNTL_I_1a_0_0	Throttle	EXPR_ROTE_II_0	CONV_ELCE_ROTE_II_0	GFFI_0
STAS_I_1a_0_0	Load Speed	EXPR_ROTE_II_0	CONV_ELCE_ROTE_II_0	GFFI_1
STAS_I_1a_1_0	Charge	PROV_ELCE_II_0	EXPR_ROTE_II_0	GFFI_2

For each model element, there is a set of local variables. These local variables are used for the actual description of the element's behavior and must be defined independently from the global variables to allow model re-use and automated assembly. These model elements are described in Tables 5.6 through 5.8 respectively. In these tables, the local identifiers for each flow, causal, integral and residual variable are shown.

Table 5.6. PROV ELCE Element Description

BModelElement	PROV_ELCE_II_0000_a	Local ID
Flow Variables	ELCE_II_2a_0_0	LFV_0
	ELCE_II_2a_0_1	LFV_1
FF Inputs	None	N/A
FF Outputs	STAS_I_1a_1_0	LFFO_0
Integrals	ELCE_II_2a_0_1	LFVI_1
Residuals	Voltage Sum	LRF_0

Table 5.7. CONV ELCE to ROTE Element Description

BModelElement	CONV_ELCE_ROTE_II_0000_a	Local ID
Flow Variables	ELCE_II_2a_0_0	LFV_0
	ELCE_II_2a_0_1	LFV_1
	ROTE_II_2a_0_0	LFV_2
FF Inputs	CNTL_I_1a_0_0	LFFI_0
	STAS_I_1a_0_0	LFFI_1
FF Outputs	None	N/A
Integrals	None	N/A
Residuals	Torque Calculation	LRF_0
	Current Calculation	LRF_1

Table 5.8. EXPR ROTE Element Description

BModelElement	EXPR_ROTE_II_0000_a	Local ID
Flow Variables	ROTE_II_2a_0_0	LFV_0
	ROTE_II_2a_0_1	LFV_1
FF Inputs	STAS_I_1a_1_0	LFFI_0
FF Outputs	CNTL_I_1a_0_0	LFFO_0
	STAS_I_1a_0_0	LFFO_1
Integrals	ROTE_II_2a_0_1	LFVI_1
Residuals	Torque/Accel Equation	LRF_0

To assemble the models, a set of maps is created to relate the local identity of variables to a set of global identifiers. The implementation of these maps in the framework is detailed in Section 5.3.2. The result of the application of this code is shown in Tables 5.9 through 5.12. The first table (5.9) shows the results of applying the **BuildFeedForwardMaps** method. The result of this method is a map (**ppuiFeedForwardInputMap**) that contains a global reference to each local feedforward input and output for each model element. As shown in this table, the first model element *PROV_ELCE_II_0000_a* (Global ID = BME_0) has no feedforward inputs (thus the elements of the array are empty). The second model element in the system, *CONV_ELCE_ROTE_II_0000_a* (Global ID = BME_1), has two local feedforward inputs that correspond to the GFF_0 and GFF_1 global causal variables respectively. The final model element, *EXPR_ROTE_II_0000_a* (Global ID = BME_2), has one feedforward input that corresponds to the GFF_2 global variable.

Table 5.9. Feedforward Input Map

ppuiFeedForwardInputMap	LFFI_0	LFFI_1
BME_0	Empty	Empty
BME_1	GFF_0	GFF_1
BME_2	GFF_2	Empty

The same mapping process is completed for the feedforward outputs and is shown in Table 5.10. In this table, global element BME_0 outputs global variable GFF_2. Global element BME_1 outputs no feedforward variables. Finally, global element BME_2 outputs both the GFF_0 and GFF_1 variables. Both tables (5.9 and 5.10) are generated automatically based on the origin and destination information supplied by the modeler.

Table 5.10. Feedforward Output Map

ppuiFeedForwardOutputMap	LFFO_0	LFFO_1
BME_0	GFF_2	Empty
BME_1	Empty	Empty
BME_2	GFF_0	GFF_1

From the input and output maps, an order of evaluation is generated for the causal operations of the model elements. Since the first element of the model (BME_0) has no causal inputs, it can be evaluated first. The second model element (BME_1) requires variables GFF_0 and GFF_1 so it cannot be evaluated until these variables have been produced. Since element BME_2 produced these values, it is added to the evaluation order after BME_0. Now, all of the variables required for BME_1 are available so it can be added to the evaluation order. A description of the code that performs these operations along with the code itself appears in Section 5.3.2. The evaluation order is shown in Table 5.11.

Table 5.11. Feedforward Evaluation Order

puiFeedForwardOrder	**Global ID**
First Evaluation	BME_0
Second Evaluation	BME_2
Third Evalution	BME_1

For the acausal flow and internal variables, a local-to-global routing map is required. This map is shown in Table 5.12 for the *FullSensTest* model. As described in Section 5.3.2, this table (**ppuiStateMap**) has two indices, the first being the model element and the second being the local variable in that element. This map is generated automatically using the flow variable origin and destination information provided by the user. As seen in Table 5.12, for the *FullSensTest* model the first function (BME_0) has two local flow variables (LFV_0 and LFV_1) that correspond to the GFV_0 and GFV_1 global variables respectively. The next element (BME_1) has three local flow variables that correspond to the GFV_0 through GFV_2 global variables. Finally, the BME_2 element has two local flow variables: GFV_2 and GFV_3. The generation of this routing map allows the solver to treat the set of elements and local variable as a single model with one vector of variables.

Table 5.12. Flow Variable Map

ppuiStateMap	**LFV_0**	**LFV_1**	**LFV_2**
BME_0	GFV_0	GFV_1	Empty
BME_1	GFV_0	GFV_1	GFV_2
BME_2	GFV_2	GFV_3	Empty

The same mapping process is used to transform the local residual evaluations to a single vector of global residuals. The global variable and global residuals are passed to the **Solver** class and used by the KINSOL solver to solve the resulting non-linear system. For a numerical example of the solution of the *FullSensTest* model see Section 4. For a more detailed system model, see the hybrid powertrain example presented in Section 6.

5.5. CONCLUSIONS

The modeling framework presented here enables a behavioral model of a system to be decomposed into elements based on functionality then automatically assembled and solved. The framework also allows a sensitivity analysis to be performed on the resulting model. This work is an extension of the function-based behavioral modeling and sensitivity work presented in Sections 3.1 and 3.2. The framework itself was created as a platform for evaluating and exploring the use of functionality-based behavioral modeling. Rather than adapt an existing component-based modeling scheme to the needs identified for the modeling framework, a from-scratch approach was used.

This approach was selected due to the lack of support for functional decompositions in existing methods and an opportunity to investigate the specific challenges necessary in producing a complete and operational model assembly and solution tool. The resulting framework and modeling tools have been implemented on several systems from the three-function test model shown here to the full design engineering example presented in Section 6. Developing a custom framework provided key insight into the challenges and opportunities present in modeling the behavior of complex systems that would have been absent from adapting existing technologies to perform this task. This insight includes the usefulness of a general purpose programming language, such as C++, in the modeling process. A general purpose language allows complete access to the tools and potential of a modern computer. However, the strict use of such a language generally limits the solution techniques for models to direct numerical methods. Specific modeling languages, such as Modelica, allow symbolic manipulation that is not possible with traditional programming languages at the expense of the flexibility of a general purpose programming language. Current trends in modeling languages, such a Modelica, suggest a movement towards incorporating features from general purpose languages. Object-oriented syntaxes and the ability to export simulation code in general purpose languages (such a C) are common in modern modeling languages. However, after extensive work exploring the needs of a modeling tool for use in engineering design, it is the view of the author that existing programming languages are sufficient for all modeling needs except symbolic representation of equations. As made apparent in the examples presented in this work, a variety of systems can be sufficiently modeled using a general-purpose language. Rather than add the feature set of general purpose programming languages to modeling languages (or general-purpose language exporters), it seems that adding the symbolic representation and solution capability to existing general purpose languages makes more sense. Essentially, it is suggested that adding one feature to an existing and robust platform entails less risk and effort than adding an entire world of features to a platform designed originally for one purpose. Having stated this, the current state-of-the-art in modeling languages and programming languages has forced this work to use one platform for its implementation. Due to the robustness and flexibility of programming languages versus current modeling languages, a programming language was selected for this work and all examples were successfully completed using it.

6. THE EARLY DESIGN OF A FORMULA HYBRID POWERTRAIN

This section contains a comprehensive example that illustrates the use of the function-based behavioral modeling and sensitivity analysis tools. The computational framework shown in Section 5 was used to implement the example and produce the included results.

6.1. EXAMPLE INTRODUCTION

To demonstrate the application of functionally derived behavioral models within the context of a complex system's design process, a comprehensive example was completed. Currently, there is not an agreed upon definition for a complex system. The definition of the term "system" itself, is however well defined. In general, a system is a set of interconnected parts that act together as a whole [24]. In the context of this work, a complex system must exhibit the following attributes:

1. A large number of functional elements,
2. Elements from varying engineering disciplines,
3. Emergent behavior resulting from interactions (the overall functionality is significantly more complex than the functionality of the elements).

In addition to satisfying the goal of being a complex system, the following criteria were used to select an appropriate example for illustrating the application of the functionally derived behavioral modeling work. The complex system should:

1. Be primarily, but not exclusively, comprised of elements from the author's engineering domain (mechanical),
2. Have a clearly defined customer and pre-existing set of needs,
3. Be appropriately scoped in scale,
4. Be topical and relevant to the challenges faced by today's engineering designers,
5. Be capable of being assessed by a set of well-defined metrics with prior examples of the design problem being available for comparison.

The first attribute is a practical requirement, in order to complete the design process in a reasonable amount of time without significant resources from other disciplines, it is necessary that most of the elements of the system be from the mechanical engineering discipline. However, the selected system must also include elements from other disciplines in order to satisfy the requirement of being considered complex.

The second attribute is necessary to ensure that the system is designed to solve an existing problem with an existing set of needs. This prevents the designer, in this case the author, from crafting the solution to fit his own needs rather than the needs of the customer. Additionally, this set of needs must be clear to avoid the influence of the designer's interpretation of the customer's needs.

The third attribute is required due to the availability of resources and the time allocated to the project. The selected system must be sufficiently complex and labor intensive to satisfy the overall objective of the work but also must be scoped so that the design process can be completed in a reasonable amount of time with the available resources.

Next, the example must be relevant to current engineering practices. Specifically, the example must reflect the challenges faced by modern system designers and should be of interest to the general populace as well. The problem must be interesting and also have practical value in its completion.

Finally, the successful completion of the example should be capable of being assessed by a clear set of metrics. This is to ensure that at the conclusion of the design process, a significant improvement over current solutions to the design problem has resulted.

To this end the example that was selected for the design process was a Formula Hybrid racecar. Formula Hybrid is a recently conceived collegiate engineering design competition that is a spin-off of the Formula SAE racing series. The objective of the Formula Hybrid competition is "to encourage and promote the development of high-efficiency automotive drive trains" [42]. The competition uses the existing set of Formula SAE rules [43]. For the Formula Hybrid competition, an open-cockpit racecar that utilizes a combination of electrical and internal combustion engine based propulsive energy must be designed, constructed and raced.

This example was chosen due to several factors in addition to its satisfaction of the previously described attributes. These factors will be presented next along with a discussion of how the chosen example satisfies the necessary attributes for the project.

A primary factor in the selection of the example was that Texas A&M's Formula SAE race team was switching to compete in Formula Hybrid in the 2009 competition year. As a result, a preliminary research class was conducted in the Spring 2008 Semester to explore various aspects of the competition and high-level requirements for the design of a hybrid racecar. The activities conducted in the class represented some of the earliest stages in a structured design process and will be included later in the presentation of this work. By selecting the design of a Formula Hybrid racecar as the primary example of this work, it will be possible to leverage the work done as part of the research class and provide a significant advantage to Texas A&M's Formula Hybrid team in future years.

The Formula Hybrid example also satisfies the previously stated project attributes. Due to the author's experience as a mechanical engineer and previous Formula SAE design work, the example fits the first listed criteria. Additionally, due to the nature of the competition, the customers and their requirements are clear and well-defined. For Formula Hybrid, the customer for the car is a weekend autocrosser [43] and the customer's needs can be directly obtained through the Formula SAE and Formula Hybrid rule book. The project is also adequately scoped for the needs of this example. Due to prior experience and the scale of the early design process it will be possible to complete the task in the allotted time. Additionally, the design of a hybrid vehicle is very relevant to today's engineering world as well as the consumer market in light of global concerns into the use of alternative energy sources. The success of the product can also be benchmarked using the points system supplied by the Formula SAE and Hybrid rules. This will enable different design concepts to be compared to each other as well as to prior Formula Hybrid entries.

Most importantly, the Formula Hybrid example represents a sufficiently complex system. The design of a Formula SAE car itself presents significant challenges but is generally limited to a few engineering domains (primarily mechanical with some aerospace aspects). Also, Formula SAE is a mature competition and as a result there has been significant evolutionary convergence within the top teams. This leaves few opportunities for novel designs that significantly improve performance over existing levels. Formula Hybrid is a newer competition with only two years of competition history and includes all of the design challenges of Formula SAE with the addition of more challenges through the increased emphasis of the electrical (power systems) and computer engineering (control systems) disciplines. The relative age of the competition along with the increased complexity means that Formula Hybrid offers a significant opportunity to develop designs that provide significant improvements to racecar performance. The relevance to modern engineering design challenges also means that there is the potential to apply these results to the industry.

6.1.1. Scoping the Example

The complete design of a Formula Hybrid vehicle is a significant task and is generally completed by a large group of students (up to 50 in the case of Texas A&M). Thus, the design problem was scoped down to a single system of the car that demonstrated all of the required aspects of a complex system. The selected system was the powertrain as this represents the most significantly changed system over a traditional FSAE car and includes the aspects of hybrid vehicle design that are most critical to its success. The powertrain system of a car includes the aspects of the car that store, supply, convert and transfer the propulsive energies used to accelerate the car in the longitudinal direction. In completing the design process on this system, the formal design method proposed in Section 2.5 was used.

6.1.2. The Design Process

As described in Section 2.5, the design method used in this example includes design decomposition and design synthesis chains of activates. These two chains appear along with a grouping of related activities in Figs. 6.1 and 6.2.

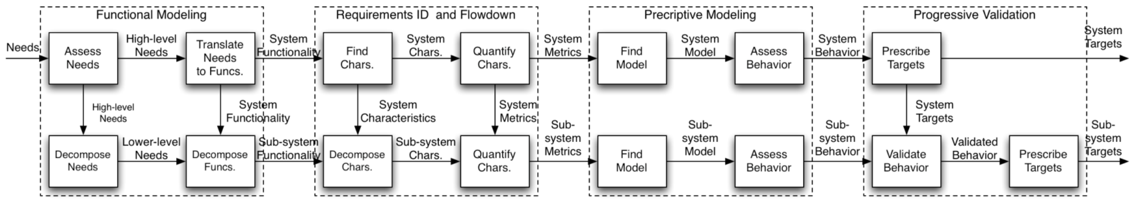


Fig. 6.1. Design Decomposition

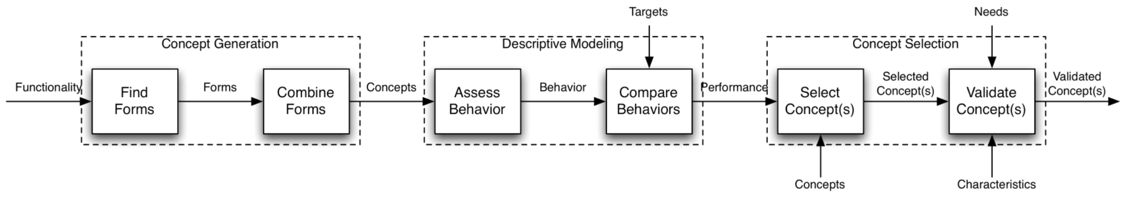


Fig. 6.2. Design Synthesis

Before the design process can be started, the activities recommended during Product Planning (in traditional design) and the pre-design process in the proposed method must be completed. The result of these activities is a product proposal that summarizes the vision, objectives and high-level requirements for the system to be designed. The following section presents the development of such a proposal for the Formula Hybrid powertrain example.

6.2. PRE-DESIGN ACTIVITIES

6.2.1. Developing a Vision

The first step in the pre-design phase of product development is to develop a vision. For this example, the product to be designed is a Formula Hybrid powertrain targeted for a 2009 competition entry. In the Formula SAE rules, the target market for the product is the non-professional weekend autocrosser [43]. The primary objective of the competition (as summarized in the FSAE rules) is "for the students to fabricate and demonstrate a prototype car for evaluation as a production item" [43]. Formula Hybrid does not explicitly change this customer but it can be assumed that the inclusion of hybrid components means that the ultimate customer has energy-efficiency on their mind in addition to on-track performance.

However, the demands of the hypothetical customer for Formula SAE (and Formula Hybrid as well) are usually ignored as they factor little into the scoring that ultimately determines the winner of the competition. In reality, the actual customer is the design team itself and their primary objective is usually to win the competition entered. As a result, the customer considered for this project is Texas A&M's Formula Hybrid team. Thus, the vision for the design process is to produce a concept that is capable of being handed over to the remainder of the design team for detailed and embodiment design within future competition entries.

6.2.2. Analyze the Market for Opportunities

Compared to the decades old Formula SAE program, Formula Hybrid is significantly younger with only two competitions (2007 and 2008) to date. As a result, there are fewer entries with 6 registered teams in 2007 and 16 in 2008. The age of the competition and the additional design challenges offered over Formula SAE produce the opportunity to develop truly novel concepts.

Since the product is prescribed by the Formula Hybrid competition rules (and hence the bulk of the Formula SAE rules), the potential product identification steps proposed by Pahl and Beitz [4] are unnecessary. Additionally, no current design exists for a Formula Hybrid car at Texas A&M. Thus, a "from-scratch" design process is required by the team to develop a hybrid racecar.

As described in the Formula SAE rules, the car is to be produced as a prototype for a manufacturing firm. The goals of the firm are to produce four units a day that cost less than $25,000 each. Once again, the described target for the car is not the actual target. Ultimately, the car will be produced by the design team and raced for one competition year (there is currently one competition for Formula Hybrid). Unlike Formula SAE, Formula Hybrid does not include a cost event as part of the competition. As a result, the actual production cost of the car is not a performance attribute but rather a feasibility attribute. For this study, the estimated powertrain budget will be $10,000.

In conclusion, the market for the car is the Texas A&M Formula Hybrid team. The cost for the product must be less than $10,000 and one unit will be produced. The next step in the design process is to use this information to develop a product proposal.

6.2.3. Formulate a Product Proposal with Objectives

A product proposal describes the product to be designed as well as its place in the market along with the highest-level objectives, requirements and functionality. Cost and budget requirements should be included as well.

The ultimate result of the early design process is a concept that is to be further developed into a complete design that will then be manufactured, tested and implemented within a complete Formula Hybrid entry. The overall functionality of the powertrain system can be simply summarized: develop and manage the propulsive energies required to produce a vehicle capable of winning the Formula Hybrid competition.

For this design project, the following objectives were established based on the needs of the team as well as the FSAE and FH rulebooks:

- Produce a design that can be practically implemented by the team,
- Produce a design that is capable of winning future competitions,
- Design must stay in budget,
- Produce a vehicle with very high performance (FSAE),
- Use as many common parts as possible (FSAE),
- Use good engineering practices (FSAE),
- Create an efficient product (FH),
- Produce a system that is easy to maintain (FSAE),
- Produce a system that is safe (FSAE).

From the overall functionality and the stated objectives, a set of high-level requirements was generated. To facilitate the development of requirements and provide a common representation format, a standard requirements table is used in the remainder of this work. The requirements table includes the object for the requirement, the level (system, sub-system, etc.), the requirement itself and any references for the requirement. Each requirement is listed in sentence form and uses the "shall" language to promote uniformity in the description. The high-level requirements for the FH design appear in Table 6.1.

Table 6.1. High Level Formula Hybrid Requirements

Req.#	The	Object	Level	Shall	Requirement
S1	The	powertrain	system	shall	be manufacturable by the target team
S2	The	powertrain	system	shall	be capable of winning the 2009 FH competition
S3	The	powertrain	system	shall	have an estimated cost of less than $10000
S4	The	powertrain	system	shall	enable a high level of performance
S5	The	powertrain	system	shall	use as many common parts as possible
S6	The	powertrain	system	shall	utilize good engineering practices
S7	The	powertrain	system	shall	be energy efficient
S8	The	powertrain	system	shall	be easy to maintain
S9	The	powertrain	system	shall	be safe

The first requirement involves the manufacturability of the product. Due to restrictions on the equipment available to the team and time limitations, the car must be relatively simple to produce. Texas A&M runs Formula SAE (and will run Formula Hybrid) as a senior design class with the first semester being design and the second semester being construction and testing. This schedule limits production time to a few months and as a result manufacturability is a highly important aspect of the car's design. This requirement is related to Requirements S3 and S5. Requirement S5 necessitates the use of common parts on the car wherever possible. This reduces the number of custom parts required and relieves some of the manufacturing burden from the team. Additionally, using common parts generally reduces the cost of the parts and helps the team meet the budgetary requirement (S3).

S2 is another highly important requirement. The powertrain must result in a car capable of winning competitions. In addition to serving as a senior design project, competing in industry sponsored projects serves as a recruiting tool and source of prestige for engineering departments. Texas A&M has enjoyed success in the Formula SAE competition and continuing this tradition in the Formula Hybrid competition is demanded. Requirement S4 is explicitly stated in the rules for Formula SAE but is also implicitly required to satisfy Requirement S2 (winning the competition).

Winning the Formula Hybrid competition requires outscoring the other participating teams based on the following points breakdown:

- Static events
    - Presentation – 100
    - Engineering Design – 100
- Dynamic Events
    - Acceleration – Electric – 75
    - Acceleration – Unrestricted – 75
    - Autocross – 150
    - Efficiency and Endurance – 400

In the context of this design exercise, the dynamic events and the engineering design events will be considered. The Presentation event is primarily a human performance-centric event rather than being car design and performance focused. The engineering design event involves the student design team defending their design choices and answering technical questions related to the car. It is difficult to factor this event into requirements for the car (due to the human judging element) so it will be assumed that methods used during the remainder of the design process will ensure that the team and car will be well-positioned for this event.

The dynamic events include two types of acceleration events, an autocross event and an efficiency and endurance event. Completing the electric acceleration event is a requirement in order to be considered a hybrid vehicle in the context of the competition and is worth 75 points. Both acceleration events involve completing a 75m straight run from a standing-stop. The time required to complete the runs is measured and compared to the other competition entries to produce a score. The electric-only acceleration requires the car to complete the event using only electrical energy stored by some means on board. The unrestricted acceleration event allows both electric and internal combustion power.

The autocross event involves completing a lap of a road-course style racetrack laid out on a large, flat surface that utilizes parking cones and/or temporarily marked lines to demarcate the track. This event begins from a standing start and the time required to complete the event is used for scoring. Hitting cones during the lap or going off course results in the addition of time penalties to the lap.

The endurance and efficiency event involves completing a roughly 22km race on a closed autocross-style course in two 11km segments (with different drivers for each segment). The time required to complete the event is used for scoring purposes. Like autocross, hitting cones or going off-course results in the addition of a time penalty.

Efficiency is not directly scored in the competition but a fixed amount of fuel is given to each team before the start of the endurance event. The maximum fuel given is approximately 85% of the fuel required for an average FSAE car to complete the same events and roughly corresponds to one gallon of gasoline. No points are awarded for any fuel not used in the endurance event. Essentially, the objective is to use the given quantity of fuel as efficiently and completely as possible while completing endurance as quickly as possible.

The same explicit and implicit relationship between Requirement S2 exists for Requirements S6 and S8. Requirement S6 states that good engineering practices should be used in the design of the car. Since this is judged in the design event, it directly affects the ability of the car to win. Additionally, good engineering practices are also necessary to produce a fast and reliable car so it indirectly affects this requirement as well. Requirement S8 as stated in the FSAE rules and is directly judged in the design event and indirectly affects the performance of the car should a maintenance activity be required during the competition.

Formula Hybrid rules necessitate improved fuel economy over a Formula SAE car. This leads to requirement S7, the need for the car to be fuel efficient (compared to an FSAE car). Since the car must complete all events on a fixed amount of energy, this directly affects the ability of the car to win (S2).

Requirement S9 is the most important requirement of all, the car must be safe. This requirement is related to Requirements S6 and S8. A system that utilizes good engineering practices is going to be safer than one that does not (S6). Additionally, a car that is easy to maintain reduces the avoidance of repairs due to excessive difficulty in performing the repair (S8). Requirement S9 is also directly implemented through the meeting of all of the technical rules and regulations associated with Formula SAE and Formula Hybrid.

The results of these pre-design activities have been summarized in the product proposal shown in Appendix 1. At the completion of this proposal, the actual design process of the system was initiated. The activities performed in this design process along with the results of the process are detailed in the following sections.

6.3. DESIGN PROCESS

6.3.1. Assessing Needs

Since most of the needs of the customer are described in the Formula SAE and Formula Hybrid rulebooks, performing a customer need assessment for this project is relatively simple compared to traditional design projects. The bulk of the customer's needs can be derived from the rules and the remainder assessed from any additional needs of the team. The objectives and requirements developed in the prior step represent most of these needs. Essentially, the primary needs of the customer are to produce a highly competitive design that meets all rules and regulations.

To satisfy this need, the team must outscore the other competition entries to the event. The Formula Hybrid competition is relatively new but there is data available that can be used to estimate the potential performance of the competitors. Table 6.2 presents the scoring results from the top five teams in the 2008 Formula Hybrid competition [42].

Since the points scoring is relative to the performance of the participating teams, it is difficult to use raw score as a competitive benchmark. However, the timing results for the events are available as well. The timing results can be used to develop a set of benchmarks for assessing the performance of the car to be designed and hence its ability to outscore competitive teams. During the performance target establishment activity later in the design process, this scoring information will be used to set target values for the performance of the car to be used in selecting potential concepts.

Table 6.2. Top Five Team Results from 2008 Formula Hybrid Competition[42]

Entry	Score	Design	Acceleration Electric	Acceleration Unrestricted	Autocross	Endurance
McGill	781.00	121.52	54.01	55.47	150.00	400.00
Embry-Riddle	629.88	200.00	28.25	43.87	113.75	244.01
Dartmouth	481.63	126.58	75.00	75.00	94.91	110.13
Illinois Inst. of Tech.	575.57	125.32			79.82	370.43
UW-Madison	256.71	67.09				189.62

6.3.2. Translating Needs Into Functionality

From the customer needs, a set of desired functions for the powertrain system can be developed. These functions can be defined and represented using the formal Functional Modeling method described in Section 3. The model used for the powertrain example appears in this section along with its derivation. The model, shown again in Fig. 6.3, represents the highest-level intended functionality of the hybrid powertrain system. In this example, only post IC transmission torque-coupled parallel hybrids are considered. This decision resulted from the consensus of the Texas A&M Formula Hybrid design team. This configuration represents one of the simplest hybrid powertrain configurations and represents the smallest change from Formula SAE architecture that produces a Formula Hybrid legal car. Series hybrid powertrains were not considered due to their increased complexity (the need for two electrical machines) and the absolute reliance on an electrical machine for all propulsive forces. Series coupled hybrids (in the context of Formula Hybrid) use an internal combustion engine connected to a generator, which is in turn connected to an energy storage device. The energy is then supplied to motor and finally to the drivetrain of the vehicle. Such a system requires two separate electrical machines, each of which carries a significant weight burden. Speed coupled parallel powertrains were not considered due to the added complexity of such systems over a fixed-ratio torque coupled system. Speed coupling requires a variable speed transmission between the motor and drive system of the car and adds significant complexity and weight over a torque-coupled system that uses a fixed mechanical coupling between the engine and motor.

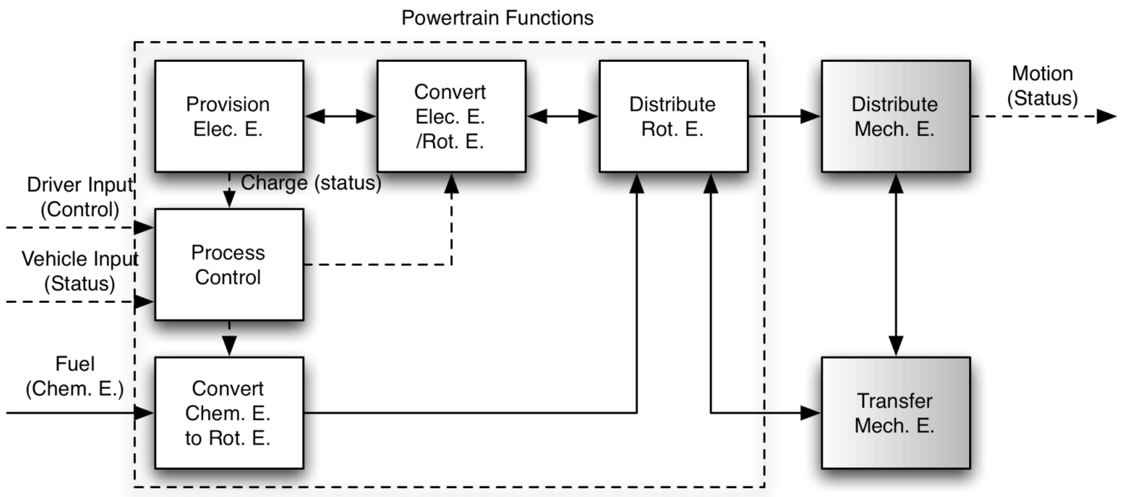


Fig. 6.3. Hybrid Powertrain Functional Model

6.3.3. Finding and Quantifying Characteristics of the System

Although the list of customer needs is relatively short and simple to describe, translating these needs into a comprehensive set of requirements is a significant challenge. High-level requirements have already been defined from the overall objectives of the powertrain system. However, to meet the need of passing all of the FSAE and Formula Hybrid rules and regulations, an extensive list of requirements was developed from the perspective of the entire car. These requirements were generated using the rulebooks as a guide. The requirements were then sorted and partitioned into manageable sets for use in analyzing specific requirements for the hybrid powertrain system.

The first step in this process was to group and re-order the requirements established during the product proposal and customer needs search. Some of the requirements were redundant and a hierarchy was not used during their development. To remedy this, a new requirements table was developed that utilized a hierarchical decomposition and numbering system. This updated table of system-level requirements appears in Table 6.3. This table represents a complete view of the car as a whole rather than a specific scoping to the powertrain system.

Table 6.3. Hybrid Racecar Structured Requirements

Req.#	The	Object	Level	Shall	Requirement
S1	The	car	system	shall	be manufacturable by the target team
S1.1	The	car	system	shall	use as many common parts as possible
S1.2	The	car	system	shall	include as few complex custom parts as possible
S1.3	The	car	system	shall	make use of efficient structures (simple yet strong)
S1.4	The	car	system	shall	be designed with manufacturability in mind
S1.5	The	car	system	shall	be designed with the team's budget in mind
S2	The	car	system	shall	be capable of winning the 2009 FH competition
S2.1	The	car	system	shall	meet all 2009 FH and relevant FSAE rules
S2.2	The	car	system	shall	pass all competition tests
S2.3	The	car	system	shall	be completed early enough to have sufficient test time
S2.4	The	car	system	shall	receive a competition score higher than all other entered teams
S2.4.1	The	car	system	shall	utilize a well-research, thorough and document design process (Design Event)
S2.4.2	The	car	system	shall	be aesthetically pleasing (Presentation and Design)
S2.4.3	The	car	system	shall	exhibit a high level of performance (Acceleration, Autocross and Endurance)
S2.4.4	The	car	system	shall	be energy efficient (Endurance)
S2.4.5	The	car	system	shall	utilize good engineering practices (All events)
S2.4.6	The	car	system	shall	be comfortable to its drivers (All events)
S2.4.7	The	car	system	shall	easy to setup and control (All events)
S2.4.8	The	car	system	shall	be reliable (All events)
S3	The	car	system	shall	be safe
S3.1	The	car	system	shall	be comfortable to its drivers
S3.2	The	car	system	shall	conform to FSAE and FH safety rules
S3.3	The	car	system	shall	utilize good engineering practices
S3.4	The	car	system	shall	be designed with safety in mind
S4	The	car	system	shall	be easy to maintain
S4.1	The	car	system	shall	use as many common parts as possible
S4.2	The	car	system	shall	be designed with maintenance in mind
S4.3	The	car	system	shall	be reliable (All events)

Satisfying the Formula Hybrid and relevant Formula SAE rules involves meeting a significant number of technical requirements. In the rulebooks, these requirements are stated in paragraph form and are roughly grouped by the impacted sub-system. The previously utilized requirements development process and representation format was used to translate the rulebooks into a set of engineering requirements. In total, over 300 requirements for the car were derived from the FSAE and Formula Hybrid rules. A complete list of these requirements appears in Appendix 2.

Only a fraction of these requirements are relevant to the early engineering design process. Most involve specific aspects of a physical implementation that cannot be addressed until significantly later in the design. As a result, the requirements had to be sorted to produce a set for consideration for the remainder of this design exercise. Each requirement was individually assessed for its relevance during conceptual design and regrouped to produce a new set of requirements. This reduced set appears in Table 6.4 and will be used for the remainder of the early design process.

Table 6.4. Early Design Requirements

The	Object	Level	Shall	Requirement	Ref.
The	car	system	shall	cost less than $25,000	1.2
The	car	system	shall	exhibit a high performance	1.2
The	car	system	shall	be designed and fabricated in accordance with good engineering practices	1.3
The	car	system	shall	be conceived, designed, fabricated and maintained by the student team	2.2.1
The	car	system	shall	be open-wheeled	3.1.1
The	car	system	shall	be open-cockpit	3.1.1
The	car	system	shall	have a wheelbase of at least 1525mm	3.1.2
The	car	system	shall	have four wheels	3.1.2
The	car	system	shall	have all four wheels not in a straight line	3.1.2
The	car	system	shall	have a smaller track that is no less than 75% of the larger track	3.1.3
The	car	system	shall	be equipped with a fully operating suspension sub-system	3.2.1
The	car	system	shall	have wheels greater than 203.2mm in diameter	3.2.3.1
The	car	system	shall	use any size or type of tire, slick or treaded as its dry tire	3.2.3.2
The	car	system	shall	have a steering sub-system that effects at least two wheels	3.2.4
The	car	system	shall	have a braking sub-system	3.2.5
The	car	system	shall	have a steering wheel assembly	3.4.6
The	car	system	shall	have a track and CG that combine to produce adequate rollover protection	3.4.8
The	car	system	shall	must not roll when tilted at angle of 60 deg to the horizontal (either direction) with the tallest driver	3.4.8.1
The	car	system	shall	have all aerodynamic or ground effect devices satisfy 3.7.1	3.7.1
The	car	system	shall	be capable of completing a 75m acceleration run in electric-only mode in less than 15s	FH-1.2
The	electrical propulsion	sub-system	shall	have a maximum voltage of 600V DC or AC RMS under all circumstances	FH-2.1
The	engine	sub-system	shall	be internal combustion, four-stroke, with a maximum displacement of 250cc	3.5.1.1FH
The	engine	sub-system	shall	be of modified or custom fabricated type and follow section 3.5.4FH or:	3.5.1.1.1FH
The	engine	sub-system	shall	be of stock type (section 3.5.1.1.2FH)	3.5.1.1.2FH
The	engine	sub-system	shall	use 93 octane pump, E-85 or Biodiesel	3.5.2FH
The	fuel	sub-system	shall	not alter the temperature of the fuel to improve fuel economy	3.5.2.1
The	fuel	sub-system	shall	not use any agents other than fuel and air	3.5.2.2

Table 6.4 Continued.

The	Object	Level	Shall	Requirement	Ref.
The	powertrain	sub-system	shall	be equipped with a muffler to reduce noise levels	3.5.5.1
The	accumulator	assembly	shall	shall meet the type and size rules listed in FH-2.5	FH-2.5
The	accumulator	assembly	shall	have a maximum standardized cost of $6000	FH-2.5
The	accumulator	assembly	shall	be of battery or capacitor type	FH-2.5
The	fuel tank	assembly	shall	be of any capacity	3.5.3.1FH
The	braking	sub-system	shall	act on all four wheels	3.2.5
The	braking	sub-system	shall	be operated by a single control	3.2.5
The	braking	sub-system	shall	have two independent hydraulic circuits with separate reservoirs	3.2.5
The	braking	sub-system	shall	be capable of locking all four wheels in the brake test	3.2.5
The	braking	sub-system	shall	not use "brake-by-wire"	3.2.5
The	braking	sub-system	shall	have its last 50% travel operate the hydraulic system described in 3.2.5 (the first 50% travel may be used for regen. braking)	3.2.5FH
The	steering	sub-system	shall	have less than +/-3 degrees of rear wheel steering	3.2.4
The	steering	sub-system	shall	be mechanically connected to front wheels	3.2.4
The	suspension	sub-system	shall	have shock absorbers front and rear	3.2.1
The	suspension	sub-system	shall	have a usable wheel travel of at least 50.8mm per wheel	3.2.1
The	suspension	sub-system	shall	have at least 25.4mm of jounce and 25.4mm of rebound	3.2.1
The	tire	component	shall	not use warmers or traction enhancers	3.2.3.2

This set of requirements can then be parsed to identify specific requirements relevant to the powertrain system. The results of this parsing process appear in Table 6.5.

Table 6.5. Filtered Early Design Requirements

The	Object	Level	Shall	Requirement	Reference
The	car	system	shall	cost less than $25,000	1.2
The	car	system	shall	exhibit a high performance	1.2
The	car	system	shall	be designed and fabricated in accordance with good engineering practices	1.3
The	car	system	shall	be conceived, designed, fabricated and maintained by the student team	2.2.1
The	car	system	shall	have four wheels	3.1.2
The	car	system	shall	have wheels greater than 203.2mm in diameter	3.2.3.1
The	car	system	shall	use any size or type of tire, slick or treaded as its dry tire	3.2.3.2
The	car	system	shall	be capable of completing a 75m acceleration run in electric-only mode in less than 15s	FH-1.2
The	electrical propulsion	sub-system	shall	have a maximum voltage of 600V DC or AC RMS under all circumstances	FH-2.1
The	engine	sub-system	shall	be internal combustion, four-stroke, with a maximum displacement of 250cc	3.5.1.1FH
The	engine	sub-system	shall	be of modified or custom fabricated type and follow section 3.5.4FH or:	3.5.1.1.1FH
The	engine	sub-system	shall	be of stock type (section 3.5.1.1.2FH)	3.5.1.1.2FH
The	engine	sub-system	shall	use 93 octane pump, E-85 or Biodiesel	3.5.2FH
The	fuel	sub-system	shall	not alter the temperature of the fuel to improve fuel economy	3.5.2.1
The	fuel	sub-system	shall	not use any agents other than fuel and air	3.5.2.2
The	powertrain	sub-system	shall	be equipped with a muffler to reduce noise levels	3.5.5.1
The	accumulator	assembly	shall	shall meet the type and size rules listed in FH-2.5	FH-2.5
The	accumulator	assembly	shall	have a maximum standardized cost of $6000	FH-2.5
The	accumulator	assembly	shall	be of battery or capacitor type	FH-2.5
The	fuel tank	assembly	shall	be of any capacity	3.5.3.1FH
The	braking	sub-system	shall	have its last 50% travel operate the hydraulic system described in 3.2.5 (the first 50% travel may be used for regen. braking)	3.2.5FH
The	tire	component	shall	not use warmers or traction enhancers	3.2.3.2

6.3.4. Develop Performance Metrics for Requirements

The next step in the design process is to distill the requirements into a set of performance metrics that can be used to assess the ability of concepts to meet the requirements and hence satisfy the customer's needs. These metrics are a set of numerical quantities (or logic statements) that quantify (or qualify) the satisfaction of requirements. For the hybrid powertrain the metrics to be used are shown in Table 6.6 along with units and the references requirements.

Table 6.6. Performance Metrics

Metric	Units	Reference Req.
Cost	U.S. Dollars	S1.5
Dynamic Event Score	Points (700)	S2
Electric Accel. Test	Pass/Fail	S2.2
Functional Sensitivities	Percentages	S2.4.7
Mass	kg	S2.4.5

The first metric to be assessed is cost. Cost is not directly associated with the performance of the car at competition but must be used to assess the ability of the team to construct the car. This metric was identified to represent Requirement S1.5, which involves the team keeping in its budget.

Requirement S2, developing a design that has the capability to win the competition, is one of the most important requirements. To assess the satisfaction of this requirement, the estimated dynamic event score metric will be used. This metric will be calculated based on estimates of event performance and will use the 2009 score normalization algorithms to compare concepts. Since efficiency is built into the scoring system, the car's energy efficiency will also be assessed by this metric.

To meet Requirement S2.2, the car must pass the electrical acceleration event. Otherwise, the car is not considered a hybrid and is not eligible for the competition. This requirement is assessed with a pass/fail metric that represents the estimated ability of the design to pass this event.

As this design process represents a first attempt at creating a hybrid powertrain system, understanding the parametric sensitivities of the system is necessary in order to allocate sufficient modeling and design resources in the embodiment and detailed stages of design. To this end, the functionality-based sensitivity approach presented in Section 4 will be applied during the design process of the powertrain system.

These metrics will be used to assess the performance of the various concepts that will be developed. However, these metrics only cover a fraction of the overall set of requirements. As a result, the satisfaction of the remaining requirements will have to be assessed on a concept-by-concept basis once a preliminary set of concepts has been developed and sorted using the metrics. Ultimately, this task lies outside the context of early design and is left to the embodiment and detailed design phases.

6.3.5. Finding a Model and Assessing Behavior

The development of the behavioral models for this system at the highest level are presented in the Type I analysis shown in Section 3. The result of this analysis for a test case is presented in that section. This same model was used for a first-pass analysis of the performance capabilities of the powertrain system in the context of the established performance metrics. To use this model, various parametric configurations were necessary. Although the development of a Type I model makes minimal assumptions regarding the form of a system, the parametric instantiation of the model requires knowledge of feasible values for the various parameters included in the model. For the Type I analysis of the hybrid powertrain, several solutions to each of the functions were identified through brainstorming sessions with the Texas A&M team and surveys of existing solutions to these functions. The parametric values used to represent these conceptual solutions appear in Table 6.7.

The automated model assembly and solution methods presented in Section 5 was used to evaluate the performance of each permutation of possible concepts using the identified solutions to the various powertrain functions. The results of this analysis were sorted based on predicted Formula Hybrid points using the total analyzed solution space to normalize the scores. The ranked concepts appear in Fig. 6.4. A complete listing of the results of this analysis appear in Appendix 5.

Table 6.7. Type I Model Parameter Values

Function	Concept	Parameter	Value	Units
Convert Chem. To Rot. E.	Small Industrial	Maximum Power	6.00	kW
		Efficiency	0.30	Unitless
		Cost	500.00	U.S. Dollars
		Mass	25.00	kg
	250CC MC 1	Maximum Power	31.00	kW
		Efficiency	0.30	Unitless
		Cost	2000.00	U.S. Dollars
		Mass	45.00	kg
	250CC MC 2	Maximum Power	27.00	kW
		Efficiency	0.30	Unitless
		Cost	1500.00	U.S. Dollars
		Mass	45.00	kg
	Hypothetical	Maximum Power	18.00	kW
		Efficiency	0.30	Unitless
		Cost	1000.00	U.S. Dollars
		Mass	35.00	kg
Convert Elec. E. to Rot. E.	Small Sep. Ex.	Maximum Power	10.00	kW
		Efficiency	0.90	Unitless
		Cost	1000.00	U.S. Dollars
		Mass	25.40	kg
	Medium Sep. Ex.	Maximum Power	15.00	kW
		Efficiency	0.90	Unitless
		Cost	1250.00	U.S. Dollars
		Mass	25.40	kg
	Large Sep. Ex.	Maximum Power	20.00	kW
		Efficiency	0.90	Unitless
		Cost	1400.00	U.S. Dollars
		Mass	25.40	kg
	Small AC Ind.	Maximum Power	13.00	kW
		Efficiency	0.90	Unitless
		Cost	2500.00	U.S. Dollars
		Mass	20.10	kg
	Medium AC Ind.	Maximum Power	20.00	kW
		Efficiency	0.90	Unitless
		Cost	2900.00	U.S. Dollars
		Mass	20.10	kg
	Large AC Ind.	Maximum Power	34.00	kW
		Efficiency	0.90	Unitless
		Cost	3200.00	U.S. Dollars
		Mass	20.10	kg
	PM DC	Maximum Power	14.10	kW
		Efficiency	0.85	Unitless
		Cost	1720.00	U.S. Dollars
		Mass	11.25	kg
Provision Elec. E.	Low Energy	Max. Energy	1000.00	kJ
		Efficiency	0.80	Unitless
		Cost	800.00	U.S. Dollars
		Mass	5.00	kg
	Medium Energy	Max. Energy	2000.00	kJ
		Efficiency	0.80	Unitless
		Cost	1600.00	U.S. Dollars
		Mass	10.00	kg
	High Energy	Max. Energy	4000.00	kJ
		Efficiency	0.80	Unitless
		Cost	3200.00	U.S. Dollars
		Mass	20.00	kg
Transfer Mech. E.	Suspension	Brake Power	120.00	kW
		Mass	40.00	kg
Distribute Mech. E.	Chassis	Mass	200.00	kg

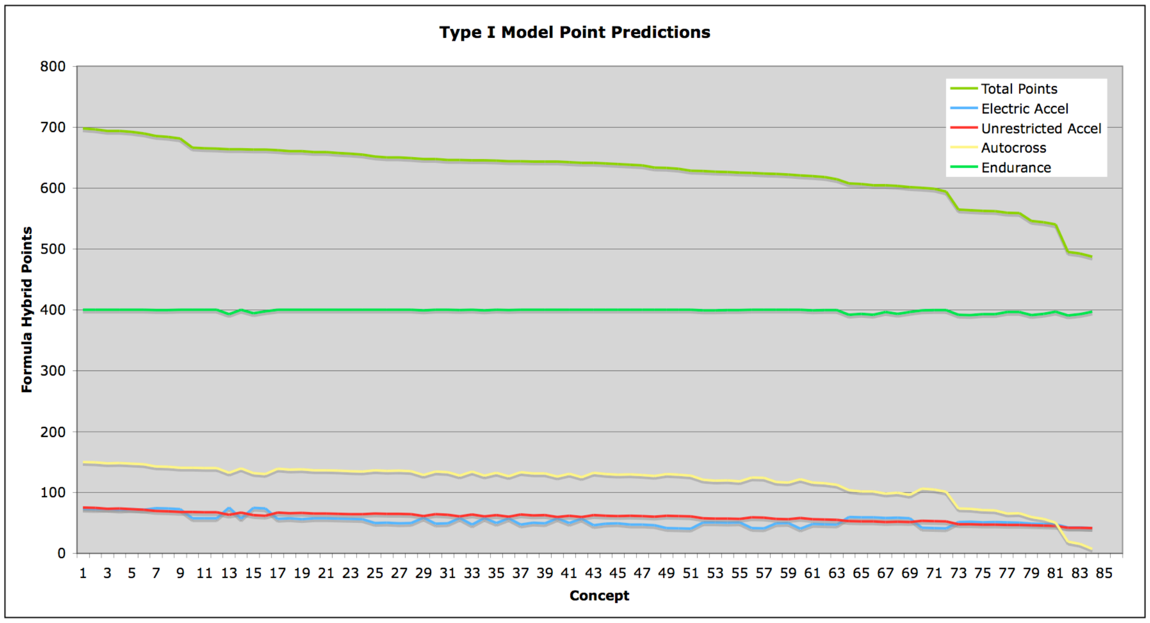


Fig. 6.4. Type I Model Point Scoring Predictions

From this Type I analysis, several predictive results were obtained. Due to the normalization factors, the endurance event, although worth the most points, provides a relatively small point spread between concepts. This results from the assumption that each concept will not fail during the event and will be operated at its maximum potential throughout the event. Although these assumptions are rarely seen in the actual results of the Formula Hybrid competition, they are appropriate within the context of an early powertrain analysis.

Under these assumptions, all concepts are predicted to complete the endurance event in a time significantly less than the maximum weighted time of one hour and will automatically receive a minimum of 300 points with the remainder weighted based on the one hour minimum time and the quickest time recorded from all of the concepts. The resulting points breakdown shows that as long as the powertrain provides enough propulsive energy to produce an endurance time significantly less than one hour, most of the points available in the event will be awarded. For the acceleration events, all concepts resulted in a relatively even number of points due to the weighting formula used in these events. Essentially, each concept finished the event in a time significantly less than 15s (the maximum weighted time) so it received most of the available points. Interestingly, the weighting of the autocross event (3.5 points for the slowest car and 150 for the fastest) provides the largest opportunity to separate the performance of a complete car from the other competition entries. Essentially, this event allows a 146.5 point swing from the fastest to slowest cars. Thus, it is highly important that this event be completed as quickly as possible.

The results of the analysis show that the best concepts are ones that include the highest performance internal combustion engines and electric motors with the lightest accumulators considered. Essentially, the performance gains from these choices overcome the addition of their extra weight (from a powertrain perspective). Although this seems like an obvious conclusion, without the modeling efforts it is not possible to know exactly how to trade off the extra weight for the additional performance.

At the completion of this analysis, it was determined that small motors were not worth considering for the remainder of the analysis. Additionally, the Sep. Ex. motors were removed as they did not provide any benefit over the AC motors. The PM DC motor was not ruled out due to its prevalence at the Formula Hybrid competition (many successful teams use this motor), low cost and availability. Performing this downselection of potential solutions at this point in the design process allows more resources to be focused on the concepts with the highest predicted performance later in the design process.

6.3.6. Sensitivity Analysis

As demonstrated in Section 4, a sensitivity analysis is a useful tool for understanding the important parameters and functions within a model. Knowing this information allows attention and resources to be focused on the most significant sources of variation in the performance of a system. To this end, a HyVar functionality-based sensitivity analysis was performed using the hybrid powertrain Type I models to obtain a break down of the sources of variation in the system's performance. This analysis was performed using the method shown in Section 4 and the results follow.

The sensitivity analysis was performed for three of the four dynamic events. The endurance sensitivity was not included in this example due to the significant increase in solution time over the endurance performance solution time. Since the relative performance of the various concepts did not have a large impact on the point score for the endurance event, this decision seemed appropriate. For the three remaining events (the accelerations and autocross), the dependant variable in the sensitivity analysis was selected to be the completion time of the event. For each sensitivity analysis for each concept, a full parametric sensitivity analysis was performed and the results grouped according the functional breakdown of the system. Pie charts illustrating the average relative contribution to the overall variation of performance for the autocross, electric only acceleration and unrestricted acceleration appear in Figs. 6.5, 6.6 and 6.7 respectively.

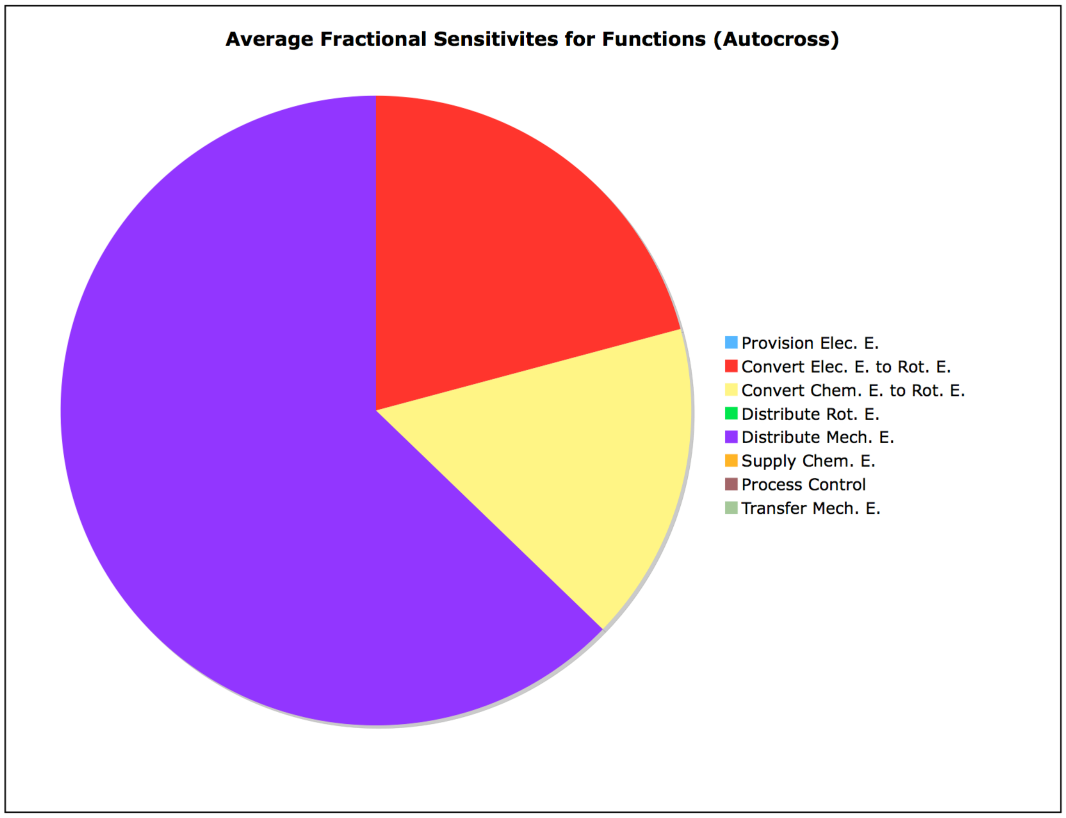


Fig. 6.5. Autocross Event Sensitivities

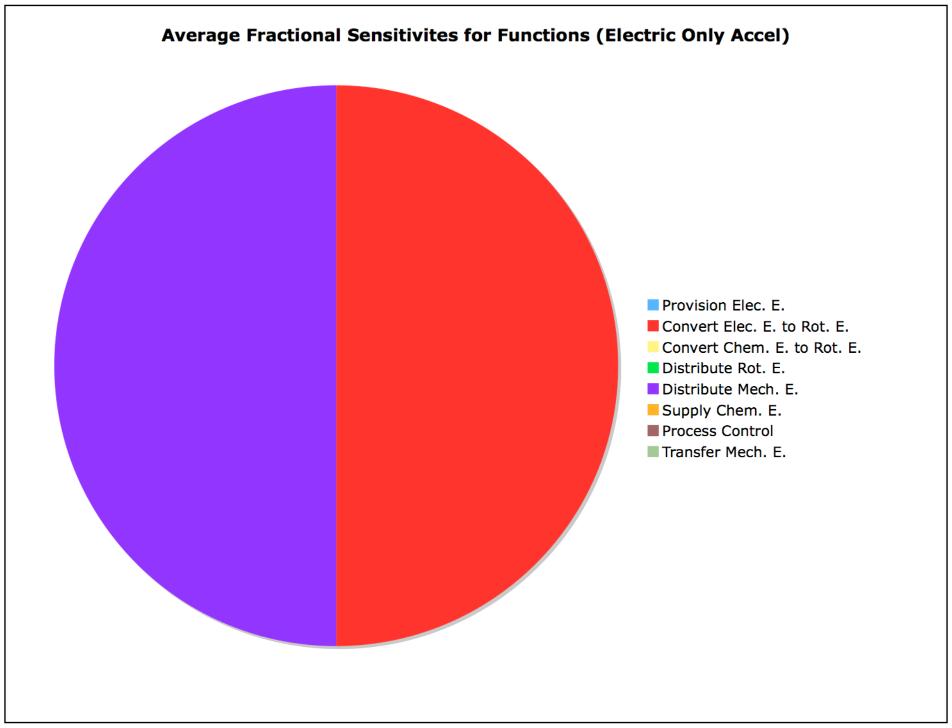


Fig. 6.6. Electric Only Acceleration Event Sensitivities

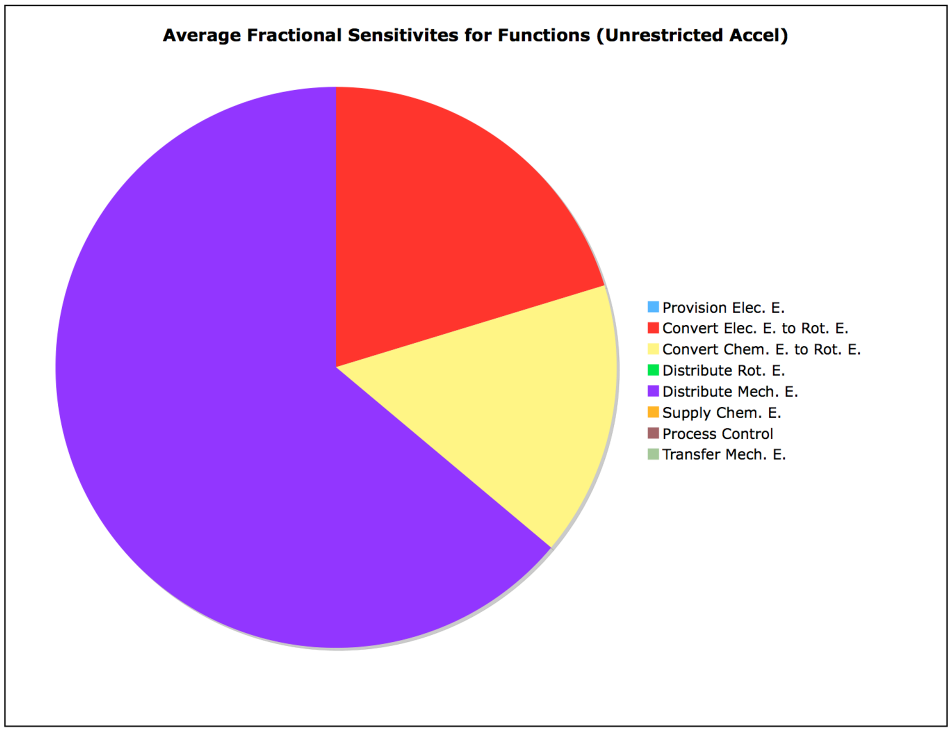


Fig. 6.7. Unrestricted Acceleration Event Sensitivities

As shown in Figs. 6.5 and 6.7, the autocross and unrestricted acceleration events have similar sensitivity profiles. The most significant contributions to variation in the performance of the system in these events is the *distribute mechanical energy* function that represents the functionality of the vehicle's chassis. The major parameter that contributes to this sensitivity is the mass of the vehicle. The next highest contributing function is the conversion of electrical energy to rotational energy (the motor) followed by the *convert chemical energy to rotational energy* function (the IC engine). These two functions are shown to be relatively equal in their parametric contributions to variation. For the electric only acceleration event, the *distribute mechanical energy* and *convert electrical energy to rotational energy* functions shared roughly the same contribution to the overall variation in the system's performance. In each of the analyses, the remaining functions contributed insignificantly to the variation of the system's performance within the context of the Type I analysis. However, the results do show that for the remainder of the design process, a relatively equal amount of attention and resources should be expended between the motor and engine modeling and selection process (one aspect should not be explicitly favored over the other).

6.3.7. Prescribing Targets

The results of the behavioral analysis show that the autocross event offers the most significant opportunity to gain points on other competition entries. In the analysis, the low and medium energy accumulators offer good performance but there are some effects that were not modeled that may show up in a more detailed analysis. The predicted times for the acceleration events are low (3.12s and 3.88s for the quickest unrestricted and electrical only events times respectively) compared to a winning time of around 5s for both events last year. Traction limits and the inability to operate the power devices at peak power levels throughout the events explain this result. As a result, the absolute values predicted from this analysis cannot be used to make confident predictions of the actual cars performance. However, the relative ranking of concepts provides an indication of which concepts have more performance potential than others. For example, the power/mass trade off for the motors and engines looks to favor power. A mass of around 315 kg and a cost of around $5100 is average among concepts. Slightly more mass, around 320kg and a cost around $6000 provides top-level performance.

6.4. DESIGN SYNTHESIS

The next step in the design process for the hybrid powertrain was to find appropriate solutions to the functions of the system and combine them into concepts. Next, the behavior of these concepts was investigated along with its sensitivity. This behavior was then used to perform another downselection and ultimately resulted in a set of feasible concepts to be considered for implementation in the racecar.

6.4.1. Finding and Combining Forms

In the Type I analysis, general information about component solutions to functionality was used to develop parameter sets for use in the model. The model itself, being targeted for a Type I analysis, was not developed with the specific performance characteristics of the available (or possible) component solutions to functionality. Now that a general set of feasible solution types is known, a more focused component solution identification process can be completed. Due to the requirement of using common and off-the-shelf components where possible, it was necessary to identify existing solutions for as many of the powertrain functions as possible. The *provision electrical energy*, *convert electrical energy to rotational energy* and *convert chemical energy to rotational energy* functions could all be solved with commercially available components. Through the course of the Texas A&M Formula Hybrid research and design courses, it was determined that commercially available lithium-ion battery packs for power tools provided the best combination of availability, cost and performance. To this end, these packs will provide the basis for selecting an appropriate accumulator to solve the *provision electrical energy* function. Specifically, combinations of DeWalt 36V power tool packs utilizing the A123 Systems brand li-ion cells were identified as the most appropriate technology to use. Additionally, 72 Volts was selected to be the nominal voltage of the accumulator due to the availability of motors in this range. Within the context of these design choices, there are several important factors to investigate. Specifically, the parameters include battery pack capacity (in amp*s), cost, and mass along with motor selection. Additionally, an appropriate internal combustion engine must be selected to complement the electric motor and provide suitable performance. Without a detailed model of these components and their contributions to system-level performance, it is not possible to make an informed decision as to the best set of solutions. As a result, a more detailed behavioral analysis of the system is required based on the new information available from making these design decisions. To make these models, the set of potential component solutions to the major functions of the hybrid powertrain system was limited to those appearing in Table 6.8. From this selection, it is now possible to make Type II models and explore the performance on the system in a higher fidelity.

Table 6.8. Identified Solutions

Function	**Solution**
Convert Chem. To Rot. E.	Honda GX240
	Yamaha WR250X
	Prototype Engine
Convert Elec. E. to Rot. E.	EM AC-1
	EM AC-2
	PMG 132-1
	PMG 132-2
Provision Elec. E.	4-36V Dewalt
	8-36V Dewalt
	16-36V Dewalt

The number of solutions identified for each powertrain function were selected so that an exhaustive search of the space generated by permuting these solutions was feasible. The total number of solution combinations generated was 36. To model each of these 36 solutions, the automated behavioral model assembly and solution method shown in Section 5 was used along with the functionality-derived behavioral modeling approach used in the Type I analysis.

Since component solution information is available, a Type II analysis could be performed. For this analysis, a better approximation of the complete vehicle dynamics was desired. To this end, the functionality of the vehicle's suspension, wheels and tires was decomposed from a single function into two functions (one for the front and one for the rear). This decomposition facilitates the development of a dynamic model that includes longitudinal weight transfer effects and allows the use of a high fidelity tire model. This tire model allows the affects of limited traction to be observed in the model and improves upon the assumptions made in the Type I analysis where potential traction was not modeled. This functional decomposition results in a new functional model for the system along with a new set of flows. For the Type II analysis developed for the hybrid powertrain system, the functional model and flow routing models shown in Figs. 6.8 and 6.9 were used.

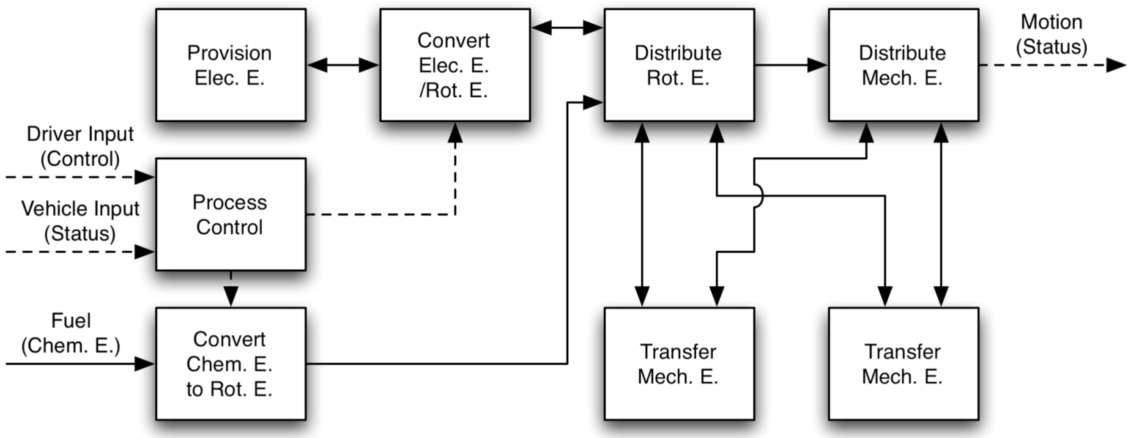


Fig. 6.8. Hybrid Powertrain Functional Model for Type II Analysis

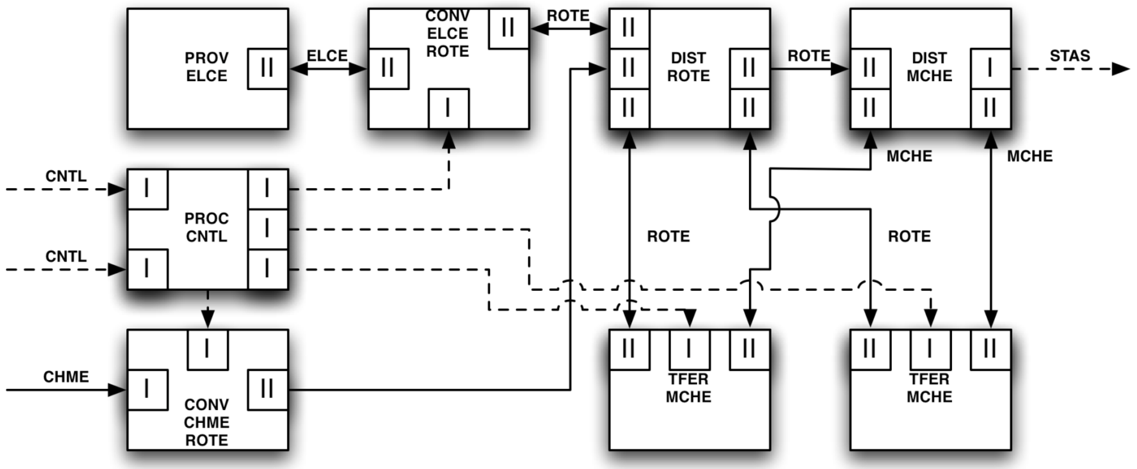


Fig. 6.9. Hybrid Flow Type/Routing Model for Type II Analysis

6.4.2. ASSESSING BEHAVIOR

The development of the Type II behavioral model elements for the hybrid powertrain system is shown in Section 3. Like the Type I analysis, a full parameterization for each solution to each function is required to use the model. For the solutions shown in Table 6.8, model parameters were identified. These parameter values appear in Table 6.9.

Table 6.9. Parameter Values for Type II Behavior Analysis

Function	Concept	Parameter	Value	Units
Convert Chem. To Rot. E.	Honda GX240	Low Speed Sample	209.00	rad/s
		Med. Speed Sample	283.00	rad/s
		High Speed Sample	377.00	rad/s
		Low Torque	16.50	N*m
		Med. Torque	17.00	N*m
		High Torque	15.50	N*m
		Max. Speed	375.00	rad/s
		Min. Speed	200.00	rad/s
		Cost	500.00	U.S. Dollars
		Mass	27.00	kg
	Yamaha WR250X	Low Speed Sample	420.00	rad/s
		Med. Speed Sample	838.00	rad/s
		High Speed Sample	1047.00	rad/s
		Low Torque	16.60	N*m
		Med. Torque	22.36	N*m
		High Torque	16.94	N*m
		Max. Speed	1100.00	rad/s
		Min. Speed	600.00	rad/s
		Cost	2000.00	U.S. Dollars
		Mass	45.00	kg
	Prototype Engine	Low Speed Sample	420.00	rad/s
		Med. Speed Sample	838.00	rad/s
		High Speed Sample	1047.00	rad/s
		Low Torque	16.60	N*m
		Med. Torque	31.00	N*m
		High Torque	20.00	N*m
		Max. Speed	1100.00	rad/s
		Min. Speed	600.00	rad/s
		Cost	2000.00	U.S. Dollars
		Mass	45.00	kg
Convert Elec. E. to Rot. E.	EM AC-1	Rated Voltage	84.00	V
		Rated Power	34.27	kW
		Max. Current	550.00	A
		Rated Torque	142.00	N*m
		Max. Speed	838.00	rad/s
		Mass	20.10	kg
		Cost	3200.00	U.S. Dollars
	EM AC-2	Rated Voltage	84.00	V
		Rated Power	18.60	kW
		Max. Current	300.00	A
		Rated Torque	108.00	N*m
		Max. Speed	838.00	rad/s
		Mass	20.10	kg
		Cost	2600.00	U.S. Dollars

Table 6.9 Continued.

Function	Concept	Parameter	Value	Units
Convert Elec. E. to Rot. E.	PMG 132-1	Torque Constant	0.19	N*m/A
		Speed Constant	5.24	(rad/s)/V
		Int. Resistance	0.03	Ohm
		Max. Current	400.00	A
		Max. Speed	350.00	rad/s
		Mass	11.25	kg
		Cost	1700.00	U.S. Dollars
	PMG 132-2	Torque Constant	0.19	N*m/A
		Speed Constant	5.24	(rad/s)/V
		Int. Resistance	0.03	Ohm
		Max. Current	200.00	A
		Max. Speed	350.00	rad/s
		Mass	11.25	kg
		Cost	1500.00	U.S. Dollars
Provision Elec. E.	4-36V Dewalt	Int. Resistance	0.10	Ohm
		Nominal Voltage	72.00	V
		Capacity	14000.00	Amp*s
		Cost	800.00	U.S. Dollars
		Mass	5.00	kg
	8-36V Dewalt	Int. Resistance	0.05	Ohm
		Nominal Voltage	72.00	V
		Capacity	28000.00	Amp*s
		Cost	1600.00	U.S. Dollars
		Mass	10.00	kg
	16-36V Dewalt	Int. Resistance	0.03	Ohm
		Nominal Voltage	72.00	V
		Capacity	56000.00	Amp*s
		Cost	3200.00	U.S. Dollars
		Mass	20.00	kg
Distribute Mech. E.	Chassis	CG X Location	0.77	m
		CG Z Location	0.05	m
		Wheelbase	1.70	m
		Mass	200.00	kg
Distribute Rot. E.	Torque-coupled DT	Max. Vehicle Speed	30.00	m/s
		Gears	5.00	Integer
Process Control	Controller	Throttle Gain	10.00	1/(m/s)
		TC Slip Start	0.15	Unitless
		TC Slip Stop	0.30	Unitless
Transfer Mech. E.	Front Suspension	Rot. Inertia	1.20	kg*m^2
		Tire Radius	0.25	m
		Peak Brake Torque	2000.00	N*m
		Mass	30.00	kg
Transfer Mech. E.	Rear Suspension	Rot. Inertia	1.20	kg*m^2
		Tire Radius	0.25	m
		Peak Brake Torque	2000.00	N*m
		Mass	30.00	kg

Using these parameter values, the performance of each of the 36 total concept permutations was predicted for the autocross, endurance and acceleration events. Like the Type I analysis, the Formula Hybrid score weighting system was used to rank these concepts by predicted point scoring capability.

6.4.3. Comparing Behaviors

The results of the Type II analysis were sorted based on predicted scoring capability and appear in Fig. 6.10. The general trends shown in this figure match the trends from the Type I analysis in that the autocross event represents the most significant source of total point variation in the competition. Essentially, all of the concepts provided enough raw performance to finish the endurance event in significantly less time that one hour and as a result received most of the points available in this event. Like the Type I analysis, the performance in the unrestricted acceleration event did not significantly vary due to the 15 second maximum weighted time. However, the electric acceleration event does show a significant variation between concepts and thus is shown to be a larger contribution to the relative performance of a concept than was determined in the lower-fidelity Type I analysis. The raw predicted times from this analysis appear in Appendix 5. Specific observations from the Type II analysis follow.

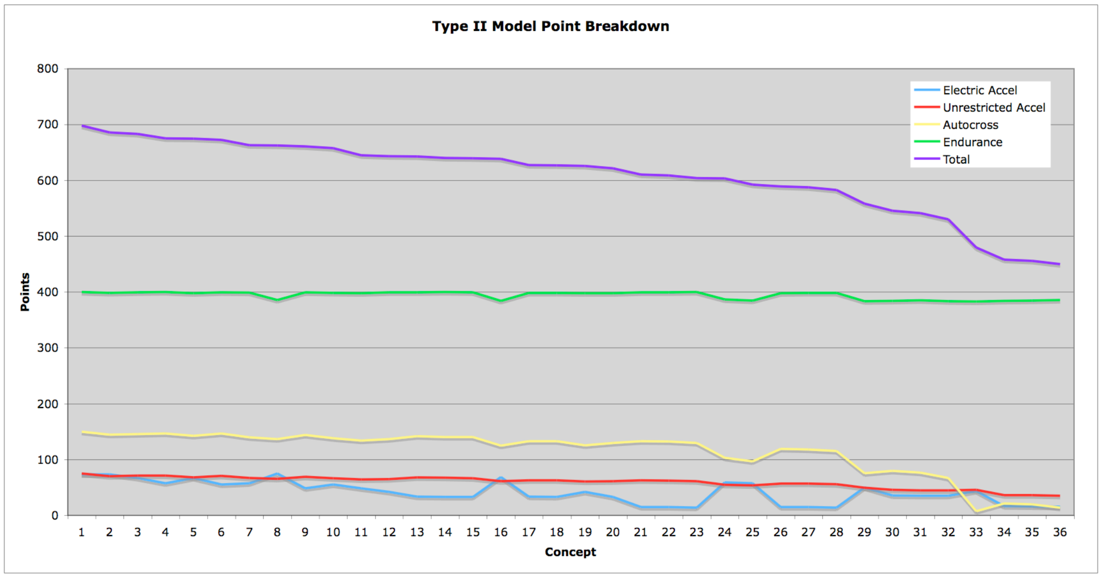


Fig. 6.10. Predicted Point Scoring from Type II Analysis

As with the Type I analysis, the best performance was achieved with large AC motors and as high of a performance engine as possible. However, unlike the Type I analysis, larger electrical accumulator sizes outperformed smaller ones for the concepts with large electrical motors. It is theorized that this result comes from the power losses incurred when large motors pull large amounts of current through a small accumulator (for a given type of battery and total accumulator voltage, the larger the capacity the lower the internal resistance). Since this effect was not modeled in the Type I analysis, it does not affect the results (this is shown in the zero percent contribution to sensitivity in the results of the Type I sensitivity analysis). However, the higher fidelity Type II analysis includes the resistance of batteries and its effect on the performance of the system. For example, the highest scoring concept completed the electric only acceleration in 5.0668s compared to the same concept with a smaller battery size completing the event in 7.1725s (both concepts did not significantly deplete the capacity of their batteries during the event). Thus, the low energy battery packs were removed from further consideration.

From the results, additional cuts were made from the set of feasible solutions. The EM AC-2 motor provides no advantages over the EM AC-1 except a slightly cheaper cost. However, the cost difference was not determined to merit its inclusion in further analyses. Likewise, the prototype engine offers significant performance over the WR250X and at the time of the analysis, engines of this kind were available. Thus, the WR250X was excluded from the set of solutions. Although the performance of the permanent magnet DC motors was significantly less than that of the AC motors, their availability and proven performance at competition was judged sufficient to retain them in the analysis. A concept with a small industrial engine and large accumulator provided a reasonably high score at 662 points but due to the need to develop a multi-ratio gearbox for this engine it was decided to exclude it from the analysis as well.

As with the Type I analysis, a sensitivity analysis was performed once infeasible performing solutions had been culled. The same sensitivity analysis technique was used to create a breakdown of the parametric sensitivities of the system's performance based on functional boundaries. Pie charts illustrating this break down for the autocross event for the highest scoring AC and DC concepts appear in Figs. 6.11 and 6.12. Unlike the

Type I sensitivity analysis, most of the functions in the system produced significant contributions to the overall performance of the concepts.

By using a functional grouping of the parametric sensitivities, it is possible to directly compare the impact of each powertrain function on the overall performance of the system between various configurations. As shown in Figs. 6.11 and 6.12, the significance of each function can vary greatly between different physical solutions. The sensitivity with respect to the functionality of the motor was roughly equal for the AC and DC concepts as was the sensitivity with respect to the internal combustion engine. However, the DC concepts proved to be much more sensitive to variation in the functions performed by the rear suspension/wheels/tire. This result at first seems counterintuitive, but results from the sensitivity of the system to the overall mechanical advantage between the motor and the tire/ground interface. AC motors, which operate primarily in a constant power regime, are much less sensitive than the DC motors used, which operate in a constant torque/current limited regime as implemented (with fixed ratio gearing). As a result, the DC motor is rarely at its peak power level. This effect is readily apparent in the significant increase in electric only acceleration time for the concept using DC motors as compared to those using AC motors.

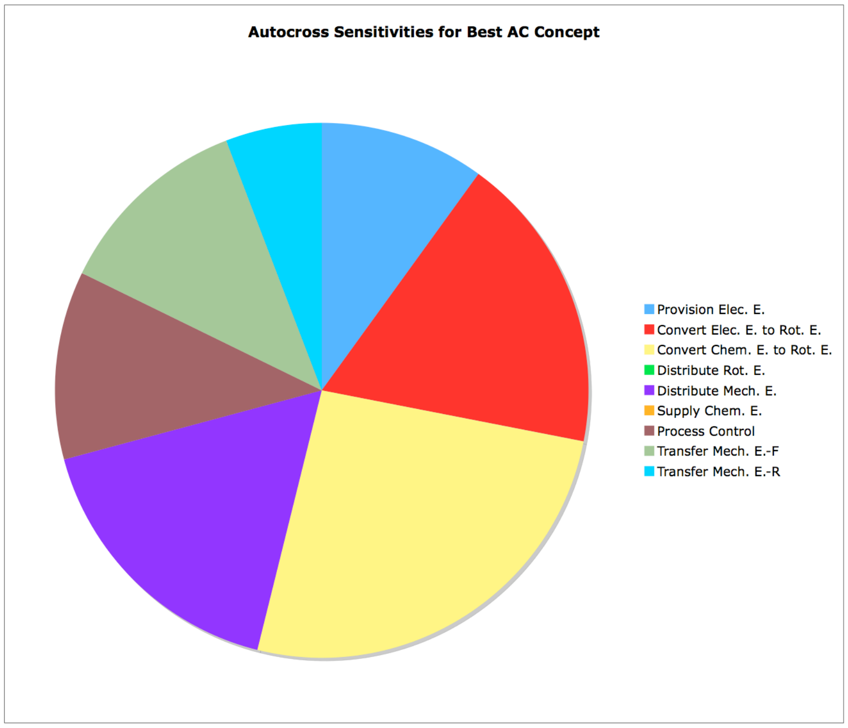


Fig. 6.11. Autocross Sensitivities for Best AC Concept

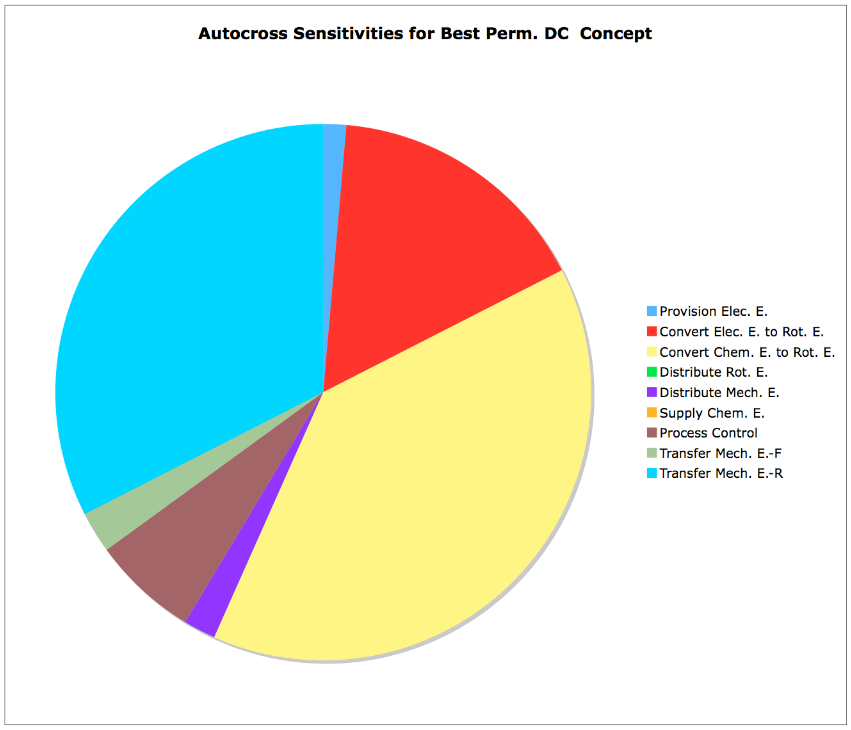


Fig. 6.12. Autocross Sensitivities for Best DC Concept

As seen in the difference between the sensitivity profiles resulting from the Type I analysis (Fig. 6.5) and the Type II analysis (Figs. 6.11 and 6.12), the contribution to sensitivity can change significantly between the two analysis types. The Type I autocross analysis breaks down the overall contribution to three functions: convert electrical energy to rotational energy, convert chemical energy to rotational energy and distribute mechanical energy. These functions represent the electric motor, IC engine and chassis respectively. The Type II analysis of the best AC concept (Fig. 6.11) shows that the motor and engine still contribute significantly to the overall sensitivity of the system but the contribution of the chassis is significantly less and the other elements of the system show significant contributions (unlike the Type I analysis). This results from the difference in the models used between the Type I and Type II analysis. In the Type I analysis, a lumped parameter model of the chassis was used. In the Type II analysis, the effects of the suspension were modeled and included in separate functions. Additionally, the increase in fidelity between the two analyses meant that more parameters and behaviors were included for each function. For the AC concept, this meant that the contribution of the chassis was reduced and replaced by elements that became more important once the fidelity of their models improved.

For the DC concept (Fig. 6.12), a different phenomenon is seen. In the Type II analysis, the transfer mechanical energy function representing the rear suspension and wheel/tire combination became one of the most significant contributions to sensitivity. This result seems out of line with the Type I analysis and the AC Type II analysis but has a simple explanation (as explained earlier). The inclusion of higher fidelity models in the Type II analysis allows this inconsistent, but explainable, behavior to be made apparent. Thus demonstrating the importance of performing a functional sensitivity contribution analysis for each model type and concept.

6.4.4. Selecting Concepts

From the results of the Type II analysis, four concepts were selected. The first selected concept utilizes the EM AC Motor Variant 2 (the larger of the two EM AC motors), a medium energy battery pack (8 DeWalt Packs) and the prototype engine. This concept offers exceptional performance (the third highest out of the entire considered set) with a significant cost decrease over the top performers. The second concept identified was the highest performing concept that utilizing the EM AC Variant 2 motor, 16 DeWelt packs and the prototype engine. This concept is the most expensive at $8400 but offers the best performance predicted through the analysis. Additionally, since concepts 1 and 2 are highly similar, either could be built and then adapted to meet the specifications of the other in subsequent testing. The final two concepts used the PMG 132 PM DC motor. These concepts offer good performance (around 640 points out of 700 in this analysis) but use a motor that is more available than the AC motor and has been tested in competition. These two concepts also offer significant cost and mass savings over the AC concepts. The primary advantage of the AC powered concepts is the reduced electrical only acceleration time. The predicted performance of these four concepts along with cost and mass appears in Table 6.10.

Table 6.10. Predicted Results for Selected Concepts

Concept	EE Accel	UR Accel	Autocross	Endurance	Points	Mass	Cost
1	5.39	4.36	51.06	1054.8	683	340	6800
2	5.07	4.18	50.92	1053.2	698	350	8400
3	8.21	4.54	51.23	1053.6	640	341	6900
4	8.10	4.51	51.19	1054.1	643	331	5300

6.4.5. Design Process Conclusions

A structured design method built around the use of functionality organized behavioral models and sensitivity analyses was performed for a hybrid powertrain system intended for use in a Formula Hybrid racecar. Feedback from the Texas A&M Formula Hybrid team was used throughout the process to assist in the selection of solutions and generation of customer needs. In this process, a large number of initial concepts was conceived and evaluated using a Type I behavioral and sensitivity analysis. The results of this analysis were then used to reduce the set of concepts to enable a more detailed analysis.

The reduced set of concepts was then used to perform a more focused search for solutions to conceptual functionality. The results of this search were then used to develop Type II behavioral models for the elements of the system. An exhaustive study of the behavior of the identified solutions was then performed along with another sensitivity analysis. The results of these analyses were then used to identify a set of feasible concepts for implementation within the racecar. The use of a structured design process and function-based design tools allowed consistent comparisons to be made between the results of the Type I and Type II analyses and provided a clear path for iteratively identifying, analyzing and selecting solutions. The analysis allowed a guided decision making process that translated a broadly scoped complex engineering design problem into a series of sequential actions that resulted in a narrowing of the available solution space to a small number of quantitatively evaluated viable solutions. Additionally, the method is structured in a manner than allows further analysis of these concepts to be performed using the same framework and basic iteration of activities.

7. CONCLUSION

During the design of a complex system, behavioral models are a necessity. Such systems exhibit behavior that is not capable of being analyzed with traditional abstraction-based modeling approaches. Component-based modeling approaches have been developed to allow the modeling of complex systems but fail to include a highly important aspect of modeling: a formal functional decomposition. Functional decompositions have been used throughout the design of complex systems in the form of schematics, block diagrams and flow charts but only recently has there been a push for developing a formal functional modeling approach.

Such a formal approach, including a modeling lexicon and a standardized modeling method, offers significant benefits over informal modeling practices. Namely, the ability to key critical design information to specific functions. This enables model re-use and shared knowledge between systems based on common functionality. Formal functional modeling also promotes the creation of a form-neutral representation of a system to support conceptual design enabling a formal representation of "what" a system must do independent of "how" it is to be done. In the context of a complex system's design, behavioral models may be used to answer both of these questions: What is the behavior I need in order to satisfy the system's needs and requirements? How can this behavior be realized with physically available solutions?

To support the answering of these questions and to integrate with recommended practices in formal functional modeling, a functionality-based behavioral modeling framework has been developed. The novelty in this approach is the use of a formal functional decomposition as the driving force for behavioral model decomposition, creation, assembly and solution. Additionally, a novel sensitivity analysis is proposed to augment the behavioral modeling framework. The implementation of these tools in a software framework and their use in an actual design problem demonstrate not only their completeness, but also their applicability.

The functionality-based behavioral modeling approach extends the concept of a component-based model to include a formal definition of function. This allows the behavioral modeling process for a system to be better integrated with the design process through the common use of functional decomposition. The formal inclusion of functionality as an abstraction above form solutions results in a modeling process that more closely resembles the object-oriented design process that has revolutionized the software development industry. Adopting the object-oriented design philosophy in the engineering design paradigm has long been prescribed. Earlier attempts at performing this reconciliation fail to recognize the necessity of decoupling functionality (what) from form (how). The work presented here extends existing philosophies to a functional abstraction and integrates the behavioral modeling process into a rigorous design method. The design method itself was carefully researched and reconciled from both the Design Theory and Systems Engineering philosophies.

The functional decomposition of behavioral modeling also supports the concept of a function-based sensitivity analysis. However, existing sensitivity analysis approaches are not suited to performing this analysis in early design. Local derivative-based approaches do not provide the measures necessary for functional association of sensitivity contribution and global approaches require too much information and have too high a computational burden to be used in early design. The HyVar method presented in this work allows a local contributive sensitivity analysis of many concepts during early design that allows a smooth transition to a full global analysis in the later stages of design. This transition is enabled through the use of a common approach (variation) and similar measures (contribution parameters). Additionally, the HyVar approach allows the calculation of sensitivity ratios that provide insight into the relative contribution of functions to overall sensitivity in a manner that is amenable to formal optimization as well as being comprehensible to a designer.

Both the behavioral modeling framework and the HyVar sensitivity analysis have been sufficiently developed to enable the creation of a computational implementation. This implementation has been used to solve a number of design problems and utilizes strong object-oriented design principles to facilitate its integration with the developed design tools and support future extensions of the work.

The methods, implemented within the computational framework, are presented along with a comprehensive design example based on a Formula Hybrid racecar powertrain. This example illustrates the process of defining and decomposing a difficult design problem using the concepts developed in this work. The tools and methods develop here are used to solve this design problem and provide a set of feasible concepts for future investigation and potential use in Formula Hybrid racecars. The result of this analysis would not be possible without the design tools and implementation developed in the course of this work.

Through the presentation of this work and the results presented in the various examples, it should be apparent that a formal functional decomposition represents a significant opportunity to improve current behavioral modeling practices and allows such modeling to better integrate within the context of a modern complex design problem. It is the author's sincerest hope that readers, if presented with a difficult modeling problem in the future, will attempt the functional decomposition and model development method suggested here.

REFERENCES

[1] Sage, A., and Armstrong, J., 2000, *Introduction to Systems Engineering*, Wiley-Interscience, New York.

[2] Blanchard, B. S., and Fabrycky, W. J., 2006, *Systems Engineering and Analysis,* Prentice-Hall, Upper Saddle River, NJ.

[3] Ulrich, K. T., and Eppinger, S. D., 2004, *Product Design and Development*, McGraw-Hill/Irwin, Boston, MA.

[4] Pahl, G., Beitz, W., Feldhusen, J., and Grote, K. H., 2007, *Engineering Design: A Systematic Approach*, Springer Verlag, London, UK.

[5] Ullman, D. G., 2002, *The Mechanical Design Process 3rd Edition*, McGraw-Hill, Inc., New York.

[6] Otto, K., and Wood, K., 2001*, Product Design: Techniques in Reverse Engineering, Systematic Design, and New Product Development*, Prentice-Hall, New York.

[7] The Modelica Association, 2009, Modelica and the Modelica Association, http://www.modelica.org.

[8] The Mathworks, Inc., 2008, The Mathworks Simulink - Simulation and Model-Based Design, http://www.mathworks.com/products/simulink.

[9] Controllab Products, 2008, 20-Sim the Power in Modeling, http://www.20sim.com.

[10] Hirtz, J., Stone, R., Mcadams, D., Szykman, S., and Wood, K., 2002, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," Research in Engineering Design, **13**(2), pp. 65-82.

[11] Strousrup, B., 1997, *The C++ Programming Language*, Addison-Wesley, Reading, MA.

[12] Bryant, C. R., Bohm, M. R., Stone, R. B., and McAdams, D. A., 2007, "An Interactive Morphological Matrix Computational Design Tool: A Hybrid of Two Methods," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2007*, Las Vegas, NV, DETC2007-35583.

[13] Kurtoglu, T., and Tumer, I. Y., 2007, "A Graph-Based Framework for Early Assessment of Functional Failures in Complex Systems*," Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2007*, Las Vegas, NV, IDETC2007-35421.

[14] Bracewell, R. H., and Sharpe, J. E. E., 1996, "Function Descriptions Used in Computer Support for Qualitative Scheme Generation – Schemebuilder," AI EDAM Journal – Special Issue: Representing Functionality in Design, **10**, pp. 333-346.

[15] Wirth, N., 2006, "Good Ideas, through the Looking Glass," Computer, **0018-9162**(06), pp. 28-39.

[16] SysML Partners, 2007, Open Source Specification Project - SysML, http://www.sysml.org.

[17] Karnopp, D. C., Margolis, D.L and Rosenberg R. C., 2000, *System Dynamics*, Wiley-Interscience, New York.

[18] Husain, I., 2003, *Electric and Hybrid Vehicles*, *Design Fundamentals*, CRC Press, Boca Raton, FL.

[19] Ehsani, M., and More, 2005, *Modern Electric, Hybrid Electric, and Fuel Cell Vehicles*, CRC Press, Boca Raton, FL.

[20] Chan, C. C., and Chau, K. T., 2001, *Modern Electric Vehicle Technology*, Oxford Science Publication, New York.

[21] Cross, N., 2000, *Engineering Design Methods: Strategies for Product Design*, John Wiley & Sons, LTD, Chichester, UK.

[22] Suh, N., 2001, *Axiomatic Design: Advances and Applications*, Oxford University Press, New York.

[23] Pugh, S., 1990, Total Design, Addison-Wesley, New York.

[24] Merriam-Webster, 2009, Dictionary and Thesaurus - Merriam-Webster Online, http://www.merriam-webster.com.

[25] Miles, L., 1961, *Techniques of Value Analysis and Engineering*, McGraw-Hill, New York.

[26] Rodenacker, W., 1971, *Methodisches Konstruieren*, Springer, Berlin, Heidelberg, New York.

[27] Gietka, P., and Verma, M., 2002, "Functional Modeling, Reverse Engineering, and Design Reuse," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2002*, Montreal, CA, IDETC2002-34019.

[28] Nagel, R. L., Hutcheson, R. S., Stone, R., Mcadams, D., and Donndelinger, J., 2008, "Function Design Framework (FDF): Integrated Process and Function Modeling for Complex System Design," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2008*, Brooklyn, NY, IDETC2008- 49369.

[29] Little, A., Wood, K., and Mcadams, D., 1997, "Functional Analysis: A Fundamental Empirical Study for Reverse Engineering, Benchmarking and Redesign," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 1997,* Sacramento, CA, IDETC-3879.

[30] Kurtoglu, T., and Campbell, M., 2009, "Automated Synthesis of Electromechanical Design Configurations from Empirical Analysis of Function to Form Mapping," ASME J. Mech. Des., **20**(1), pp. 83-104.

[31] Cross, M., and Moscardini, A. O., 1985, *Learning the Art of Mathematical Modeling*, Wiley and Sons, New York.

[32] Fowkes, N. D., and Mahony, J. J., 1994*, An Introduction to Mathematical Modeling*, John Wiley, New York.

[33] Giordano, F., 1985, *An Introductory Course in Mathematical Modelling*, Brooks Cole, Pacific Grove, CA.

[34] DynaSim A.B., 2008, Dymola Multi-Engineering Modeling and Simulation, http://www.dynasim.se.

[35] Elmqvist, H., 1978, "A Structured Model Language for Large Continuous Systems," Lund Institute of Technology, Lund, Sweden.

[36] Paynter, H. M., 1961, *Analysis and Design of Engineering Systems*, MIT Press, Cambridge, MA.

[37] Bhatta, S., and Goel, A., 1994, "Innovations in Analogical Design: A Model-Based Approach," *Proc. AI in Design 1994*, Lausanne, Switzerland.

[38] Cellier, F., Elmqvist, H., and Ottter, M., 1995, *The Control Handbook*, CRC Press, Boca Raton, FL.

[39] Jobe, J. M., Johnson, J. A., and Paredis, C. J. J., 2008, "Multi-Aspect Component Models: A Framework for Model Reuse in SysML," Proc. ASME *International Design Engineer Technical Conferences and Computers and Information in Engineering Conference* 2008, Brooklyn, NY, IDETC-49339.

[40] Hutcheson, R., Mcadams, D., Stone, R., and Tumer, I., 2007, "Function-Based Behavioral Modeling," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2007*, Las Vegas, NV, IDETC-35337.

[41] Lawrence Livermore National Laboratories, 2008, Sundials - Suite of Nonlinear and Differential/Algebraic Solvers, https://computation.llnl.gov/casc/sundials/main.html.

[42] Dartmouth College, 2008, Formula Hybrid International Competition, http://formula-hybrid.org.

[43] SAE International, 2007, SAE International Formula Series, http://students.sae.org/competitions/formulaseries.

[44] Ascher, U., and Petzold, L., 1998, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*, SIAM, Philadelphia, PA.

[45] Bohm, M., Stone, R., and Szykman, S., 2005, "Enhancing Virtual Product Representations for Advanced Design Repository Systems," J. Comp. Inf. Sci, in Eng., **5**(4), pp. 360-372.

[46] Saltelli, A. E. A., 2004, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models,* Wiley and Sons, Hoboken, NJ.

[47] Saltelli, A. E. A., 2008, *Global Sensitivity Analysis. The Primer*, Wiley and Sons, Hoboken, NJ.

[48] Chen, W., Jin, R., and Sudjianto, A., 2005, "Analytical Variance-Based Global Sensitivity Analysis in Simulation-Based Design under Uncertainty," ASME J. Mech. Des., **127**, pp. 875-886.

[49] Leamer, E., 1983, "Let's Take the Con Out of Econometrics," The American Econ. Rev., **73**(1), pp. 31-43.

[50] Larocque, G. R. E. A., 2006, "The Importance of Uncertainty and Sensitivity Analyses in Process-Based Models of Carbon and Nitrogen Cycling in Terrestrial

Ecosystems with Particular Emphasis on Forest Ecosystems," *Proc. International Society for Ecological Modelling (ISEM 2006)*, Burlington, VT.

[51] Martin, J., and Simpson, T., 2005, "A Methodology to Manage Uncertainty During System-Level Conceptual Design," *Proc. ASME International Design Engineer Technical Conferences and Computers and Information in Engineering Conference 2005*, Long Beach, CA, IDETC-84984.

[52] Du, X., Sudjianto, A., and Chen, W., 2004, "An Integrated Framework for Optimization under Uncertainty Using Inverse Reliability Strategy," ASME J. Mech. Des., **126**, pp. 562-570.

[53] The Open Source Modeling Consortium, 2009, The Openmodelica Project, http://www.ida.liu.se/labs/pelab/modelica/OpenModelica.html.

[54] Dedasys, L.L.C., 2009, Programming Language Popularity, http://www.langpop.com.

[55] Prechelt, L., 2002, "Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++ and Java," Adv. in Comp., **57**, pp. 207-271.

[56] Chapra, S., and Canale, R., 2006, *Numerical Methods for Engineers*, McGraw-Hill Higher Education, Boston, MA.

[57] Pantelides, C. C., 1988, "The Consistent Initialization of Differential-Algebraic Systems," SIAM J. Sci. and Stat. Comp., **9**, pp. 213-231.

APPENDIX 1. PROJECT PROPOSAL EXAMPLE

2009 Texas A&M Formula Hybrid Product Proposal
Prepared by: Ryan S. Hutcheson
Date: 6/9/2008

PROJECT OBJECTIVE

The objective of this project is to develop an entry for the 2009 Formula Hybrid competition. Formula Hybrid is an extension of the Formula SAE competition and involves the design, manufacture, testing and racing of an open-wheeled, open-cockpit (formula-style) racecar. The specific aspect covered by this proposal is the early design phase of the product including product planning and conceptual design.

MARKET

Texas A&M's 2009 Formula Hybrid team is the market for this product. In past years, Texas A&M has entered in the Formula SAE competition with great success including three first place finishes. For 2009, TAMU will switch to competing in Formula Hybrid rather than Formula SAE. Continuing TAMU's winning history is a critical result of this project.

Formula Hybrid is a spin-off series from Formula SAE and utilizes most of the competition and entry rules. The primary difference between the series is the requirement of an electric-internal combustion engine hybrid powertrain in Formula Hybrid. The specific competition events are somewhat different as well. Formula Hybrid (FH) allows the re-use of existing Formula SAE cars but an early analysis of prior FH competitors revealed that the increase in weight and changes in drivetrain configuration resulting from the addition of hybrid components necessitates the design of a new car.

BUDGET

The proposed budget for the construction of the car is $25,000. Unlike Formula SAE, Formula Hybrid includes no cost scoring event. The inclusion of a budget into the design process is purely from a production feasibility standpoint rather than a performance standpoint.

DESIRED PRODUCT FUNCTIONALITY

The ultimate result of the early design process will be a concept that is to be further developed into a complete design that will then be manufactured, tested and raced. The overall functionality of the product can be simply summarized: win the 2009 Formula Hybrid competition.

DETAILED OBJECTIVES

- From the Formula Hybrid and Formula SAE rules as well as the goals of the team, a set of high-level product objectives can be formulated for the early design process of the car. These objectives are itemized in the following list.
- Produce a design that can be practically implemented by the team
- Produce a design that is capable of winning the 2009 competition
- Stay within the allocated budget
- Produce a design that is capable of high performance (FSAE)
- Produce an aesthetically pleasing design (FSAE)
- Produce a comfortable design (FSAE)
- Use as many common parts as possible (FSAE)
- Use good engineering practices (FSAE)
- Create an efficient design (FH)
- Produce a design that is easy to maintain (FSAE)
- Produce a safe design (FSAE)
- Produce a design that accommodates from 5th percentile female to 95th percentile male (FSAE) drivers

REQUIREMENTS

The overall functionality of the finished product and the objectives for the early design process have been formulated as a set of engineering requirements. The "shall" language for representing these requirements along with a standard numbering system will be used for the remainder of the project (Table A1.1).

Table A1.1 - Product Requirements

Req. #	The	Object	Level	Shall	Requirement	Reference
S1	The	car	system	shall	be manufacturable by the target team	Objective 1
S2	The	car	system	shall	be capable of winning the 2009 FH competition	Objective 2
S3	The	car	system	shall	be aesthetically pleasing	Objective 5
S4	The	car	system	shall	have an estimated cost of less than XXXX	Objective 3
S5	The	car	system	shall	exhibit a high level of performance	Objective 4
S6	The	car	system	shall	be comfortable to its drivers	Objective 6
S7	The	car	system	shall	utilize good engineering practices	Objective 8
S8	The	car	system	shall	be energy efficient	Objective 9
S9	The	car	system	shall	be easy to maintain	Objective 10
S10	The	car	system	shall	be safe	Objective 11
S11	The	car	system	shall	fit drivers from the 5th % female to the 95th % male	Objective 12

CONCLUSIONS

To complete the design process for the Formula Hybrid entry, a structured design process reconciled from various engineering design texts will be used. This process will include a detailed analysis of the tasks to be performed along with the use of state-of-the-art function-based design tools. The objective of this design process is to identify and select a concept that meets the stated requirements and is ready for the detailed and embodiment stages of product design.

APPENDIX 2. COMPLETE LIST OF CAR REQUIREMENTS

Table A2.1. Car System Requirements

The	Object	Level	Shall	Requirement	Reference
The	car	system	shall	cost less than $25,000	1.2
The	car	system	shall	fit a 95th percentile male and a 5th percentile female	1.2
The	car	system	shall	be aesthetically pleasing	1.2
The	car	system	shall	exhibit a high performance	1.2
The	car	system	shall	be comfortable	1.2
The	car	system	shall	be designed and fabricated in accordance with good engineering practices	1.3
The	car	system	shall	be conceived, designed, fabricated and maintained by the student team	2.2.1
The	car	system	shall	be open-wheeled	3.1.1
The	car	system	shall	be open-cockpit	3.1.1
The	car	system	shall	have "no openings through the bodywork into the driver compartment" other than the cockpit opening	3.1.2
The	car	system	shall	have a wheelbase of at least 1525mm	3.1.2
The	car	system	shall	have four wheels	3.1.2
The	car	system	shall	have all four wheels not in a straight line	3.1.2
The	car	system	shall	have a smaller track that is no less than 75% of the larger track	3.1.3
The	car	system	shall	have all items on inspection form visible of capable of being made visible	3.1.4
The	car	system	shall	be equipped with a fully operating suspension sub-system	3.2.1
The	car	system	shall	not contact the ground during events (except for tires)	3.2.2
The	car	system	shall	have wheels greater than 203.2mm in diameter	3.2.3.1
The	car	system	shall	use any size or type of tire, slick or treaded as its dry tire	3.2.3.2
The	car	system	shall	use any size or type of treaded or grooved tire as its rain tire	3.2.3.2
The	car	system	shall	have a steering sub-system that effects at least two wheels	3.2.4
The	car	system	shall	have a braking sub-system	3.2.5
The	car	system	shall	have no non-crushable objects extend forward of the front bulkhead	3.3.6.5
The	car	system	shall	provide adequate visibility for the front and the sides of the car	3.4.3.1
The	car	system	shall	provide a minimum field of vision of 200 deg (100 deg to either side)	3.4.3.1
The	car	system	shall	utilize a head restraint	3.4.4
The	car	system	shall	have a floor closeout component	3.4.5
The	car	system	shall	have a steering wheel assembly	3.4.6
The	car	system	shall	allow the driver to exit the side of the vehicle in no more than five seconds	3.4.7
The	car	system	shall	have a track and CG that combine to produce adequate rollover protection	3.4.8
The	car	system	shall	must not roll when tilted at angle of 60 deg to the horizontal (either direction) with the tallest driver	3.4.8.1
The	car	system	shall	have a master switch assembly	3.4.9
The	car	system	shall	have a firewall assembly	3.4.10.1
The	car	system	shall	have a driver's leg protection assembly	3.4.14
The	car	system	shall	prevent the leaking of fuel or any other fluids when tiled to at least 45 degree when full of fluids	3.5.3.6.1
The	car	system	shall	not exceed 110 dBA, fast-weighting in the noise test	3.5.5.3
The	car	system	shall	be numbered in three locations as per section 3.6.1	3.6.1
The	car	system	shall	include the school's name as per 3.6.2	3.6.2
The	car	system	shall	include the SAE logo as per 3.6.3	3.6.3
The	car	system	shall	have room for technical inspection stickers on its nose as per 3.6.4	3.6.4
The	car	system	shall	have all aerodynamic or ground effect devices satisfy 3.7.1	3.7.1
The	car	system	shall	use fasteners that satisfy 3.7.2	3.7.2
The	car	system	shall	have any compressed gas cylinders and lines meet 3.7.4	3.7.4
The	car	system	shall	have any high pressure hydraulic pumps and lines meet 3.7.5	3.7.5
The	car	system	shall	use a transponder as per 3.8	3.8
The	car	system	shall	include the IEEE logo as per 3.6.3FH	3.6.3FH
The	car	system	shall	be capable of completing a 75m acceleration run in electric-only mode in less than 15s	FH-1.2
The	car	system	shall	not operate in wet conditions unless Rain Certified per FH-2.1.3	FH-2.1.3
The	car	system	shall	have no HV connections exposed as per FH-2.2	FH-2.2
The	car	system	shall	have no HV connections behind the instrument panel or on any cockpit switches or control panels	FH-2.2
The	car	system	shall	have all controls, indicators and data acquisition connections isolated using optical isolators, transformers or equivalent	FH-2.2

Table A2.1 Continued.

The	Object	Level	Shall	Requirement	Reference
The	car	system	shall	have all HV systems insulated and wired per FH-2.3	FH-2.3
The	car	system	shall	have all HV systems properly fused as per FH-2.4	FH-2.4
The	car	system	shall	have a warning strobe that meets FH-2.8 on the highest point of the roll bar to indicated when the vehicle is energized (HV outside the accumulator)	FH-2.8
The	car	system	shall	have LV systems ground to the frame	FH-2.9
The	car	system	shall	have LV systems protected by proper fuse if not current limited	FH-2.9
The	car	system	shall	have all charging systems maintained in safe working condition	FH-2.10

Table A2.2. Powertrain Requirements

The	Object	Level	Shall	Requirement	Reference
The	air intake	assembly	shall	lie within the surface defined by the top of the main roll hoop and the outside edge of the four tires	3.5.3.9
The	air intake	assembly	shall	have any portion lower than 350mm above ground shielded by structure that meets 3.3.8	3.5.3.9
The	cooling	sub-system	shall	use only plain water or plain water plus a rust and corrosion inhibitor at no more than 0.015L/L of water	3.5.1.6
The	drivetrain	sub-system	shall	be sealed to prevent leakage	3.5.1.5
The	drivetrain	sub-system	shall	have separate catch can components for the cooling and engine oil systems	3.5.1.5
The	electrical propulsion	sub-system	shall	have a maximum voltage of 600V DC or AC RMS under all circumstances	FH-2.1
The	electrical propulsion	sub-system	shall	not have any connections between the frame and any part of HV circuits (HV defined in FH-1.1)	FH-2.1
The	electrical propulsion	sub-system	shall	have all HV components isolated from LV circuits following FH-2.1	FH-2.1
The	electrical propulsion	sub-system	shall	have an on-board ground fault detector that meets FH-2.1.1	FH-2.1
The	electrical propulsion	sub-system	shall	be capable of passing the FH-2.1.2 ground fault check	FH-2.1.2
The	engine	sub-system	shall	be internal combustion, four-stroke, with a maximum displacement of 250cc	3.5.1.1FH
The	engine	sub-system	shall	be of modified or custom fabricated type and follow section 3.5.4FH or:	3.5.1.1.1FH
The	engine	sub-system	shall	be of stock type (section 3.5.1.1.2FH)	3.5.1.1.2FH
The	engine	sub-system	shall	use 93 octane pump, E-85 or Biodiesel	3.5.2FH
The	fuel	sub-system	shall	lie within the surface defined by the top of the main roll hoop and the outside edge of the four tires	3.5.3.9
The	fuel	sub-system	shall	not alter the temperature of the fuel to improve fuel economy	3.5.2.1
The	fuel	sub-system	shall	not use any agents other than fuel and air	3.5.2.2
The	fuel	sub-system	shall	be capable of being filled without manipulating the tank	3.5.3.4
The	fuel	sub-system	shall	be designed to prevent spillage from contacting the driver position, exhaust sub-system, hot engine parts or the ignition sub-system	3.5.3.5
The	fuel	sub-system	shall	not allow fuel to accumulate on belly pans	3.5.3.5
The	fuel	sub-system	shall	be vented in a manner than prevents fuel from spilling during hard cornering or acceleration	3.5.3.6
The	fuel	sub-system	shall	include a vent check valve to prevent fuel leakage when the tank is inverted	3.5.3.6
The	fuel	sub-system	shall	must vent outside of the body work	3.5.3.6
The	fuel	sub-system	shall	not use plastic lines	3.5.3.7
The	fuel	sub-system	shall	use bulb or barb fittings to retain rubber lines that use hose clamps	3.5.3.7
The	fuel	sub-system	shall	use hose clamps specifically designed for fuel systems	3.5.3.7
The	fuel	sub-system	shall	have its lines securely attached to the vehicle and/or engine	3.5.3.7
The	fuel	sub-system	shall	use lines that are shielded from possible rotating equipment failure or collision damage	3.5.3.7
The	fuel injection	assembly	shall	use metal braided with crimped on or re-usable threaded fittings for flexible lines or:	3.5.3.8

Table A2.2 Continued.

The	Object	Level	Shall	Requirement	Reference
The	fuel injection	assembly	shall	use reinforced rubber hose with an abrasion resistant protection for flexible lines	3.5.3.8
The	fuel injection	assembly	shall	not use hose clamps on braided lines	3.5.3.8
The	fuel injection	assembly	shall	have any fuel rails securely attached to the engine block, cylinder head, intake manifold with brackets and mechanical fasteners	3.5.3.8
The	fuel injection	assembly	shall	have its intake manifold securely attached to the engine block or cylinder head with brackets and mechanical fasteners	3.5.3.8
The	powertrain	sub-system	shall	have drivetrain guard components	3.5.1.4
The	powertrain	sub-system	shall	be equipped with a muffler to reduce noise levels	3.5.5.1
The	catch can	components	shall	have a minimum volume of ten percent of the fluid being contained or 0.9L	3.5.1.5
The	catch can	components	shall	be capable of containing boiling water without deformation	3.5.1.5
The	catch can	components	shall	be located rearwards of the firewall below driver's shoulder level	3.5.1.5
The	catch can	components	shall	have a vent with a minimum diameter of 3mm	3.5.1.5
The	catch can	components	shall	have a vent that points away from the driver	3.5.1.5
The	catch can	components	shall	have their vent lines routed to the intake system upstream of the restrictor (if routed to intake)	3.5.1.5
The	drivetrain guard	components	shall	be at least 2.66mm steel and have a width at least three times the width of the chain (chain systems)	3.5.1.4
The	drivetrain guard	components	shall	be at least 3.0mm aluminum and a minimum width that is equal to the width of the belt plus 35% on each side (1.7 times the belt width) (belt systems)	3.5.1.4
The	drivetrain guard	components	shall	be attached with a minimum of 6mm grade M8.8 hardware	3.5.1.4
The	drivetrain guard	components	shall	mounted so they remain laterally aligned with the chain or belt under all conditions	3.5.1.4
The	battery	component	shall	be securely attached to the frame of the car	3.4.11
The	battery	component	shall	be enclosed in a nonconductive marine-type container if of wet-cell design	3.4.11
The	battery	component	shall	have its hot terminal insulated	3.4.11
The	drain fitting	component	shall	be at the lowest point on the fuel tank	3.5.3.1FH
The	drain fitting	component	shall	be used to drain the tank	3.5.3.1FH
The	drain fitting	component	shall	be accessible from under the vehicle	3.5.3.1FH
The	drain fitting	component	shall	not protrude below the lowest plane of the vehicle frame	3.5.3.1FH
The	drain fitting	component	shall	have a provision for safety wiring	3.5.3.1FH
The	filler cap	component	shall	have a provision for a seal component	3.5.3.2FH
The	filler cap seal	component	shall	prevent the filler cap from being removed and should met 3.5.3.2FH	3.5.3.2FH
The	accumulator	assembly	shall	be securely attached to the frame of the car	3.3FH
The	accumulator	assembly	shall	shall meet the type and size rules listed in FH-2.5	FH-2.5
The	accumulator	assembly	shall	have a maximum standardized cost of $6000	FH-2.5
The	accumulator	assembly	shall	be of battery or capacitor type	FH-2.5
The	accumulator	assembly	shall	be in a closed container	FH-2.6
The	accumulator	assembly	shall	be isolated with normally open relays	FH-2.6
The	accumulator	assembly	shall	contain an appropriately rated fuse or circuit breaker	FH-2.6
The	accumulator	assembly	shall	not use contactors or realys containing mercury	FH-2.6
The	accumulator	assembly	shall	meet the voltage decay, probe provision and indicator rules in FH-2.6	FH-2.6
The	accumulator	assembly	shall	be of sturdy construction and electrically insulating, fireproof and transparent to meet FH-2.7	FH-2.7
The	accumulator	assembly	shall	be labeled with high voltage signs that meet FH-2.7	FH-2.7
The	accumulator	assembly	shall	have an active vent if H2 gas can be released	FH-2.7
The	exhaust	assembly	shall	be routed in a manner that prevents the drivers from being subjected to fumes at any vehicle speed	3.5.5.2
The	exhaust	assembly	shall	have its outlet not more than 60cm behind the centerline of the rear axle	3.5.5.2

Table A2.2 Continued.

The	Object	Level	Shall	Requirement	Reference
The	exhaust	assembly	shall	have its outlet no more than 60cm above the ground	3.5.5.2
The	exhaust	assembly	shall	have any protruding components in front of the main roll hoop shielded to prevent contact by persons approaching the car or the driver exiting	3.5.5.2
The	fuel tank	assembly	shall	be shielded from side impact collisions (by structure that meets 3.3.8)	3.5.3.9
The	fuel tank	assembly	shall	be of any capacity	3.5.3.1FH
The	fuel tank	assembly	shall	have drain fitting component	3.5.3.1FH
The	fuel tank	assembly	shall	have a filler cap component	3.5.3.2FH

Table A2.3. Structure and Safety Requirements

The	Object	Level	Shall	Requirement	Reference
The	chassis	sub-system	shall	have a jacking point assembly	3.2.6
The	chassis	sub-system	shall	include a primary structure assembly	3.3.1
The	chassis	sub-system	shall	include an impact attenuator component	3.3
The	head restraint	sub-system	shall	have a minimum area of 232 sq. cm	3.4.4
The	head restraint	sub-system	shall	be padded with an energy absorbing material	3.4.4
The	head restraint	sub-system	shall	be attached in a manner that can withstand a 890N force in the rearward direction	3.4.4
The	main roll hoop	sub-system	shall	include a roll bar pad over all areas than can be impacted by the driver's head	3.4.4
The	driver's restraint harness mount	components	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	driver's restraint harness mount	components	shall	have a wall thickness of 1.75mm	3.3.3.1
The	front bulkhead	components	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	front bulkhead	components	shall	have a wall thickness of 1.75mm	3.3.3.1
The	front bulkhead support	components	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	front bulkhead support	components	shall	have a wall thickness of 1.5mm	3.3.3.1
The	roll hoop brace	components	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	roll hoop brace	components	shall	have a wall thickness of 1.75mm	3.3.3.1
The	roll hoop	components	shall	have 4.5mm inspection holes drilled in non-critical locations	3.3.9
The	roll hoop	components	shall	prevent the driver's head and hands from contacting the ground at any rollover attitude	3.3.4
The	roll hoop	components	shall	allow 50mm of clearance between a line extending from the front to main hoop and all drivers and the 95th percentile male template	3.3.4
The	roll hoop	components	shall	not have a bend radius of less than three times their outside diameters	3.3.4.1
The	roll hoop	components	shall	have smooth and continuous bends with no evidence of crimping	3.3.4.1
The	roll hoop	components	shall	be securely integrated into the primary structure assembly	3.3.4.1
The	side impact	components	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	side impact	components	shall	have a wall thickness of 1.75mm	3.3.3.1
The	side impact	components	shall	be comprised of at least three tubular members on each side of the driver	3.3.8.1
The	diagonal side impact	component	shall	connect the upper and lower side impact components forward of the main hoop and rearward of the front hoop	3.3.8.1
The	energy absorbing material	component	shall	have a minimum thickness of 38mm	3.4.4
The	energy absorbing material	component	shall	be less than 25mm away from the helmet of the driver in the uncompressed state	3.4.4
The	floor closeout	component	shall	be made of at least one panel	3.4.5
The	floor closeout	component	shall	have gaps between panels of less than 3mm	3.4.5
The	floor closeout	component	shall	extend from the foot area to the firewall	3.4.5

Table A2.3 Continued.

The	Object	Level	Shall	Requirement	Reference
The	floor closeout	component	shall	prevent track debris from entering the car	3.4.5
The	floor closeout	component	shall	be made of a solid non-brittle material	3.4.5
The	front bodywork	component	shall	have no forward facing sharp edges or other protruding components	3.3.7
The	front bodywork	component	shall	have no forward facing edges with a radius of less than 38mm 45 degrees from the forward direction	3.3.8.1
The	front bulkhead	component	shall	be constructed of closed steel tubing	3.3.6.1
The	front bulkhead	component	shall	be located forward of all non-crushable objects	3.3.6.1
The	front bulkhead	component	shall	be located so the soles of the driver's feet, when touching the pedals, are rearward of the bulkhead plane	3.3.6.1
The	front bulkhead	component	shall	be securely integrated to the frame	3.3.6.2
The	front bulkhead	component	shall	be supported back to the front roll hoop by at least three frame members on each side with the top member being at least 50.8mm from the top of the component	3.3.6.2
The	front bulkhead	component	shall	be supported with frame members connected node-to-node forming triangles	3.3.6.2
The	front bulkhead	component	shall	be supported with closed section tubing	3.3.6.2
The	front roll hoop	component	shall	be no lower than the top of the steering wheel	3.3.4
The	front roll hoop	component	shall	be braced no lower than 50mm from its top	3.3.4
The	front roll hoop	component	shall	be integrated into the frame and surrounding structure	3.3.4
The	front roll hoop	component	shall	be constructed of closed section metal tubing	3.3.4.3
The	front roll hoop	component	shall	extend from the lowest frame member on one side of the frame up over and down to the lowest frame member on the other side of the frame	3.3.4.3
The	front roll hoop	component	shall	be no more than 250mm forward of the steering wheel component	3.3.4.3
The	front roll hoop	component	shall	have no part included greater than 20 deg from vertical	3.3.4.3
The	front roll hoop	component	shall	be braced by two main front hoop brace components	3.3.5.2
The	front roll hoop	component	shall	be braced from the rear if its angle of inclination is >10 deg from vertical	3.3.5.2
The	front roll hoop brace	component	shall	be integrated into the frame and surrounding structure	3.3.4
The	front roll hoop brace	component	shall	protect the drivers legs	3.3.5.2
The	front roll hoop brace	component	shall	extend forward of the front roll hoop	3.3.5.2
The	impact attenuator	component	shall	be forward of the front bulkhead	3.3.1
The	impact attenuator	component	shall	be installed forward the front bulkhead	3.3.6.3
The	impact attenuator	component	shall	be at least 200mm long	3.3.6.3
The	impact attenuator	component	shall	be oriented along the fore/aft axis of the frame	3.3.6.3
The	impact attenuator	component	shall	be at least 100mm high	3.3.6.3
The	impact attenuator	component	shall	be at least 200mm for a minimum distance of 200mm forward of the front bulkhead	3.3.6.3
The	impact attenuator	component	shall	not penetrate the front bulkhead in the event of an impact	3.3.6.3
The	impact attenuator	component	shall	include a 1.5mm steel plate or 4.0mm aluminum plate of foam filled or honeycomb	3.3.6.3
The	impact attenuator	component	shall	be attached securely and directly to the front bulkhead	3.3.6.3
The	impact attenuator	component	shall	provide a minimum deceleration of 20gs in the event of the impact of a 450kg car with a solid, non-yielding barrier with an impact velocity of 7m/s	3.3.6.4(FH)
The	lower side impact	component	shall	connect the bottom of the main and front hoops	3.3.8.1
The	main roll hoop	component	shall	have an outer diameter of greater than 25.0mm	3.3.3.1

Table A2.3 Continued.

The	Object	Level	Shall	Requirement	Reference
The	main roll hoop	component	shall	have a wall thickness of 2.5mm	3.3.3.1
The	main roll hoop	component	shall	be braced no lower than 16cm from its top	3.3.4
The	main roll hoop	component	shall	be braced with at least a 30 deg included angle	3.3.4
The	main roll hoop	component	shall	be constructed of a single piece of uncut, continuous, close section steel tubing	3.3.4.2
The	main roll hoop	component	shall	extend from the lowest frame member on one side of the frame up over and down to the lowest frame member on the other side of the frame	3.3.4.2
The	main roll hoop	component	shall	have its top portion within 10 deg of vertical	3.3.4.2
The	main roll hoop	component	shall	have its vertical members at least 380mm apart at the location where the component is attached to the major structure of the frame	3.3.4.2
The	main roll hoop	component	shall	be braced by two main roll hoop brace components	3.3.5.1
The	main roll hoop brace	component	shall	be constructed of closed section steel tubing	3.3.5.1
The	main roll hoop brace	component	shall	extend from the main hoop on the same side it is inclined towards	3.3.5.1
The	main roll hoop brace	component	shall	be attached as close to the top of the main hoop as possible	3.3.5.1
The	main roll hoop brace	component	shall	be straight (no bends)	3.3.5.1
The	main roll hoop brace	component	shall	transmit load from the main hoop to the major structure of the frame without compromising	3.3.5.1
The	main roll hoop brace	component	shall	not transmit all loads through the engine or transmission	3.3.5.1
The	roll hoop brace	component	shall	be securely attached to the frame using 8mm Grade 8.8 bolts (if not welded)	3.3.5.3
The	roll hoop brace	component	shall	use mounting plates at least 2.0mm thick	3.3.5.3
The	roll hoop brace	component	shall	not allow the drivers shoulder to pass under or neck to contact	3.3.5.4
The	seat	component	shall	be no lower than the bottom surface of the lower frame rails or:	3.4.13
The	seat	component	shall	have a tube the meets the requirements for side impact tubing pass below the seat	3.4.13
The	roll bar pad	component	shall	be comprised of an energy absorbing material	3.4.4
The	roll bar pad	component	shall	have a minimum thickness of 12mm	3.4.4
The	driver's leg protection	assembly	shall	keep all moving or sharp components between the front roll hoop and a vertical plane 100mm rearward of the pedals away from the driver's legs	3.4.14
The	driver's leg protection	assembly	shall	be consist of a solid shield	3.4.14
The	driver's leg protection	assembly	shall	allow access to the suspension and steering components	3.4.14
The	driver's leg protection	assembly	shall		3.5.11(FH)
The	shoulder harness mounting	component	shall	have an outer diameter of greater than 25.0mm	3.3.3.1
The	shoulder harness mounting	component	shall	have a wall thickness of 2.5mm	3.3.3.1
The	upper side impact	component	shall	connect the main hoop and front hoop at a height between 300mm and 350mm above the ground with a 77kg driver seated in the normal driving position	3.3.8.1
The	firewall	assembly	shall	separate the driver compartment from all components of the fuel supply, engine oil and liquid cooling systems	3.4.10.1
The	firewall	assembly	shall	protect the neck of the tallest driver	3.4.10.1

Table A2.3 Continued.

The	Object	Level	Shall	Requirement	Reference
The	firewall	assembly	shall	extend sufficiently far upwards and/or rearwards such that any point less than 100mm above the bottom of the helmet of the tallest driver is not direct sight with any part of the fuel, cooling or oil systems	3.4.10.1
The	firewall	assembly	shall	be made of a non-permeable surface	3.4.10.1
The	firewall	assembly	shall	be fire-resistant	3.4.10.1
The	firewall	assembly	shall	utilize grommets to seal pass-through	3.4.10.1
The	firewall	assembly	shall	be sealed at joints between any panels	3.4.10.1
The	frontal impact	assembly	shall	completely contain the driver's feet	3.3.6
The	jacking point	assembly	shall	be capable of supporting the car's weight	3.2.6
The	jacking point	assembly	shall	be capable of engaging the organizers' quick jacks	3.2.6
The	jacking point	assembly	shall	be at the rear of the car	3.2.6
The	jacking point	assembly	shall	be oriented horizontally	3.2.6
The	jacking point	assembly	shall	be perpendicular to the centerline of the car	3.2.6
The	jacking point	assembly	shall	be made from round 25-29mm O.D. tube	3.2.6
The	jacking point	assembly	shall	be made from aluminum or steel	3.2.6
The	jacking point	assembly	shall	be a minimum of 300mm long	3.2.6
The	jacking point	assembly	shall	be exposed around the lower 180 degrees of its circumference	3.2.6
The	jacking point	assembly	shall	be exposed over a minimum length of 280mm	3.2.6
The	jacking point	assembly	shall	be at least 75mm above the ground at its lowest point	3.2.6
The	jacking point	assembly	shall	allow the wheels to be clear of the ground at full rebound when the bottom of the assembly is 200mm above ground	3.2.6
The	primary structure	assembly	shall	include a front roll hoop component	3.3.1
The	primary structure	assembly	shall	include a main roll hoop component	3.3.1
The	primary structure	assembly	shall	include roll hoop brace components	3.3.1
The	primary structure	assembly	shall	include side impact components	3.3.1
The	primary structure	assembly	shall	include front bulkhead components	3.3.1
The	primary structure	assembly	shall	include front bulkhead support components	3.3.1
The	primary structure	assembly	shall	include all components that transfer load from the driver's restraint sub-system into the primary structure components	3.3.1
The	primary structure	assembly	shall	be constructed of round, mild or alloy, steel tubing with a minimum of 0.1% carbon	3.3.3
The	chassis	sub-system	shall	have a jacking point assembly	3.2.6
The	chassis	sub-system	shall	include a primary structure assembly	3.3.1
The	chassis	sub-system	shall	include an impact attenuator component	3.3

Table A2.4. Control Requirements

The	Object	Level	Shall	Requirement	Reference
The	brake light	component	shall	be of at least 15W or equivalent	3.2.5.3
The	brake light	component	shall	emit red light	3.2.5.3
The	brake light	component	shall	be clearly visible from the rear of the car in very bright sunlight	3.2.5.3
The	brake light	component	shall	be mounted between the wheel centerline and driver's shoulder level vertically	3.2.5.3
The	brake light	component	shall	be approximately on the vehicle's centerline laterally	3.2.5.3
The	brake over-travel switch	component	shall	shut down all drive systems and trip the accumulator isolation relays	3.2.5.2FH
The	primary master switch	component	shall	be red, 60mm diameter latching type	3.4.9FH
The	quick-release	component	shall	be capable of being operated by the driver while in a normal driving position with gloves on	3.4.6.2
The	brake light	assembly	shall	be equipped with a brake light component	3.2.5.3
The	brake over-travel switch	assembly	shall	kill the ignition and cut power to any electrical fuel pumps	3.2.5.2

Table A2.4 Continued.

The	Object	Level	Shall	Requirement	Reference
The	brake over-travel switch	assembly	shall	not allow a repeated actuation of the switch to restore power	3.2.5.2
The	brake over-travel switch	assembly	shall	be designed so that the driver cannot reset it	3.2.5.2
The	brake over-travel switch	assembly	shall	not be implemented by digital components	3.2.5.2
The	master switch	assembly	shall	be capable of stopping the engine	3.4.9
The	master switch	assembly	shall	be identified with the international electrical symbol	3.4.9
The	master switch	assembly	shall	shut down the engine and any other energy generation systems	3.4.9FH
The	master switch	assembly	shall	break the flow of the current holding the accumulator relays closed	3.4.9FH
The	master switch	assembly	shall	have three switches	3.4.9FH
The	master switch	assembly	shall	have switches located to the driver's left and right at approximately the level of the driver's head and on the dash in reach of the driver	3.4.9FH
The	steering wheel	assembly	shall	have a continuous perimeter	3.4.6.1
The	steering wheel	assembly	shall	be near circular or near oval	3.4.6.1
The	steering wheel	assembly	shall	have a quick-release component	3.4.6.2
The	wheel	assembly	shall	use a positive nut retention if using a single wheel nut	3.2.3.1
The	vehicle control	sub-system	shall	be operated from inside the cockpit without any part of the driver being outside the planes of the side impact assembly	3.4.12
The	braking	sub-system	shall	act on all four wheels	3.2.5
The	braking	sub-system	shall	be operated by a single control	3.2.5
The	braking	sub-system	shall	have two independent hydraulic circuits with separate reservoirs	3.2.5
The	braking	sub-system	shall	be capable of locking all four wheels in the brake test	3.2.5
The	braking	sub-system	shall	not use "brake-by-wire"	3.2.5
The	braking	sub-system	shall	not use unarmored plastic lines	3.2.5
The	braking	sub-system	shall	be projected by scatter shields from failure of the drivetrain or collisions	3.2.5
The	braking	sub-system	shall	have an over travel switch assembly that stops the engine if the brakes fail	3.2.5.2
The	braking	sub-system	shall	have a brake light assembly	3.2.5.3
The	braking	sub-system	shall	have its last 50% travel operate the hydraulic system described in 3.2.5 (the first 50% travel may be used for regen. braking)	3.2.5FH
The	steering	sub-system	shall	have positive steering stops	3.2.4
The	steering	sub-system	shall	have less than 7deg of steering system play measured at the steering wheel	3.2.4
The	steering	sub-system	shall	have less than +/-3 degrees of rear wheel steering	3.2.4
The	steering	sub-system	shall	be mechanically connected to front wheels	3.2.4

Table A2.5. Suspension Requirements

The	Object	Level	Shall	Requirement	Reference
The	rain tire	component	shall	have a manufacturer supplied tread or groove pattern	3.2.3.2
The	rain tire	component	shall	have a minimum of 2.4mm tread depth	3.2.3.2
The	suspension	sub-system	shall	have shock absorbers front and rear	3.2.1
The	suspension	sub-system	shall	have a usable wheel travel of at least 50.8mm per wheel	3.2.1
The	suspension	sub-system	shall	have at least 25.4mm of jounce and 25.4mm of rebound with driver seated	3.2.1
The	suspension	sub-system	shall	have at least 25.4mm of ground clearance with driver	3.2.2
The	tire	component	shall	not use warmers or traction enhancers	3.2.3.2

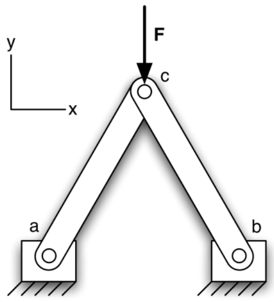
Table A2.6. Driver's Safety Requirements

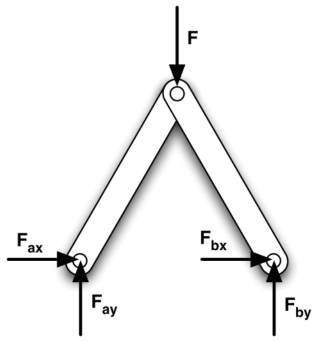
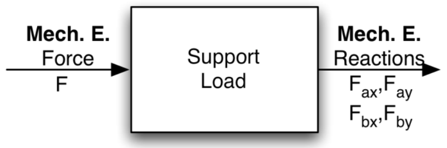
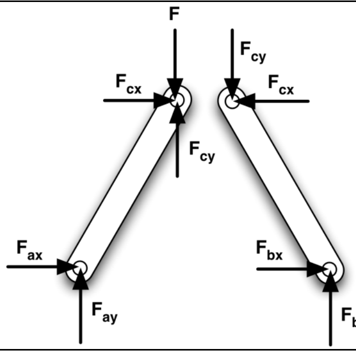
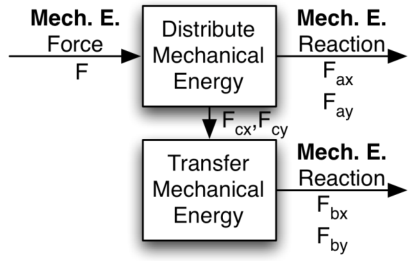
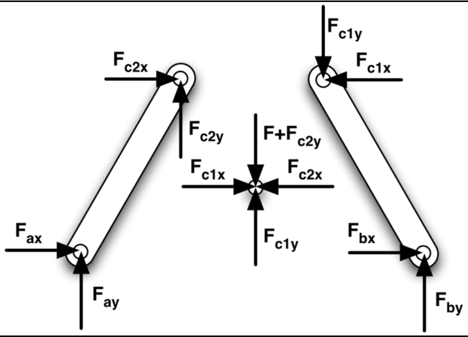
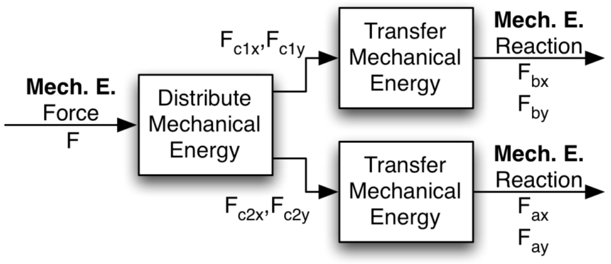
The	Object	Level	Shall	Requirement	Reference
The	shoulder harness	component	shall	be of the over-the-shoulder type	3.4.1
The	shoulder harness	component	shall	be of a separate strap design	3.4.1
The	shoulder harness	component	shall	be 76mm wide	3.4.1
The	shoulder harness	component	shall	be threaded through three bar adjusters	3.4.1
The	shoulder harness	component	shall	be mounted behind the driver to a structure that meets section 3.3.3 of the FSAE rules	3.4.1
The	shoulder harness	component	shall	have mounting points between 178mm and 229mm apart	3.4.1

Table A2.6 Continued.

The	shoulder harness	component	shall	must be between 10 deg. above and 20 deg. below horizontal	3.4.1
The	lap belt	component	shall	pass around the pelvic area of the driver below the Anterior Superior Illiac Spines	3.4.1
The	lap belt	component	shall	be at an able of between 45 and 65 degrees to horizontal	3.4.1
The	lap belt	component	shall	not be routed over the sides of the seat	3.4.1
The	driver's restraint harness	assembly	shall	be of 5,6 or 7 point design	3.4.1
The	driver's restraint harness	assembly	shall	be made of Nylon or Dacron and in new or perfect condition	3.4.1
The	driver's restraint harness	assembly	shall	use a single quick-release type latch	3.4.1
The	driver's restraint harness	assembly	shall	meet SFI Specification 16.1 or FIA specification 8853/98	3.4.1
The	driver's restraint harness	assembly	shall	be replaced following 12/31 on the 2nd year after the date of manufacture (SFI)	3.4.1
The	driver's restraint harness	assembly	shall	be replaced following 12/31 on the year marked on the label (FIA)	3.4.1
The	driver's restraint harness	assembly	shall	consist of a 76mm lap belt, 76mm shoulder straps and a 51mm anti-submarine strap (5-point)	3.4.1
The	driver's restraint harness	assembly	shall	consist of a 76mm lap belt, 76mm shoulder straps and two 51mm anti-submarine strap (5-point)	3.4.1
The	driver's restraint harness	assembly	shall	be securely attached to the primary structure assembly	3.4.1
The	driver's restraint harness	assembly	shall	be connected by freely pivoting attachments	3.4.1

APPENDIX 3. STATICS ANALOGY



Statics Free-Body Diagrams	Corresponding Functional Models
	 Mech. E. Force F Support Load Mech. E. Reactions F_{ax},F_{ay} F_{bx},F_{by}
 F F_{cy} F_{cx} F_{cx} F_{cy} F_{ax} F_{bx} F_{ay} F_{by}	 Mech. E. Force F Distribute Mechanical Energy Mech. E. Reaction F_{ax} F_{ay} F_{cx},F_{cy} Transfer Mechanical Energy Mech. E. Reaction F_{bx} F_{by}
 F_{c1y} F_{c2x} F_{c1x} F_{c2y} F+F_{c2y} F_{c1x} F_{c2x} F_{c1y} F_{ax} F_{bx} F_{ay} F_{by}	 F_{c1x},F_{c1y} Transfer Mechanical Energy Mech. E. Reaction F_{bx} F_{by} Mech. E. Force F Distribute Mechanical Energy F_{c2x},F_{c2y} Transfer Mechanical Energy Mech. E. Reaction F_{ax} F_{ay}

APPENDIX 4. VARIABLE DEFINITIONS

Table A4.1 Type I Model Variables

Function	Sym.(E)	Sym.(L)	Type	Units	Description
Convert Chem. E. to Rot. E.	P_E	$\mathrm{ROTE}_{Ia,0,0}$	Flow	kW	Power Output
	P_F	$\mathrm{CHME}_{Ia,0,0}$	Flow	kW	Power Input
	ξ_E	PARM_0	Parameter	[0,1]	Conversion Efficiency
	$P_{E,max}$	PARM_1	Parameter	kW	Maximum Conversion Power
	C_E	$\mathrm{CNTL}_{Ia,0,0}$	Control	[0,1]	Engine Control
Prov. Elec. E.	P_A	$\mathrm{ELCE}_{Ia,0,0}$	Flow	kW	Power Flow
	C_A	Int_0	Internal	unitless	Charge
	E_{max}	PARM_0	Parameter	kJ	Max. Energy Storage
	ξ_A	PARM_1	Parameter	(0,1]	Provision Efficiency
Convert Elec.. E. to Rot. E.	P_M	$\mathrm{ROTE}_{Ia,0,0}$	Flow	kW	Rotational Power
	P_A	$\mathrm{ELCE}_{Ia,0,0}$	Flow	kW	Electrical Power
	ξ_M	PARM_0	Parameter	(0,1]	Conversion Efficiency
	$P_{M,max}$	PARM_1	Parameter	kW	Maximum Power
	C_M	$\mathrm{CNTL}_{Ia,0,0}$	Control	[-1,1]	Motor Control
Dist. Rot. E.	P_M	$\mathrm{ROTE}_{Ia,0,0}$	Flow	kW	Motor Power
	P_E	$\mathrm{ROTE}_{Ia,1,0}$	Flow	kW	Engine Power
	P_D	$\mathrm{ROTE}_{Ia,2,0}$	Flow	kW	Drive Power
Dist. Mech. E.	P_D	$\mathrm{ROTE}_{Ia,0,0}$	Flow	kW	Drive Power
	P_B	$\mathrm{ROTE}_{Ia,1,0}$	Flow	kW	Brake Power
	m_H	PARM_0	Parameter	kg	Vehicle Mass
	v_H	$\mathrm{STAS}_{Ia,0,0}$	Status	m/s	Vehicle Speed
Transfer Mech. E.	P_B	$\mathrm{ROTE}_{Ia,0,0}$	Flow	kW	Brake Power
	C_B	$\mathrm{CNTL}_{Ia,0,0}$	Control	[0,1]	Brake Control
	$P_{B,max}$	PARM_0	Parameter	kW	Max Braking Power
Process Control	v_H	$\mathrm{STAS}_{Ia,0,0}$	Status	m/s	Vehicle Speed
	v_T	N/A	Internal	m/s	Target Speed
	d_H	N/A	Internal	m	Distance Traveled
	g_C	PARM_0	Parameter	s/m	Control Gain
	C_T	N/A	Internal	[-1,1]	Speed Control
	C_E	$\mathrm{CNTL}_{Ia,0,0}$	Control	[-1,1]	Electrical Throttle
	C_C	$\mathrm{CNTL}_{Ia,1,0}$	Control	[0,1]	Chemical Throttle
	C_B	$\mathrm{CNTL}_{Ia,2,0}$	Control	[0,1]	Brake Control
	f()	N/A	Function	m/s	Target Speed Profile

Table A4.2. Type II Model Variables

Function	Sym.(E)	Sym.(L)	Type	Units	Description
Convert Chem. E. to Rot. E.	M_E	$\mathrm{ROTE}_{IIa,0,0}$	Flow	N*m	Engine Moment
	ω_E	$\mathrm{ROTE}_{IIa,0,1}$	Flow	rad/s	Engine Speed
	P_F	$\mathrm{CHME}_{Ia,0,0}$	Flow	kW	Fuel Power Input
	ξ_E	PARM_0	Parameter	[0,1]	Conversion Efficiency
	f	N/A	Function	N*m	Spline Fit Function
	C_E	$\mathrm{CNTL}_{Ia,0,0}$	Control	[0,1]	Engine Control
Prov. Chem. E.	V_B	$\mathrm{ELCE}_{IIa,0,0}$	Flow	V	Battery Voltage
	I_B	$\mathrm{ELCE}_{IIa,0,1}$	Flow	A	Battery Current
	C_A	N/A	Internal	unitless	Charge
	$V_{B,nom}$	PARM_0	Parameter	V	Nominal Voltage
	P_B	PARM_1	Parameter	Ω	Internal Resistance
	$Q_{B,max}$	PARM_2	Parameter	A*s	Battery Capacity
Convert Elec. E. to Rot. E.	M_M	$\mathrm{ROTE}_{IIa,0,0}$	Flow	N*m	Motor Moment
	ω_M	$\mathrm{ROTE}_{IIa,0,1}$	Flow	rad/s	Motor Speed
	V_M	$\mathrm{ELCE}_{IIa,0,0}$	Flow	V	Supply Voltage
	I_S	$\mathrm{ELCE}_{IIa,0,1}$	Flow	A	Supply Current
	V_{Rated}	PARM_0	Parameter	V	Rated Voltage
	P_{Max}	PARM_1	Parameter	kW	Maximum Power
	I_{Rated}	PARM_2	Parameter	A	Rated Current

Table A4.2. Continued.

Function	Sym.(E)	Sym.(L)	Type	Units	Description
Convert Elec. E. to Rot. E.	ω_T	N/A	Internal	rad/s	Transition Speed
	C_M	$CNTL_{Ia,0,0}$	Control	[-1,1]	Motor Control
Dist. Rot. E.	M_M	$ROTE_{IIa,0,0}$	Flow	N*m	Motor Moment
	ω_M	$ROTE_{IIa,0,1}$	Flow	rad/s	Motor Speed
	M_E	$ROTE_{IIa,1,0}$	Flow	N*m	Engine Moment
	ω_E	$ROTE_{IIa,1,1}$	Flow	rad/s	Engine Speed
	M_R	$ROTE_{IIa,2,0}$	Flow	N*m	Rear Wheel Moment
	ω_R	$ROTE_{IIa,2,1}$	Flow	rad/s	Rear Wheel Speed
	M_F	$ROTE_{IIa,3,0}$	Flow	N*m	Front Wheel Moment
	ω_F	$ROTE_{IIa,3,1}$	Flow	rad/s	Front Wheel Speed
	M_{Rxn}	$ROTE_{Ia,0,0}$	Flow	N*m	Frame Reaction Moment
	X_G	N/A	Internal	unitless	Engine Gear Ratio
	X_M	N/A	Internal	unitless	Motor/Engine Ratio
	f	N/A	Function	unitless	Transmission Model
Dist. Mech. E.	$F_{F,x}$	$MCHE_{IIa,0,1}$	Flow	N	Front Wheel Force, x
	$F_{F,z}$	$MCHE_{IIa,0,2}$	Flow	N	Front Wheel Force, z
	$M_{F,y}$	$MCHE_{IIa,0,0}$	Flow	N*m	Front Wheel Moment, y
	$F_{R,x}$	$MCHE_{IIa,0,1}$	Flow	N	Rear Wheel Force, x
	$F_{R,z}$	$MCHE_{IIa,0,2}$	Flow	N	Rear Wheel Force, z
	$M_{R,y}$	$MCHE_{IIa,0,0}$	Flow	N*m	Rear Wheel Moment, y
	M_{Rxn}	$ROTE_{Ia,1,0}$	Flow	N*m	Driveline Reaction
	m_H	$PARM_0$	Parameter	kg	Vehicle Mass
	CG_x	$PARM_1$	Parameter	m	Vehicle CG Location, x
	CG_z	$PARM_2$	Parameter	m	Vehicle CG Location, z
	w_H	$PARM_3$	Parameter	m	Vehicle Wheelbase
	g	$PARM_4$	Parameter	m/s^2	Gravitational Acceleration
	v_H	$STAS_{Ia,0,0}$	Status	m/s	Vehicle Speed
Transfer Mech. E.	$F_{W,x}$	$MCHE_{IIa,0,1}$	Flow	N	Wheel Force, x
	$F_{W,z}$	$MCHE_{IIa,0,2}$	Flow	N	Wheel Force, z
	M_W	$MCHE_{IIa,0,0}$	Flow	N*m	Wheel Moment, y
	M_D	$ROTE_{IIa,1,0}$	Flow	N*m	Drive Moment
	ω_W	$ROTE_{IIa,1,1}$	Flow	rad/s	Wheel Speed
	C_B	$CNTL_{Ia,0,0}$	Flow	unitless	Brake Control
	v_W	$STAS_{Ia,0,0}$	Flow	m/s	Ground Speed
	$M_{B,max}$	$PARM_0$	Parameter	N*m	Max Braking Moment
	r_W	$PARM_1$	Parameter	m	Wheel Radius
	J_W	$PARM_2$	Parameter	$kg*m^2$	Wheel Inertia, y
	f	N/A	Function	N	Pacejka '96 Magic Eq.
	κ_W	N/A	Internal	unitless	Slip Ratio
Process Control	v_H	$STAS_{Ia,0,0}$	Status	m/s	Vehicle Speed
	v_T	N/A	Internal	m/s	Target Speed
	d_H	N/A	Internal	kW	Distance Travelled
	g_C	$PARM_0$	Parameter	s/m	Control Gain
	C_T	N/A	Internal	[-1,1]	Speed Control
	C_E	$CNTL_{Ia,0,0}$	Control	[-1,1]	Electrical Throttle
	C_C	$CNTL_{Ia,1,0}$	Control	[0,1]	Chemical Throttle
	C_B	$CNTL_{Ia,2,0}$	Control	[0,1]	Brake Control
	C_S	N/A	Internal	[0,1]	Slip Control
	κ_R	$STAS_{Ia,1,0}$	Status	m/s	Rear Wheel Slip Ratio
	κ_1	$PARM_1$	Parameter	m/s	Slip Ratio Cut Start
	κ_2	$PARM_2$	Parameter	m/s	Slip Ratio Cut Stop
	f()	N/A	Function	m/s	Target Speed Profile

APPENDIX 5. COMPLETE EXAMPLE RESULTS

Table A5.1. Raw Type I Analysis Results

UN Accel	EE Accel	Auto-cross	Endur.	Mass	Cost	Provision Elec. E.	Convert Elec. E. to Rot. E.	Convert Chem. E. to Rot. E.
3.12	3.88	50.17	1043.37	315.1	6000	Low Energy	Large AC	250CC MC 2
3.14	3.90	50.17	1043.15	320.1	6800	Med. Energy	Large AC	250CC MC 2
3.19	3.88	50.20	1043.90	315.1	5500	Low Energy	Large AC	250CC MC 1
3.17	3.94	50.19	1043.05	330.1	8400	High Energy	Large AC	250CC MC 2
3.21	3.90	50.21	1043.51	320.1	6300	Med. Energy	Large AC	250CC MC 1
3.24	3.94	50.22	1043.15	330.1	7900	High Energy	Large AC	250CC MC 1
3.33	3.83	50.27	1048.40	305.1	5000	Low Energy	Large AC	Hypothetical
3.35	3.86	50.27	1046.64	310.1	5800	Med. Energy	Large AC	Hypothetical
3.38	3.90	50.29	1043.38	320.1	7400	High Energy	Large AC	Hypothetical
3.39	4.62	50.29	1043.66	315.1	5700	Low Energy	Medium AC	250CC MC 2
3.40	4.65	50.30	1043.40	320.1	6500	Med. Energy	Medium AC	250CC MC 2
3.40	4.65	50.30	1043.71	320.4	4050	Low Energy	Large SepEx	250CC MC 2
3.59	3.79	50.40	1100.11	295.1	4500	Low Energy	Large AC	Small Industrial
3.42	4.67	50.31	1043.43	325.4	4850	Med. Energy	Large SepEx	250CC MC 2
3.61	3.81	50.41	1088.22	300.1	5300	Med. Energy	Large AC	Small Industrial
3.65	3.86	50.43	1063.73	310.1	6900	High Energy	Large AC	Small Industrial
3.44	4.70	50.32	1043.41	330.1	8100	High Energy	Medium AC	250CC MC 2
3.48	4.62	50.34	1044.30	315.1	5200	Low Energy	Medium AC	250CC MC 1
3.46	4.72	50.33	1043.43	335.4	6450	High Energy	Large SepEx	250CC MC 2
3.50	4.65	50.35	1043.79	320.1	6000	Med. Energy	Medium AC	250CC MC 1
3.50	4.65	50.35	1044.42	320.4	3550	Low Energy	Large SepEx	250CC MC 1
3.52	4.67	50.36	1043.87	325.4	4350	Med. Energy	Large SepEx	250CC MC 1
3.53	4.70	50.37	1043.57	330.1	7600	High Energy	Medium AC	250CC MC 1
3.55	4.72	50.38	1043.59	335.4	5950	High Energy	Large SepEx	250CC MC 1
3.49	5.15	50.35	1043.77	306.25	4520	Low Energy	PM DC	250CC MC 2
3.52	5.12	50.36	1043.83	320.4	4200	Low Energy	Medium SepEx	250CC MC 2
3.51	5.17	50.36	1043.54	311.25	5320	Med. Energy	PM DC	250CC MC 2
3.54	5.14	50.37	1043.57	325.4	5000	Med. Energy	Medium SepEx	250CC MC 2
3.69	4.57	50.45	1049.71	305.1	4700	Low Energy	Medium AC	Hypothetical
3.55	5.23	50.37	1043.59	321.25	6920	High Energy	PM DC	250CC MC 2
3.58	5.20	50.39	1043.62	335.4	6600	High Energy	Medium SepEx	250CC MC 2
3.71	4.60	50.47	1047.34	310.1	5500	Med. Energy	Medium AC	Hypothetical
3.56	5.34	50.38	1043.83	315.1	5300	Low Energy	Small AC	250CC MC 2
3.72	4.60	50.47	1050.12	310.4	3050	Low Energy	Large SepEx	Hypothetical
3.60	5.15	50.40	1044.46	306.25	4020	Low Energy	PM DC	250CC MC 1
3.74	4.63	50.48	1047.64	315.4	3850	Med. Energy	Large SepEx	Hypothetical
3.58	5.36	50.39	1043.63	320.1	6100	Med. Energy	Small AC	250CC MC 2
3.63	5.12	50.42	1044.56	320.4	3700	Low Energy	Medium SepEx	250CC MC 1
3.62	5.17	50.41	1043.92	311.25	4820	Med. Energy	PM DC	250CC MC 1
3.75	4.65	50.49	1044.02	320.1	7100	High Energy	Medium AC	Hypothetical
3.65	5.14	50.43	1043.90	325.4	4500	Med. Energy	Medium SepEx	250CC MC 1
3.78	4.67	50.50	1044.06	325.4	5450	High Energy	Large SepEx	Hypothetical
3.61	5.42	50.41	1043.68	330.1	7700	High Energy	Small AC	250CC MC 2
3.66	5.23	50.43	1043.80	321.25	6420	High Energy	PM DC	250CC MC 1
3.69	5.20	50.45	1043.83	335.4	6100	High Energy	Medium SepEx	250CC MC 1
3.67	5.34	50.44	1044.49	315.1	4800	Low Energy	Small AC	250CC MC 1
3.69	5.36	50.45	1043.85	320.1	5600	Med. Energy	Small AC	250CC MC 1
3.73	5.42	50.47	1043.92	330.1	7200	High Energy	Small AC	250CC MC 1
3.66	5.85	50.43	1043.95	320.4	3800	Low Energy	Small SepEx	250CC MC 2
3.68	5.88	50.44	1043.81	325.4	4600	Med. Energy	Small SepEx	250CC MC 2
3.72	5.94	50.47	1043.88	335.4	6200	High Energy	Small SepEx	250CC MC 2
3.87	5.09	50.56	1050.04	296.25	3520	Low Energy	PM DC	Hypothetical
3.89	5.06	50.58	1050.49	310.4	3200	Low Energy	Medium SepEx	Hypothetical

Table A5.1 Continued.

UN Accel	EE Accel	Auto-cross	Endur.	Mass	Cost	Provision Elec. E.	Convert Elec. E. to Rot. E.	Convert Chem. E. to Rot. E.
3.89	5.12	50.57	1047.57	301.25	4320	Med. Energy	PM DC	Hypothetical
3.92	5.09	50.59	1047.69	315.4	4000	Med. Energy	Medium SepEx	Hypothetical
3.79	5.85	50.51	1044.73	320.4	3300	Low Energy	Small SepEx	250CC MC 1
3.81	5.88	50.52	1044.10	325.4	4100	Med. Energy	Small SepEx	250CC MC 1
3.93	5.17	50.60	1044.58	311.25	5920	High Energy	PM DC	Hypothetical
3.96	5.14	50.62	1044.66	325.4	5600	High Energy	Medium SepEx	Hypothetical
3.85	5.94	50.54	1044.19	335.4	5700	High Energy	Small SepEx	250CC MC 1
3.95	5.28	50.62	1050.16	305.1	4300	Low Energy	Small AC	Hypothetical
3.98	5.31	50.64	1047.35	310.1	5100	Med. Energy	Small AC	Hypothetical
4.02	5.36	50.67	1044.99	320.1	6700	High Energy	Small AC	Hypothetical
4.15	4.52	50.79	1108.28	295.1	4200	Low Energy	Medium AC	Small Industrial
4.17	4.55	50.82	1095.49	300.1	5000	Med. Energy	Medium AC	Small Industrial
4.17	4.55	50.82	1110.18	300.4	2550	Low Energy	Large SepEx	Small Industrial
4.22	4.60	50.87	1069.05	310.1	6600	High Energy	Medium AC	Small Industrial
4.19	4.58	50.84	1097.06	305.4	3350	Med. Energy	Large SepEx	Small Industrial
4.24	4.63	50.90	1070.67	315.4	4950	High Energy	Large SepEx	Small Industrial
4.11	5.79	50.76	1050.79	310.4	2800	Low Energy	Small SepEx	Hypothetical
4.14	5.82	50.78	1047.66	315.4	3600	Med. Energy	Small SepEx	Hypothetical
4.18	5.88	50.82	1046.41	325.4	5200	High Energy	Small SepEx	Hypothetical
4.47	5.03	51.20	1110.41	286.25	3020	Low Energy	PM DC	Small Industrial
4.48	5.01	51.21	1112.77	300.4	2700	Low Energy	Medium SepEx	Small Industrial
4.50	5.06	51.24	1097.75	291.25	3820	Med. Energy	PM DC	Small Industrial
4.50	5.04	51.25	1098.72	305.4	3500	Med. Energy	Medium SepEx	Small Industrial
4.55	5.09	51.32	1070.25	315.4	5100	High Energy	Medium SepEx	Small Industrial
4.55	5.12	51.31	1071.35	301.25	5420	High Energy	PM DC	Small Industrial
4.60	5.22	51.39	1111.62	295.1	3800	Low Energy	Small AC	Small Industrial
4.63	5.25	51.44	1097.51	300.1	4600	Med. Energy	Small AC	Small Industrial
4.68	5.31	51.52	1068.24	310.1	6200	High Energy	Small AC	Small Industrial
4.90	5.73	51.96	1115.19	300.4	2300	Low Energy	Small SepEx	Small Industrial
4.93	5.76	52.02	1100.04	305.4	3100	Med. Energy	Small SepEx	Small Industrial
4.98	5.82	52.13	1068.61	315.4	4700	High Energy	Small SepEx	Small Industrial

Table A5.2. Raw Type II Analysis Results

EE Accel	UR Accel	Auto-cross	Endur.	Mass	Cost	Provision Elec. E.	Convert Elec. E. to Rot. E.	Convert Chem. E. to Rot. E.
5.07	4.18	50.92	1053.2	350	8400	16 Pack	EM AC-2	Prototype
5.07	4.41	51.10	1063.1	350	7900	16 Pack	EM AC-2	WR250X
5.39	4.36	51.06	1054.8	340	6800	8 Pack	EM AC-2	Prototype
5.97	4.35	51.03	1051.7	350	7800	16 Pack	EM AC-1	Prototype
5.39	4.51	51.17	1066.4	340	6300	8 Pack	EM AC-2	WR250X
6.10	4.36	51.04	1054.0	340	6200	8 Pack	EM AC-1	Prototype
5.97	4.55	51.24	1061.3	350	7300	16 Pack	EM AC-1	WR250X
4.97	4.63	51.35	1170.0	332	6900	16 Pack	EM AC-2	Honda GX240
6.62	4.45	51.12	1055.1	335	5400	4 Pack	EM AC-1	Prototype
6.10	4.59	51.29	1065.1	340	5700	8 Pack	EM AC-1	WR250X
6.62	4.68	51.43	1067.3	335	4900	4 Pack	EM AC-1	WR250X
7.17	4.65	51.34	1056.0	335	6000	4 Pack	EM AC-2	Prototype
8.10	4.51	51.19	1054.1	331	5300	8 Pack	PMG132-400	Prototype
8.21	4.54	51.23	1053.6	341	6900	16 Pack	PMG132-400	Prototype
8.21	4.57	51.22	1054.6	326	4500	4 Pack	PMG132-400	Prototype
5.29	4.87	51.71	1186.6	322	5300	8 Pack	EM AC-2	Honda GX240
8.10	4.78	51.46	1065.6	331	4800	8 Pack	PMG132-400	WR250X
8.21	4.77	51.46	1064.1	341	6400	16 Pack	PMG132-400	WR250X
7.17	4.91	51.70	1068.5	335	5500	4 Pack	EM AC-2	WR250X
8.21	4.85	51.57	1066.2	326	4000	4 Pack	PMG132-400	WR250X
11.33	4.77	51.46	1054.6	326	4300	4 Pack	PMG132-200	Prototype
11.42	4.79	51.48	1054.2	331	5100	8 Pack	PMG132-200	Prototype
11.63	4.87	51.56	1053.5	341	6700	16 Pack	PMG132-200	Prototype

Table A5.2 Continued.

EE Accel	UR Accel	Auto-cross	Endur.	Mass	Cost	Provision Elec. E.	Convert Elec. E. to Rot. E.	Convert Chem. E. to Rot. E.
5.84	5.25	52.42	1160.7	332	6300	16 Pack	EM AC-1	Honda GX240
5.97	5.34	52.62	1180.8	322	4700	8 Pack	EM AC-1	Honda GX240
11.33	5.11	51.90	1066.2	326	3800	4 Pack	PMG132-200	WR250X
11.42	5.13	51.93	1065.5	331	4600	8 Pack	PMG132-200	WR250X
11.63	5.20	52.03	1064.0	341	6200	16 Pack	PMG132-200	WR250X
6.51	5.63	53.34	1191.5	317	3900	4 Pack	EM AC-1	Honda GX240
7.90	5.93	53.19	1183.5	313	3800	8 Pack	PMG132-400	Honda GX240
7.95	6.03	53.30	1174.7	323	5400	16 Pack	PMG132-400	Honda GX240
7.94	6.03	53.62	1188.2	308	3000	4 Pack	PMG132-400	Honda GX240
7.05	5.93	55.74	1196.0	317	4500	4 Pack	EM AC-2	Honda GX240
10.98	6.86	55.24	1187.9	308	2800	4 Pack	PMG132-200	Honda GX240
11.08	6.89	55.30	1182.6	313	3600	8 Pack	PMG132-200	Honda GX240
11.26	7.01	55.51	1171.5	323	5200	16 Pack	PMG132-200	Honda GX240

VITA

Name:	Ryan Scott Hutcheson, Ph.D.
Address:	Texas A&M University Department of Mechanical Engineering 3123 TAMU College Station, TX 77843
Email Address:	ryan.hutcheson@gmail.com
Education:	B.S., Mechanical Engineering, Missouri University of Science and Technology, 2003 M.S., Mechanical Engineering, Missouri University of Science and Technology, 2005