12-2020

A Finite State Automata-Based Description of Device States for Function Modeling of Multi-State Technical Devices

Ahmed Mohammed Sobhan Chowdhury

A Finite State Automata-Based Description of Device States for Function Modeling of Multi-State Technical Devices

by

Ahmed Mohammed Sobhan Chowdhury

A thesis submitted to the College of Engineering and Science
Florida Institute of Technology
in partial fulfillment of the requirements
for the degree of

Masters of Science
in
Mechanical Engineering

Melbourne, Florida
December, 2020

We the undersigned committee hereby approve the attached thesis, “A Finite State Automata-Based description of Device States for Function Modeling of Multi-State Technical Devices,” by Ahmed Mohammed Sobhan Chowdhury

Dr. Chiradeep Sen
Assistant Professor, Mechanical and Civil Engineering
Committee Chair

Dr. Hector Gutierrez
Professor, Mechanical and Civil Engineering

Dr. Siddhartha Bhattacharyya
Assistant Professor, Computer and Engineering Sciences

Dr. Ashok Pandit
Professor and Department Head, Mechanical and Civil Engineering

Abstract

Title: "A Finite State Automata-Based description of Device States for Function Modeling of Multi-State Technical Devices "

Author: Ahmed Mohammed Sobhan Chowdhury

Advisor: Dr. Chiradeep Sen

Many modern and innovative design problems require multi-modal, reconfigurable solutions. Function modeling (FM) is a common tool used to explore solutions in early stages of mechanical engineering design. Currently, function structure representations do not abet the modeling of formally-defined reconfigurable function models as graph-based function models used in early-stage systems design usually represent only one operational mode of the system. Currently there is a need, but no rigorous formalism to model multiple possible modes in the model and logically predict the behavior of the system as it transitions between the modes. Additionally, function modeling representations will benefit from dynamically capturing the effects of state change of a flow property on the operating mode of the system.

This thesis presents a formal representation (1) of operational modes and states of technical devices and systems based on automata theory for both discrete and continuous state transitions and a formal representation (2) to capture the duality of specific functions through four verbs that shift from one mode of operation to its logical and topological opposite, based on the existence of, or the value of a signal from, an input flow. It then presents formal definitions of three signal-processing verbs that actuate or regulate energy flows: Actuate_E, Regulate_E_Discrete, and Regulate_E_Continuous. Additionally, three conjugate verbs: CEnergize_M, CStore_E, CDistribute_M, CTypeChange_E are also presented alongside an approach to extend these functions to function features using the example of conjugate features: CHandover_E and CConvergize_EM in order to support physics-based reasoning on the interactions between flows. The graphical templates, definitions, and application of each verb in modeling is illustrated.

Lastly, FSAs are integrated with an existing software for concept modeling and system-level models are used to illustrate the verbs' modeling and reasoning ability, in terms of cause-and-effect propagation. Finally, the representation is shown to the demonstrate the ability to support reasoning on operating modes of systems, provide quantitative reasoning on the efficiency of those modes, and offer modeling efficacy to the designers.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

EE	Electrical Energy
ME	Mechanical Energy
ThE	Thermal Energy
PE	Potential Energy
FM	Function Modeling
FSA	Finite-State-Automata

Acknowledgements

First and foremost, I would like to thank my research advisor Dr. Chiradeep Sen for believing in me and my research capabilities. This thesis would not have been possible without his resolute support and forthright directions. Additionally, I would like to express my gratitude towards my committee members Drs. Hector Gutierrez and Siddhartha Bhattacharyya who have agreed to review this thesis.

I would also I like to thank my lab mates Lakshmi Narasimhon, Xiaoyang Mao, Jicmat Ali, Arnold Tsoka and Amaninder Gill who have helped me tremendously throughout my graduate studies.

Lastly but not leastly, I want to express my gratitude to my family and the family of friends that I have made here at Florida Tech. I would not have been able to persevere through my graduate degree, had it not been for the undying support and patience of my parents, Safia Perera and Monsur Ahmed, and my sister, Zubaidah Anjuman Chowdhury. My friends Alishan Premani, Joy Onyullo, Tricia Muhebwa, Daniel Hochiemy, Devanshi Shah, and Drushti Rane have all provided me with a sense of comfort and home that supported me throughout my stay at Florida Tech.

Dedication

To all the educators who have sowed and nourished my learning

"This tree has two million and seventy-five thousand leaves. Perhaps I missed a leaf or two but I do feel triumphant at having persisted in counting by hand branch by branch and marked down on paper with pencil each total. Adding them up was a pleasure I could understand; I did something on my own that was not dependent on others, and to count leaves is not less meaningful than to count the stars, as astronomers are always doing. They want the facts to be sure they have them all. It would help them to know whether the world is finite. I discovered one tree that is finite. I must try counting the hairs on my head, and you too. We could swap information."

Information, David Ignatov

Chapter 1
Introduction

1.1 Overview of Function Modeling

Function modeling is a well-accepted technique for exploring the solution space and generating concepts in the early stages of product design [1,2]. It is also a common tool for recording the knowledge of existing products discovered through reverse engineering [3,4]. Yet, a limitation of this representation lies in its inability to formally capture the different operational modes of a device within the same model and support reasoning using that information. When a function model is constructed in current practice, only one mode of the device—usually the default or dominant one—is modeled. An example is the hairdryer function model shown in Figure 1, which shows an operational mode where all the subsystems are running to produce a stream of hot air. This model was obtained from the Oregon State University Design Repository [5], although models of this type are found throughout design literature [6–9].

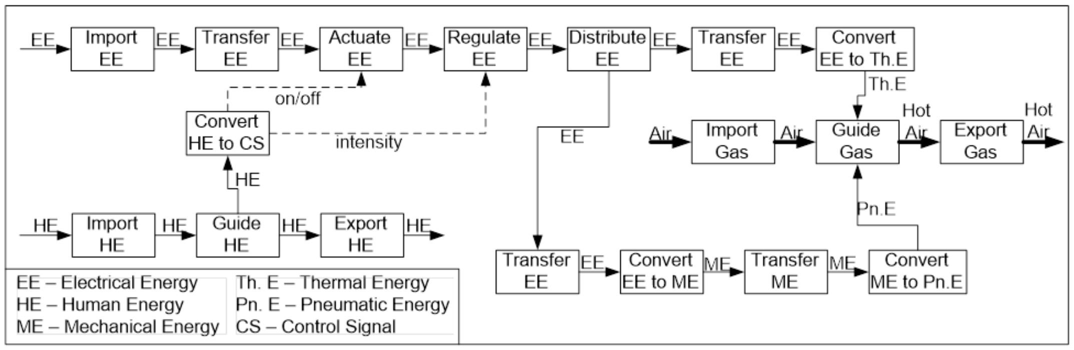


Figure 1: A sample function model, representing the normal operating mode of a hairdryer

This research is limited to the graph-based function structures shown in Figure 1. The arrows (edges) represent flows of type: Material (shown in thick solid lines, e.g. air), energy (shown in thin solid lines, e.g. EE) and Signals (shown in thin dotted lines, e.g. intensity). Functions (vertices) are shown in blocks and they refer to an action performed by the system. Based on this discussion, a few key concepts pertinent to this research are defined below:

Flow: A flow is defined as an occurrence of a material, energy or a signal that is either used or produced by an action performed by the system.

Function: A function is an occurrence of a transformative action that transforms an input set of flows into a different set of flows by changing the attributes of the input flow.

Function Model: A function model is a representation describing the functionality of an artifact based on the tail-node and head-node relations between flows and function blocks.

Function models typically capture the actions performed by a system on the flows in a single mode as functions are limited to performing a single transformative action on a flow. However, most complex devices execute multiple functions based on its mode and there exists a need for those devices to be represented in function modeling.

1.2 Need for Capturing Device States in Function Modeling

Despite its popularity, one limitation of the graph-based function models lies in their inability to capture multiple states and operational modes of a device in the same model or to use such information for computer-based reasoning [10]. Many devices are, in fact, designed to operate in multiple modes, and their functions may vary significantly between these modes [11]. A study on reconfigurable devices estimated that engineered artifacts such as consumer electronics, appliances, toys, and weapons constituted 34% of reconfigurable devices [12]. For ease of reference, we review the concepts of modes and states from [10], below.

Operational mode: The process that a system executes over time, represented by a subset of functions (subgraph of the function model) that are executed at the time, and which could change if the mode changed. For example, a car could be in any one of several modes at a time during a ride, such as idling, forward acceleration, forward braking, reverse acceleration, etc. [10].

Physical state: The physical configuration of a device at a given time, given by a single or combination of variables describing the system. For example, the shifter in a four-speed manual gearbox of a car must be in one of the following discrete states: {P, N, 1, 2, 3, 4, R}, and never outside these options [10]. The gas pedal, likewise, can assume a position within a range between the two extremes at a time – a continuous range in this case. Any combination of these two variables gives a state of the car.

As seen in the example above, it is the state of the device that decides its mode. Usually, the device's function model changes with modes, and the transition between the modes is triggered by a signal flow: either a control signal from the outside or a status signal that is an attribute of a flow. An example of the first kind (transition by control signal) is reversible leaf blowers that exist in two modes. In the blower mode, they drive air forward (subfunctions: suck air from outside, add kinetic energy, and discharge through the nozzle), while in the vacuum mode they drive air backward (subfunctions: suck air and leaves, separate leaves, bag leaves, filter air, discharge through the vents). The mode transition happens because of a control signal from the user flipping a switch. An example of the second kind (status signal) are rechargeable batteries, which, in their charging mode convert electrical to chemical energy and store the latter, while in the discharging mode convert chemical to electrical energy and supply the latter. The transition in this case occurs because of the voltage difference in the connected circuit, which is an attribute of the flows. The mode of a device is therefore determined by a control or a status signal input to the functions and they are defined below.

Control Signal: The definition of a control signal is adapted from [6] to be an operational command that is used to convey information regarding the structural state of a device. In the example of the light switch, the command "up" or "down" that is used to flip the state of the switch is a control signal.

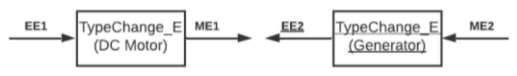
Status Signal: Status signals carry information regarding the state of a flow or a flow attribute. Information regarding the temperature, velocity, pressure, viscosity, electric charge, etc., are regarded as status signals.

Device states and modes have been discussed in function literature previously [11,13–17]. For the graph-based function models, recent research presents a formalism of modeling states using finite-state automata (FSAs) and use them to perform predictive causal reasoning [10] and proposes the idea of functional conjugacy to represent multiple operating modes of devices [18].

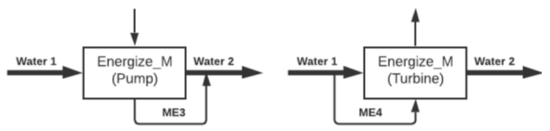
1.3 Need for Demonstrating Conjugate Behavior

By studying multi-modal systems, we recognize that devices can exist in dual modes at a component-, subsystem-, or a system-level. For example, heat transfer between a hot metal plate and the surrounding cooler air or heat transfer between a cool metal plate and hotter air are examples of dual mode at a functional level (addition or removal of thermal energy (ThE) from air by the plate) based on a status signal (temperature of the air). Similarly, a heat pump cooling or heating the indoor space depending on the state of the thermostat setting is a system level application of reverse operation based on a control signal input from the user. Systems that exist in modes that are topological opposites of each other and accomplish opposite purpose are said to demonstrate functional conjugacy, as defined below.

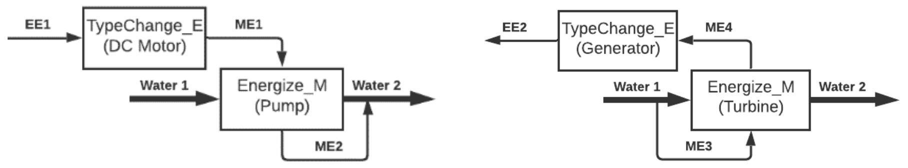
Functional conjugacy: The appearance of operational modes in a component, device, system, or a pair of function verbs that are topological opposites of each other on the function model. Examples include DC motors that function as generators and pumps that function as turbines depending on the direction of energy conversion. Figure 2 (a) and (b) show that their topologies are mutually opposite. By extension, combining these two devices will form a small system that displays conjugacy at a system level, as shown in Figure 2 (c): (1) using electricity to raise water (DC motor + pump) and (2) using falling water to produce electricity (turbine + generator).



(a) DC motor and generator



(b) Pump and turbine



(c) motor-pump assembly and turbine-generator assembly

Figure 2: Examples of functional conjugacy

Conjugacy is also seen in the verbs of function vocabularies. In the Functional Basis [19], verbs such as store and supply, actuate and stop, mix and separate, and couple and distribute are essentially topological opposites. In this paper, we call these verbs Conjugate Verbs. Examples include piping junctions that may couple or distribute flows based on the pressure gradients (status signals), capacitors that can store or supply electrical energy based on the direction of current (status signals), pressure regulators can add or remove thermal potential energy to/from its constituents based on the state of the valve (control signal), and electrochemical cells can convert chemical energy to electrical or vice-versa based on the presence or absence of an electrical current. Thus, conjugate behavior is ubiquitous. The gaps in representing device states and capturing conjugacy in function modeling establishes the research questions in the following section.

1.4 Research Questions, Hypothesis and Tasks

This section outlines the two main research questions answered in this thesis in Table 1. Each research question is supported with a hypothesis and research tasks that are used in answering each question. Research questions are numbered using the abbreviation "RQ" and research hypothesis are numbered as "RH". The sections associated with each research task is listed in parenthesis for ease of reference.

Table 1 Research questions, hypothesis, and tasks

Main Research Question and Hypothesis	RQ-1. How can device-states be represented in function modeling?
	RH-1. Integration of FSAs that capture device state into the formal definitions of functions will ensure that the resulting function model is reflective of the device state.
Sub-questions and tasks	RQ-1.1. How can the formal language capture effect of state-change of a device on the function model? Task 1. Integrate logical statements into the formal definitions of functions to determine a valid output flow from a set of multiple possible flows based on its associated FSA (Chapter 5).

	RQ-1.2. Is the proposed formalism computable? Task 2. Implement FSAs and reasoning algorithms to determine the final state of an FSA based on a signal flow input onto ConMod. Ensure that the FSA is valid (pre-requisite for reasoning) and capture the effect of state transition on the output flow (Chapter 8).
	RQ-1.3. Does the formalism support causal dependencies of functions on each other? Task 3. Develop algorithm to propagate the effect of a flow attribute change throughout the function model (Chapter 8.5).
Main Research Question and Hypothesis	RQ-2. How can function models accommodate multiple operating modes of a system?
	RH-2. Development of a thorough vocabulary of conjugate functions and features allow for functions to capture more than one mode of operation of a device (Chapter 6).
	RQ-2.1. How many conjugate verbs can be found in the ConMod language? Task 4. Establish a list of conjugate functions from a comprehensive list of the current ConMod vocabulary (Chapter 4.2).
	RQ2.2. Is the idea of conjugacy extendable to functional features? Task 5. Demonstrate the idea of conjugacy in functional features and formally define conjugate features (Chapter 7).

	RQ2.3: Can conjugacy be used to reason on the operating mode of a system? Task 6. Demonstrate the application of conjugate verbs in a system level model to study the effect of state change on the operating mode of the system (Chapter 10).
Question and Hypothesis	RQ-3. What reasoning tasks are necessary to support causal and modal reasoning on a system-level function model.
	RH-3. Identification of the state of the function, state of the flow, and propagating the effect of a change in the state of a flow through the function model based on headnode relations will achieve the above reasoning tasks (Chapter 8).

1.5 Solution Overview

Current function modeling practice does not provide a consistent means to model conjugate behaviors or perform computational reasoning on device states, which are the research gaps addressed here. We posit that addressing this gap could improve both modeling efficacy and model-based reasoning of predicting system response, and enable causal and modal reasoning on function models. In order to address this gap a finite state automata-based approach is introduced to the formal language of function modeling language of ConMod to represent device states and function states to allow multi-modal representation of a system. This is achieved through (1) the introduction of three signal processing verbs to the existing ConMod vocabulary: Actuate_E, Regulate_E_Continuous, and Regulate_E_Discrete in Chapter 5, (2) introducing a vocabulary of formally defined conjugate verbs that capture the topological reversals of select functions: CEnergize_M, CStore_E, CDistribute_M, and CTypeChange_E in Chapter 6, (3) and presenting an approach to formally define conjugate features based on signal inputs alongside two formally defined conjugate features: CHandover_E and CConvergize_EM in Chapter 7.

All the three approaches use a signal flow input to determine the state of the device/function through the finite state automata associated with the verb.

Chapter 8 shows the computability of the proposed formalism through the implementation of Actuate_E onto ConMod 2.0. Lastly, FSA-based causal reasoning capabilities of the proposed representation is discussed in Chapter 9, and conjugacy-based modal reasoning capabilities are discussed in Chapter 10. In summary, the state and mode-based representation of electromechanical products is supported through the extended vocabulary and reasoning capabilities proposed by this formalism in the following chapters.

Chapter 2
Literature Review

Functional reasoning schemes offer theories and techniques to explain and derive the functions of artifacts. While function based reasoning exist in several fields ranging from biology to sociology [13], this literature review is limited to function-based reasoning in engineering design as it is considered a crucial step in conceptualizing design solutions. More specifically, we focus only on the graph-based function models that describe devices as transformative action on material, energy, and signal flows, based on the models proposed in design texts [1–3].

2.1 Function Representations and Vocabularies

Several approaches have been proposed to represent functions in artifacts: some focus on functions as form independent such as the Functional Basis approach [20], while others consider functions to be form dependent such as the FBS [21] or the Chakrabarti and Bligh approach [22]. These various approaches lead to multiple definitions of the term function itself. Pahl, et. al., for example, consider functions as transitional operations performed between input and output flows [1]. Gero's Function Behavior Structure model describe functions as intermediates between the user's goals and the system's behavior [21], while Deng's Function-Environment-Behavior-Structure model defines functions by specifying the set of physical structures required to achieve them [23]. This paper employs the definition used by Pahl, et. al. [1], which is also found across various functional reasoning schemes such as the Functional Basis [24], the product design methodology by Ulrich, et. al. [4], and the mechanical design process suggested by Ullman [2] and Otto & Wood [25]. Some of these approaches have also been implemented in software programs. The FBS modeler, for example, enables conceptual design [26], SOFAST describes and stores functional decomposition trees in a knowledge repository [27], KRITIK and IDEAL support concept generation from a repository of FBS models [28], Schemebuilder decomposes functions into sub-functions [29], and ConMod supports qualitative and quantitative reasoning on graph based function models [30].

Per Pahl, et. al., functions performed by a system are defined as operations on energy, material, and signal flows and are represented using a graph-based representation of the product where its edges represent the flows through the system and nodes represent the operations performed on the flows by the system [1,2]. In forward engineering, they can be used to search for solutions by mapping each function block to known working principles [1]. Additional applications include identifying design faults and their propagation paths [31–33], exploring design analogies by analyzing functional similarity[34,35] and using function-based patent-knowledge retrieval tools [36], estimating market value and assembly time based on functional complexity [37,38], and in performing physics-based causal reasoning [39]. Function modeling is also used in reverse engineering where the product's sub-functions are discovered via disassembly and its functionality is analyzed to obtain different physical principles under which concept variants could operate [25]. Finally, a recent study also suggests that function modeling could be used for educational purposes to improve students' understanding of complex systems [40].

The function modeling methodology established by design texts [1,2] and function researchers [41–44] have produced methods to decompose a functional black box model into detailed function models. These models were unable of "producing repeatable function models of a particular product", and the variability within function models produced by different designers posed a question on the accuracy and consistency of the modeling language and representation [45–47]. To address this gap, efforts were made into formalizing the language for function modeling. Kirschman et. al. presented an elementary mechanical design taxonomy that aimed at providing a "common language for designers to discuss functions" to address the issue of semantic inconsistencies by dividing products into four groups: motion, control, power, and enclose [48]. Later, the Functional Basis [24] was developed where the verb-object vocabulary was standardized to form a list of broadly applicable flows and functions. The flows were divided into material, energy, and signal, and each class had a taxonomy of sub-classes. Through vast empirical studies, functions were also categorized into primary, secondary and tertiary tiers, with each term worded by human modelers called them in their studies [24]. The Functional Basis was later reconciled to a set of 42 flows and 53 functions that was compiled to be used as a foundation for design repositories and to support new design methods and design teaching [20]. The primary and secondary class of verbs within the Functional Basis are represented in Table 2.

Table 2: The Functional Basis function set (only primary and secondary levels shown)

Primary level	branch	channel	connect	control magnitude
Secondary level	separate	import	couple	actuate
	distribute	export	mix	regulate
		transfer		change
		guide		stop
Primary level	convert	provision	signal	support
Secondary level		store	sense	stabilize
		supply	indicate	secure
			process	position

2.2 Function-Based Reasoning

The Design Repository [5] is a research tool and a database of design information of technical products that supports concept generation and other design tasks, and serves as an archive of existing design knowledge [49–51]. The Functional Basis has been used to model over 250 products to date, which are stored within the Design Repository, and the application of information metrics on function models from the Design Repository has shown that formalizing the grammar for the Functional Basis greatly reduced the ambiguity and variations in the models [52]. The Functional Basis's vocabulary only defines the verbs in a textual manner, however, formal definitions controlling the topology of the model has shown to increase the expressiveness of the vocabulary [53,54], which makes it more suitable for computer-based reasoning applications [55]. Approaches to function-based reasoning applications include failure analysis [32], causal reasoning [39,41], design concept generation [49,56], concept generation from black box models [57], identifying functional similarities between products [58], and in automated synthesis of functions using genetic algorithms [59] and through datamining of the design repository [60]. Additionally, reasoning capabilities will aid in applications of AI on function modelling. Progressions on more AI based approaches such as problem decomposition [21,61], analogical reasoning [26], qualitative and semantic reasoning[62], and causal analysis [41] will be possible as AI algorithms will be more compatible with a more expressive language [63]. Efforts were also made in developing a graph grammar for function modelling which was followed by a human study to demonstrate consistency in function modelling when the grammar was applied [46,47].

2.3 Representation of States and Modes

Chandrasekaran identified the concept of "mode of deployment" which captured the relationship between structural configuration of the device and the functional effect produced by the system [14]. The need for capturing device states into function modeling has been recognized in other function representations. The Function-Behavior-State (FBS) model [64] describes functions based on the state of the device, the Structure-Behavior-

Function model recognizes the function of a device to be a product of its structural state [28], and the purpose-function-working space-structure-behavior framework describes the physical state of the device through its behavior [65]. These approaches, however, are incompatible with the graph-based representation of functions on which we focus in this paper. The need for state representation was also explored in other modeling methodologies. For example, Gupta, et. al., proposed an interaction state (akin to functional modes) based modeling framework to aid in conceptual design of mechatronic devices using state-transitions [66]. In applications of bond graph theory, a concept of switched power junctions [67] and a hybrid bond graph modelling technique using finite state automata have been introduced to model multi-modal systems [68], applications of which can be seen in domain-specific multimodal systems such as automotive powertrains [69,70]. The System State Flow Diagram (SSFD) uses a conditional fork node to support modeling of multi-modal products [16]. Lastly, the Integrated Function Modeling framework captures multiple process views to characterize different states of a system by integrating multiple modeling frameworks onto a single one [71].

In graph based function modeling literature, Chakrabarti introduced the need to represent multi-modal products after logically examining function-structure relationships in engineered devices [15]. Buur proposed a theoretical approach to capture multiple operating modes in function modeling through a hierarchical function model where a function branch may or may not be executed based on the output of a branch on a higher level than itself [72]. The Function Design Framework captures multiple event-level function models that capture different working modes in a higher level system model [73]. More recently, a reconfigurable function design methodology proposed a method to capture multiple working modes using a logic gate controlled by a state-transition diagram [74].

This paper proposes a finite-state-automata based approach to formally capture different operating modes of devices in in the graph-based function models, which was heretofore unrealizable. To achieve this goal, this paper will utilize an existing formal function-modeling language that is demonstrably consistent with the laws of classical physics while constructing graph-based function models [75]. This language provides an extendible architecture of progressively more complex languages, each layer of which is founded on the proven soundness of the lower layers [76]. The first layer defines the basic concepts of function modeling such as function, flow, and environment in a manner that guarantees that the models are consistent with the balance laws of mass and energy and the principle of irreversibility [30,75]. The second layer extends this ability to define a physics-based vocabulary of functions [77]. The third layer proposes the concept of functional features, i.e., features available on the tool set of a function modeling software that designers could reuse in modeling and obtain more advanced reasoning ability [76,78,79]. The current paper builds upon some of these concepts, especially the definitions of certain features described in [76].

Chapter 3
Frame of Reference

In this chapter, we briefly review the past work on which this paper is directly built. The representation of conjugacy presented here builds on the recent formalization of modes and states within graph-based function models [10], as mentioned earlier. That ability, in turn, depends on a formal language of function modeling [75] and the introduction of feature-based modeling of functions [78,80,81], as reviewed below.

3.1 Structure of ConMod Language of Function Modeling

The Concept Modeler, or ConMod, is a hierarchy of formal languages, language extensions, and their computer implementation for constructing graph-based function models on the computer and using them in model-based reasoning. The language hierarchy is shown in Figure 3.

Layer 1 provides a vocabulary of terms such as function, flow, material, energy, signal, environment, and relations such as head node, tail node, and carrier flows, which permit constructing function graphs using plain-text function and flow names [75].A set of 33 grammar rules ensure that the models are consistent with the first and second laws of thermodynamics and enable reasoning such as checking physics-wise correctness and feasibility of modeled concepts and calculating system efficiency [30]. Layer 2 provides physics-wise consistent, formal definitions of eight verbs and two balance nodes, which extend the previous ability to keyword-based function and flow names [82]. Layer 3 extends this ability to the idea of features—encapsulations of repeatedly used functional subgraphs or snippets that are formalized into modeling entities that can be directly added to models and support feature-level reasoning [80]. This layer presents an extended vocabulary of seven verb features specifically designed for function modeling and analysis of thermal-fluid systems [78,81].

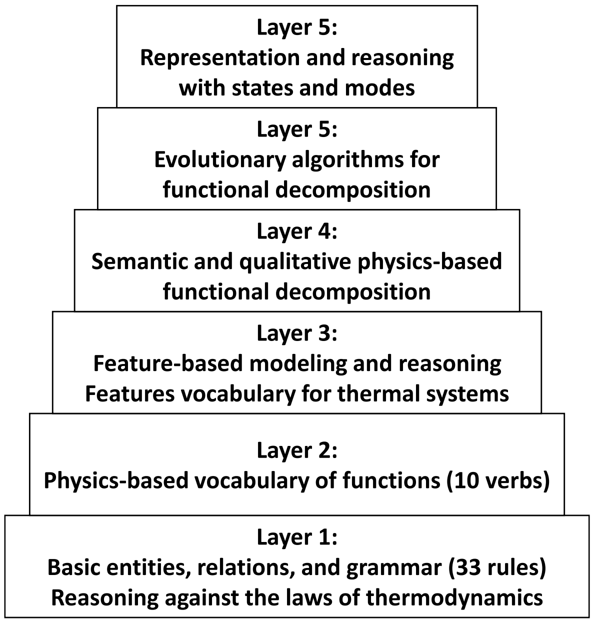

Layer 5:
Representation and reasoning with states and modes
Layer 5:
Evolutionary algorithms for functional decomposition
Layer 4:
Semantic and qualitative physics-based functional decomposition
Layer 3:
Feature-based modeling and reasoning
Features vocabulary for thermal systems
Layer 2:
Physics-based vocabulary of functions (10 verbs)
Layer 1:
Basic entities, relations, and grammar (33 rules)
Reasoning against the laws of thermodynamics

Figure 3: The ConMod language hierarchy

The function structure construction using the ConMod language is regulated by the grammar rules and the Entity Relation Attribute diagram in Figure 4 shows the all the valid relationships between the nodes and nouns. The rectangles are entities, the relations are shown in diamonds and the ellipses constitute attributes associated with each entity. Entity-Relations can have a one to one (1-1), one to many (1-n), or many to many (m-n) relationship. Every entity type, relation type (except is_a) and attribute in the ERA diagram is instantiable in the ConMod software. The vocabulary established in layers 2 and 3 are implemented in this software and this research will utilize ConMod to validate the concept of FSA implementation (Layer 5).

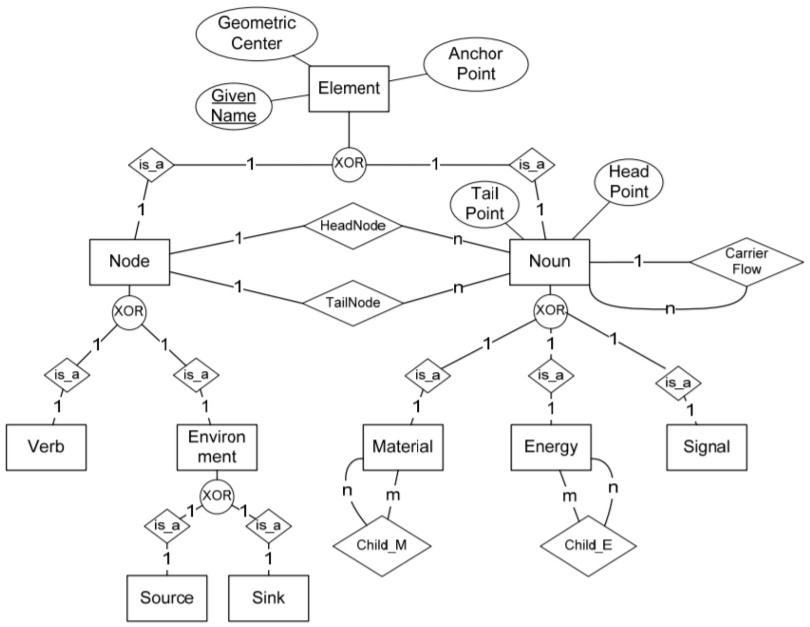


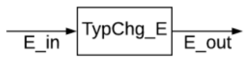
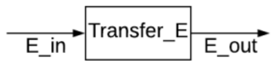
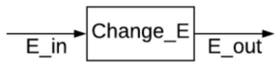
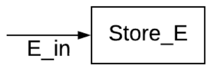
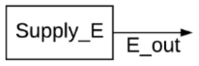
Figure 4: Entity Relation Attribute model for the vocabulary in ConMod Language

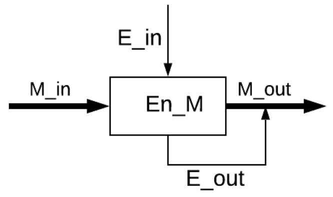
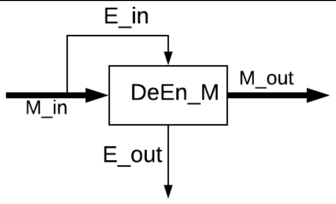
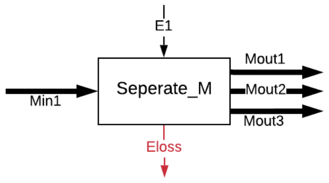
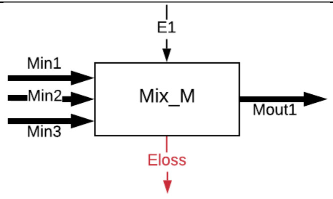
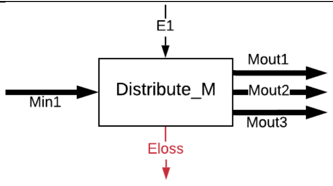
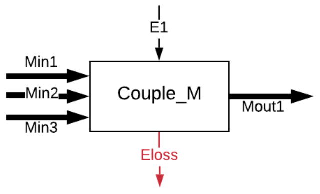
Layer 4 and Layer 5 are two extensions that enable automatic functional decompositions from given black box models. Layer 4 uses semantic reasoning and qualitative physics to determine modeling intent from the plain-English flow and function terms and computes the physical processes needed to decompose the black box [62]. Layer 5 uses evolutionary algorithms to generate fittest decompositions [59]. Finally, Layer 5 presents a formal representation of device states and modes using finite-state automata (FSAs), including a new vocabulary of three new terms: Actuate_E, Regulate_E_Discrete, and Regulate_E_Continuous and a vocabulary of conjugate functions. The vocabulary and grammar of each layer are formally defined in first-order logic and pseudocode, implemented as object-oriented classes within the ConMod program, and their reasoning efficacy is shown using ConMod in the references cited above. The last layer forms the discussions in this thesis.

3.2 Vocabulary of Functions of the ConMod Language

The current ConMod language comprises of a vocabulary of functions and function features under the verb class. In this context, function verbs are atomic functions that can be used in modeling, and they are formally defined to capture the semantics of their respective actions using their topology and grammar rules [74]. Table 3 lists all the vocabulary of function verbs that currently exist within this language.

Table 3: Vocabulary of Primary Functions

Function	Textual definition	Graphical Construct
TypeChange_E	To change the subtype of an energy flow [82]	
Transfer_E	To change the location of an energy flow in geometric space [82]	
Change_E	To change the quantitative parameters of an energy flow without changing its type [82]	
Store_E	To store an energy flow in a material medium where the medium behaves like a sink that can receive an infinite amount of the energy type [82]	
Supply_E	To obtain energy from a material medium, where the medium behaves like a source that can release an infinite amount of the energy type [82]	

Energize_M	To add an energy flow to a material flow [82]	
DeEnergize_M	To remove an energy flow from a material flow [82]	
Separate_M	To resolve a material flow that is a physical mixture of other material flows to its components (thereby causing a change of composition or type) [83]	 E1 Min1 Seperate_M Mout1 Mout2 Mout3 Eloss
Mix_M	To combine several material flows into a single material flow (thereby causing a change of composition or type) [83]	
Distribute_M	To distribute material flow into multiple flows without changing its type or composition [83]	 E1 Min1 Distribute_M Mout1 Mout2 Mout3 Eloss
Couple_M	To bring together several material flows such that the members are still distinguishable from each other [83]	

3.3 Finite State Automata-Based Descriptions in Function Modeling

In automata theory and formal linguistics [84], finite-state automata (FSAs) are an abstraction of artifacts that can exist in one of many states so that commands received by the machine could push it from one state to another. The possible number of states for a machine can be large but always finite. For example, a 1 kB computer memory has 1024 × 8 bits. Since each bit could be in one of two states $\{0, 1\}$, the chip, as a whole, has $2^{(1024 \times 8)}$ possible states – large but still finite. An FSA is defined as a quintuple $(\boldsymbol{\Sigma}, \boldsymbol{S}, \boldsymbol{s_0}, \boldsymbol{F}, \boldsymbol{\delta})$ where:

1. $\boldsymbol{\Sigma}$ is a finite set of input symbols (the alphabet of the FSA). The edges in state transition diagram are labelled using the input symbol to the state.

2. $\mathbf{S}$ is a finite set of states, they are represented in bubbles in the state transition diagram.

3. $\boldsymbol{s_0}$ is the initial state of the FSA, $\boldsymbol{s_0} \in \boldsymbol{S}$. The initial state is represented using a single arrow with no tail-node.

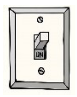
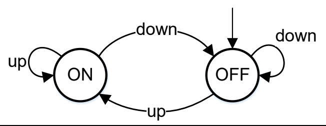
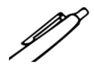
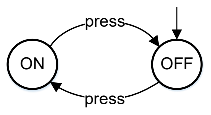
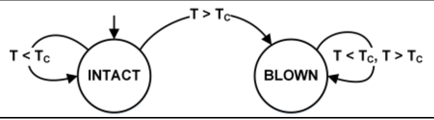
4. F is a set of final or accepting state, $\boldsymbol{F} \subseteq \boldsymbol{S}$, and they are usually represented in a bubble bordered with a concentric circle (All states are valid final states for the examples discussed in this research).

5. $\boldsymbol{\delta}: \boldsymbol{S} \times \boldsymbol{\Sigma} \longrightarrow \mathbf{S}$ is a state transition. In the state transition diagram, each edge where the tail- and head-nodes are states correspond to a state transition.

The definition of an FSA is adapted to suit the ConMod language for its software implementation (Chapter 8). FSAs will be used in this research as an input to functions upon their instantiation if the mode of operation of the function is dependent on the device's structural state. Contrarily, if functions have multiple topologies based on logical conditions placed on flow properties, FSAs will be embedded into the formal definition of the function.

We propose to describe the states and state-transitions of mechanical devices as FSAs. Table 4 shows the state models for three different devices. In each case, the circular bubbles describe states and the arrows describe the commands that takes the device from one state to another. For example, in the first figure, the bubbles indicate that the light switch can exist in either the ON or the OFF state. The labels on the arrows describe the commands received by this device: in this case, upward push or downward push. When the device is in the OFF state, a downward push keeps it in the OFF state but an upward push transitions the device to the ON state. While in the ON state, the opposite is true. Note that the ballpoint pen receives only one type of command, a press, and toggles between the two possible states in response to the same command, depending on the current state. The fuse remains intact if the temperature T is less than a critical value (T_C). If $T > T_C$, then the fuse transitions to the blown state and remains there forever: no value of temperature, however low, is able to restore the intact state. Thus, these three examples illustrate that different types of logical behavior of state-transitions can be captured by FSAs. In this research, the state-transition behavior of the devices is included as an integral part of the formal definition of the signal-processing verbs.

Table 4: Some discrete control devices and their state diagrams as FSAs

 Light switch	
 Ballpoint pen switch	
Electric fuse	

By extension of this representation, continuous devices can also be modeled, as shown in Figure 5. Here the bubbles show the terminal states such as the rheostat's control pushed to the extreme left or right, and the arrows L and R show the signal the device can accept at any state (slide left, slide right). Any interim state between the extremes can be shown as a function of the range. Applications of both the continuous and discrete state diagrams are shown in the next sections.

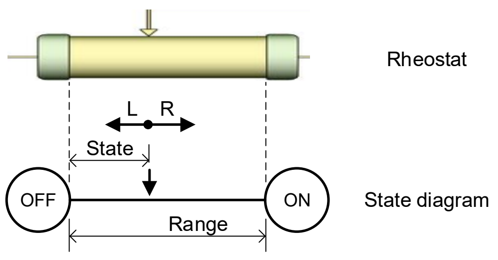


Figure 5: A continuous control device and its state diagram as FSA

Like devices, functions can also be represented using an FSA if the function performs more than one operation. Each operation performed by the function would correspond to a state of the FSA and the logical conditions determining the mode of operation will be the alphabet of the associated FSA. In the example of heat transfer to/from a thermal reservoir, the function executed by the system is to either store or supply thermal energy. The logical condition here will check if the temperature of a material flow entering the system has a temperature less/greater than that of the reservoir to determine if the system will supply/store thermal energy. Therefore, the FSA associated with the function Supply/Store will have two states: Supply and Store, and in each instance, it will have a different graphical construct and different flow topology. The alphabet of this FSA will include the two logical conditions: $T<T_{reservoir}$ and $T>T_{reservoir}$. The initial state of the FSA will be the initial modeling construct instantiated by the modeler and the final state would be determined by the state transitions governed by the logical conditions when the status signal (temperature) changes. This idea finds the basis of the formal definitions of the conjugate functions established in Chapter 6.

Chapter 4
Scope of Research

This chapter outlines the scope of the proposed formalism in addressing the research gaps outlined in Chapter 1. Section 4.1 discusses the scope of the signal processing verbs represented in the vocabulary of the extended ConMod language and section 4.2 details the scope of the vocabulary of conjugate verbs supported by the extension.

4.1 Scope of Representation for Signal-Processing Verbs

The signal-processing verbs in the Functional Basis are of two types: (1) verbs that operate on material or energy in response to signals such as Actuate and Regulate, and (2) verbs that receive or emit signals to communicate with the user such as Sense and Indicate. Of these, Actuate and Regulate are used by technical products significantly more frequently than Sense and Indicate. In a search through 130 products in the Design Repository, Actuate and Regulate were found 219 and 163 times, respectively, while Sense and Indicate occurred 65 and 24 times. Further, Actuate and Regulate initiate actions as a result of incoming signals, which more precisely aligns with the scope of this research. Thus, the current scope only includes the formalizing the definitions of Actuate and Regulate. The functional basis functions that contribute to this definition belong under the primary function of Control Magnitude and they are as follows:

Actuate: to commence the flow of energy, signal, or material in response to an imported control signal[20].

Regulate: to adjust the flow of energy, signal, or material in response to a control signal, such as a characteristic of a flow [20].

Stop: to cease, or prevent, the transfer of a flow (material, energy, signal) [20].

Change: to adjust the flow of energy, signal, or material in a predetermined and fixed manner [20].

Further, the Functional Basis definition of Stop shown in the footnote includes preventing an energy flow, which is logically opposite to Actuate as long as such prevention happens in response to an incoming signal. This special case of Stop is included in the present scope and is modeled as a logical negation of Actuate, as seen later.

As shown in Chapter 5, formalizing Regulate pre-requires formalizing the definition of “Change” from the Functional Basis. Change, as applied to energy flows, was previously formalized as Change_E [77]. It requires a change in at least one parameter of the energy flow while disallowing any change of energy type. This definition is used here as a basis of defining the change of energy flows in response to signals. Thus, to summarize, the present scope includes (1) Actuate and Regulate applied to energy flows, (2) the special case of Stop applied to energy flows in response to signals, and (3) the preexisting definition of Change_E.

Further, we note that modeling the actuation and regulation of energy flows is different from those verbs operating on material flows. Particularly, previous research in physics-based function modeling shows that any material-processing function needs energy exchange as a consequence of the second law of thermodynamics [30,39,85,86], which adds additional complexity to their definitions. Thus, the present scope only includes the actuation and regulation of energy flows, while such operations on material is saved for the future.

4.2 Scope of Representation for Conjugacy in Function Modeling

Conjugate functions will need to switch from one mode of operation to its reverse based on logical rules applied to attributes of flows. Computational reasoning based on these rules require the function model to be represented in a formal ontology to allow manipulation of relationships between classes (functions, flows, environments). For this reason, the scope of conjugate functions and features are restricted to the vocabulary of the ConMod language recounted in Chapter 3. The logical conditions causing mode reversals are explored in Chapter 6 alongside four formally defined conjugate verbs that use vocabulary established in this language.

Of the functions listed in Table 3: Vocabulary of Primary Functions in Chapter 3, eight of them exist in four pairs of opposite functions: Store_E/ Supply_E, Energize_M/ DeEnergize_M, Separate_M/ Mix_M, and Distribute_M/ Couple_M. These pairs of functions demonstrate conjugate behavior by operating in distinctly opposite modes; however, this paper will exclude the Separate_M/ Mix_M pair as the process of separation and mixing, unlike distributing and coupling, encompass various physical process which may or may not be reversible. For example, the physical process of mixing salt in water (dissolving) is very different from the process of separating salt from water (distillation) – and should not be represented using a singular composite function. Similarly, any mixture of two gasses is irreversible and its conjugate operation (separation of the gaseous mixture) is physically impossible in a single process without requiring additional energy. Therefore, the functions Separate_M and Mix_M will not be represented in a singular conjugate function.

Of the remaining functions, Transfer_E and Change_E allow the modeling of opposite modes as they make quantitative differences to an energy flow. For example, Transfer_E could be used to represent the transfer of ThE from a hot plate to a cold pan or vice-versa by changing the location of ThE, and Change_E could be used to capture both the increase and the decrease of angular velocity in an electric motor by changing the parameter torque associated with rotational mechanical energy. Both Transfer_E and Change_E require the attributes of a flow to be filliped and will be excluded from the proposed extension due to the varied nature of circumstances under which that occurs. TypeChange_E, a function used to capture energy conversions from one subtype to another does not support conjugate behavior and could benefit from a conjugate representation to capture reversible energy transformations. Based on the above discussions, Section 5 formally defines four conjugate functions: CEnergize_M, CStore_E, CDistribute_M, and CTypeChange_E. The addition of these four conjugate verbs will extend the vocabulary of the physics- and logic-wise consistent modeling language to support the conjugate behavior of its current vocabulary. The addition of the letter 'C' as a prefix indicates the conjugate behavior exhibited by the new functions.

4.3 Scope of Reasoning for the Formalism

Formal reasoning "is characterized by rules of logic and mathematics, with fixed and unchanging premises" [87]. It utilizes a set of rules (grammar) and symbols (vocabulary) within a representation to draw conclusions about the model that were not user inputs to the said model. For example, most CAD programs allow the user to create a shape by inputting geometric co-ordinates on a GUI with respect to a fixed co-ordinate system through mouse clicks, and the program is capable of producing information such as the shape's volume, surface area, geometric center, etc. The logical deductions necessary to derive the conclusions on the shape's geometric properties from its description through the use of algorithmic rules is described as formal reasoning. In this research, the model description is captured in the form of a function model using the modeling entities (functions, flows, and environment) and the grammar rules established in Layer 1 of the ConMod language. This section identifies the reasoning tasks that the representation must accomplish to support the modeling of multiple modes in a function model based on the representational requirements stipulated in Section 4.1 and 4.2.

1. Check the validity of the FSA description: Verbs that perform multiple operations are controlled either by an FSA input to the verb (Actuate_E, Regulate_E_Discrete, CStore_E, CTypeChange_E, and CConvergize_EM) or through state transition logic built into the formal definitions of verb (CEnergize_M, CDistribute_M, and CHandover_E). In the former case, the user entered FSA must be valid. This is ensured by checking that the input FSA

description has a correct initial state, state names, command signals (alphabet), and is complete. The algorithm to ensure this is described in detail in Section 8.4.1.

2. Determine the state of the function: The verb vocabulary introduced in this research operates in multiple modes as determined by the state of the function. Actuate_E commences and stops a flow, CStore_E stores and supplies an energy flow, and CEnergize_M adds and removes energy from a material flow. This is achieved through logical statements built into the formal definitions of the functions as described in Chapter 5 through Chapter 7. The vocabulary consists of verbs existing in two or more states, logical statements govern the transitions between the states, and the control signals constitute the alphabet to the FSA. This is implemented algorithmically into the ConMod software for the Actuate_E function in Section 8.4.2.

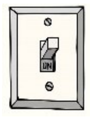
3. Determine the operating mode of the function model: Function models constitute multiple functions connected with flows. As a consequence, when a flow changes its state, the function that is the headnode of the flow must also be affected by the change. The functions defined within this representation have multiple possible flows as an output – of which the active flow is determined by the state of the function. The active mode is determined by checking the head-node relations of the flows that are an output from these functions. This is done algorithmically in Section 8.5 employing the idea of hidden and active flows.

The above reasoning tasks are obtained from the definitions of signal processing verbs and conjugate verbs outlined in Section 4.1 and 4.2 and the reasoning tasks are implemenented in Chapter 8 and demonstrated in Chapter 9 and Chapter 10.

Chapter 5
Representation: Formal Definitions of Signal Processing Verbs

Recent research on Signal flows propose that Actuate is “to discretely toggle a flow” and an actuator is “a discrete control device used to turn another flow on or off” [88]. It also says that Regulate is to “adjust a flow quantity in an analog manner” [88]. While these definitions provide for both toggling and controlling flow quantities, they do not include the situation where flow quantities are adjusted discretely, without turning them off. Consequently, a survey of the Design Repository revealed the need for supporting four distinct modeling scenarios related to the actuation and regulation of energy (Table 5). Note that the second scenario is hypothetical because an example for it was not readily found but such a device is conceivable. The requirement on the language is that it support the other three, non-hypothetical scenarios. Consequently, three verbs are identified in the last column of this table. These verbs are formally defined in the next section.

Table 5: Modeling scenarios required to be supported

Device type	Example device	Verb Needed
Devices that actuate (on) or stop (off) energy in discrete steps but do not regulate it in between the extreme states	 Light switch	Actuate_E
Devices that regulate energy but do not actuate or stop it	Hypothetical case included to ensure coverage. Survey in the Design Repository did not find any clear example.	

Device type	Example device	Verb Needed
Devices that actuate (on) or stop (off) energy in discrete steps and regulate it in between in discrete steps	Rotary switch with multiple positions including zero	Regulate_E_Discrete
Devices that actuate (on) or stop (off) energy in discrete steps and regulate it in between continuously	Rheostat with the ability to connect/disconnect EE	Regulate_E_Continuous

The three verbs identified above are defined in this section. Each subsection below presents a verb's formal definition, graphical template, and modeling examples using devices from Table 5. In sections 5.1 – 5.3, these verbs are used to construct system-level models and to illustrate their reasoning capability.

5.1 Actuate_E

Actuate_E is a verb that commences or stops an energy flow in response to an incoming control signal. In doing so, it combines the effects of Actuate and Stop within the Functional Basis. Figure 3 and Table 4 show the graphical representation and formal definition of this verb, respectively.

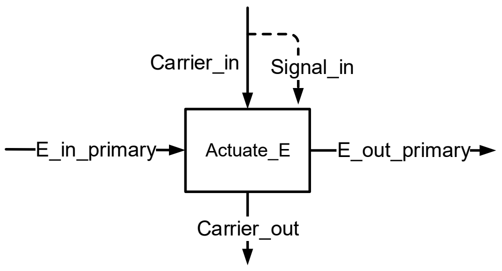


Figure 6: Formal graphical representation of Actuate_E

These formal definitions are written and are best understood in the context of prior research in formalizing function verbs [73]. For example, the first line in Table 6 states that Actuate_E is derived from the class Verb defined in [73] and therefore inherits all its properties. Therefore, statements within the class Verb are not repeated here; only the statements specific to Actuate_E are presented. In summary, the verb has three input and two output flows. E_in_primary is the energy flow being actuated or stopped, and E_out_primary is its manifestation on the output side. Signal_in is a signal flow carried by Carrier_in_primary, which can be a material or an energy and leaves the device as Carrier_out_primary. Signal_in controls the verb's action based on the state transition grammar mentioned in the last section of Table 6, which says that there are only two possible states (State_1, State_2) and multiple possible values of the input signal. The last sentence of Table 6 says that if the state of the device is State_1, then E_out_primary has the same type as E_in_primary, which means that the incoming energy flow is present at the output. Otherwise, the E_out_primary's type is set to null, which means that the flow ceases to exist, thus simulating Stop.

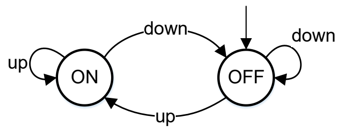
Table 6: Formal definition of Actuate_E

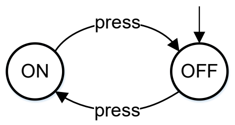
```
Class Actuate_E : Verb // Inherited from Verb
{
  // Flow counts in input and output
   In_List   = {E_in_primary, Carrier_in_primary, Signal_in};
   Out_List  = {E_out_primary, Carrier_out_primary};

  // Flow types in input and output
  E_in_primary ∈ E;
  Carrier_in_primary ∈ (M ∪ E);
  Signal_in ∈ Control_Signal;
  E_out_primary ∈ E;
  Carrier_out_primary ∈ (M ∪ E);

  // Flow constraints
  Carrier_out_primary.Subtype = Carrier_in_primary.Subtype;
  Signal_in.Carrier = Carrier_in_primary;

  // State transition grammar
  States = { State_1, State_2 };
  Signal_in.Values = { a, b, c, … };

  E_out_primary.Subtype  =
  {
    if (state == State_1)
      then (E_in_primary.Subtype)
      else (ϕ);   // ϕ = null
  };
}
```

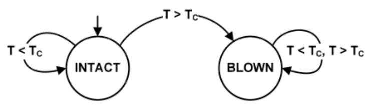
Note that the state transition grammar allows exactly two states but unlimited signal values and does not enforce any state change rule within the definition. This design is in recognition of the fact that different two-state devices can have quite different grammars, as illustrated in Table 4, yet we want the definition to be widely applicable to two-state devices. Accordingly, the designer must declare the state transition rules using an FSA for the specific device. Table 7 shows the modeling declarations required to emulate the FSAs of Table 4. Note that each FSA in this table represents not just a device but a class of devices. For example, what the light switch does to EE is the same as what a mechanical clutch does to ME, namely, actuating or stopping the flow. So, the light switch model could be used for a clutch. Likewise, the ballpoint pen represents all two-state toggle operations, while the fuse describes all irreversible processes. In this manner, the formal definitions produce the ability to model the change in the device's states under various commands by formalizing the state transition grammar and including that in the definition of the verb.

Table 7: Modeling with Actuate_E

Device class	Model declarations


Light switch
	`State_1 = ON;`
	`State_2 = OFF;`
	`signal_in.values = {up, down};`
	`ON.up = ON;`
	`ON.down = OFF;`
	`OFF.up = ON;`
	`OFF.down = OFF;`


Ballpoint pen switch
	`State_1 = ON;`
	`State_2 = OFF;`
	`signal_in.values = {press};`
	`ON.press = OFF;`
	`OFF.press = ON;`

 $T > T_C$
$T < T_C$
INTACT
BLOWN
$T < T_C$, $T > T_C$
Electric fuse
	`State_1 = INTACT;`
	`State_2 = BLOWN;`
	`signal_in.values = {<, >};`
	`INTACT.< = INTACT;`
	`INTACT.> = BLOWN;`
	`BLOWN.< = BLOWN;`
	`BLOWN.> = BLOWN;`

5.2 Regulate_E_Discrete

Regulate_E_Discrete represents devices similar to the rotary switch in Table 5, i.e., this verb must support all actions of Actuate_E from the previous section, plus the regulation of the primary flow in a discrete manner. Therefore, the definition is inherited from Actuate_E, while it is distinct from Actuate_E in the state transition grammar, which is overridden in Table 8. Regulate_E_Discrete has the same graphical template as that of Actuate_E (Figure 7). Further, regulating the energy flow necessarily implies that at least one parameter of the output primary flow changes at output. This behavior is already captured by a pre-existing verb, Change_E [77]. Thus, Regulate_E_Discrete is also inherited from Change_E.

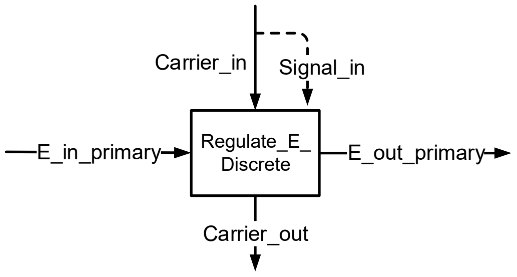


Figure 7: Graphical representation of Regulate_E_Discrete

This definition differs from Actuate_E in three ways, which can be verified in Table 8: (1) the device has a total n+1 states, (2) the output flow vanishes in State_0 and exists in all other states, and (3) there is a parameter of the primary energy flow that changes value between the input and output.

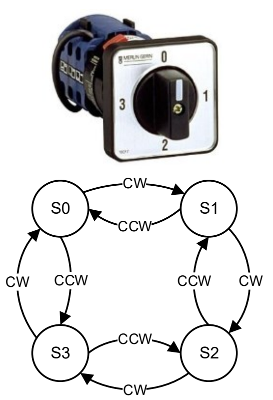
Table 8: Formal definition of Regulate_E_Discrete

```
Class Regulate_E_Discrete : Actuate_E, Change_E
  // Inherited from Actuate_E and Change_E
{
  // Override - State transition grammar
  States = { State_0, State_1, … , State_n };
  Signal_in.Values = {a, b, c, …};

  E_out_primary.Subtype  =
  {
    if (state == State_0)
       E_out_primary.Subtype  =  ϕ;
    else
       E_out_primary.Subtype  =  E_in_primary.Subtype);
  };

  ∃ parameter → E_in_primary.parameter.value ≠
  E_out_primary.parameter.value;
}
```

Table 9 shows a modeling application of this verb for the rotary switch introduced earlier. As before, the flexibility of the verb's definition allows the modeler to represent the FSA-like behavior of the device during modeling, by assigning values to the states S0 through S3, the commands for clockwise (CW) or counterclockwise (CCW) rotation of the knob, and the eight state transition rules shown in Table 9.

Table 9: Modeling with Regulate_E_Discrete

Device class	Model declarations

 S0
S1
S2
S3
CW
CCW
CW
CCW
CCW
CW
CCW
CW
Rotary switch
	`State_0 = S0;`
	`State_1 = S1;`
	`State_2 = S2;`
	`State_3 = S3`
	`signal_in.values = {CW, CCW};`
	`// state transition rules`
	`S0.CW = S1;`
	`S1.CW = S2;`
	`S2.CW = S3;`
	`S3.CW = S0;`
	`S0.CCW = S3;`
	`S1.CCW = S0;`
	`S2.CCW = S1;`
	`S3.CCW = S2;`

5.3 Regulate_E_Continuous

The objective of this verb is to describe devices that regulate the incoming energy flow in a continuous manner, such as the rheostat shown in Figure 5, while all other actions of the verb are similar to the previous verb, Regulate_E_Discrete. Thus, this verb too is inherited from Actuate_E and Change_E, with the state transition grammar overridden for its specific purpose. Figure 8 shows the graphical template for this verb, while Table 10 shows the formal definition.

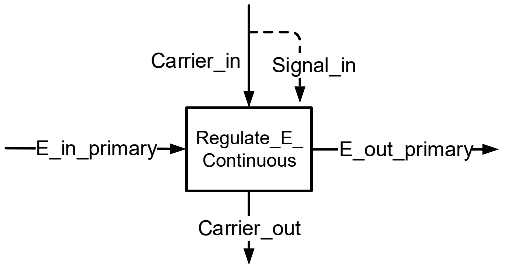


Figure 8: Graphical template ofc Regulate_E_Continuous

Table 10: Formal definition of Regulate_E_Continuous

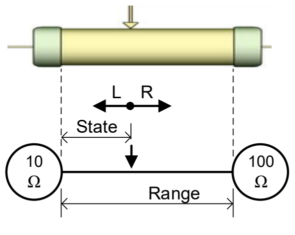
```
Class Regulate_E_Continuous : Actuate_E, Change_E
  // Inherited from Actuate_E and Change_E
{
  // Override - State transition grammar
  ∃state_0, state_1;
  range = (state_1 - state_0);
  ∃state → state_0 ≤ state ≤ state_1;

  Signal_in.Value;  // command

  state = state + Signal_in.Value;
  state_ratio = ((state - state_0) / range);

  E_out_primary.Subtype  =
  {
    if (state == State_0)
      then (ϕ)
      else (E_in_primary.Subtype);
  };

  ∃ parameter → E_out_primary.parameter.value =    f(state_ratio) *
  E_in_primary.parameter.value;
}
```

In order to support continuous variation of states, this definition includes two terminal states: state_0 and state_1. The current state can be any value within this range. The commands carried by the signal are numeric values that can be added to the current state to produce the new state. State ratio captures the proportion of the current state over the range. When the current state is the left terminal (state_0), the output flow ceases to exist (subtype = ϕ). In all other cases, the flow exists in the same type as the input flow, and there is at least one parameter that is computed as a function of the input parameter and the state ratio. For rheostats, this flow parameter is the current, which depends on the device's state (distance of the slider from the left end). For rotary flow control valves in water lines, this flow parameter is the discharge rate, and it depends on the device's state (angle of rotation of the wheel).

Table 11 shows an application of this definition for a linear rheostat that varies between 10 Ω and 100 Ω. The terminal states assume these two values. Once a signal is received, the new state is computed, and the parameter of concern is computed from the state ratio as a linear function of the range. Variations of this linear controller can be obtained by assigning different functions to the variable "`f`", which captures the mathematical relation between the input and output parameters.

Table 11: Modeling with Regulate_E_Continuous

Device class	Model declarations


Linear rheostat
	State_0 = 10 Ω;
	State_1 = 100 Ω;
	`State = 30 Ω;`
	signal_in.value = +5 Ω;
	`E_in_primary.parameter = current;`
	`f = "linear";`

Chapter 6
Representation: Formal Definitions of Conjugate Verbs

6.1 CEnergize_M

CEnergize_M is a new verb that combines Energize_M (adding energy to a material flow) and DeEnergize_M (removing energy from a material flow) defined in [82], with an ability to toggle between the two based on some threshold parameters. Table 12 illustrates a few devices that combine these conjugate actions. In the heat exchanger, the temperature difference between the fluids in the drum and the tube determines which fluid will be energized and which one deenergized. In the pump-turbine, the water is energized in the pump mode when the impeller adds mechanical energy to it and is deenergized in the turbine mode when it transfers energy to the impeller. The transition between these modes can be caused by several parameters attached to the energy flows, i.e., status signals, some of which are listed in the table.

Table 12: Examples illustrating need for CEnergize_M

Device	Energy type	Threshold parameter
Drum & tube heat exchanger	ThE	Temperature difference between the fluids
Pump-turbine	ME	Direction of water flow, pressure gradient or head difference between inlet and discharge, or direction of torsional deflection of shaft

Figure 9 (a) and (b) show the graphical templates of Energize_M and DeEnergize_M in [82], from which the template of CEnergize_M is derived in Figure 9 (c). Figure 9 (d) shows the FSA of that conjugate verb. Table 13 shows the formal definition of CEnergize_M in pseudocode form.

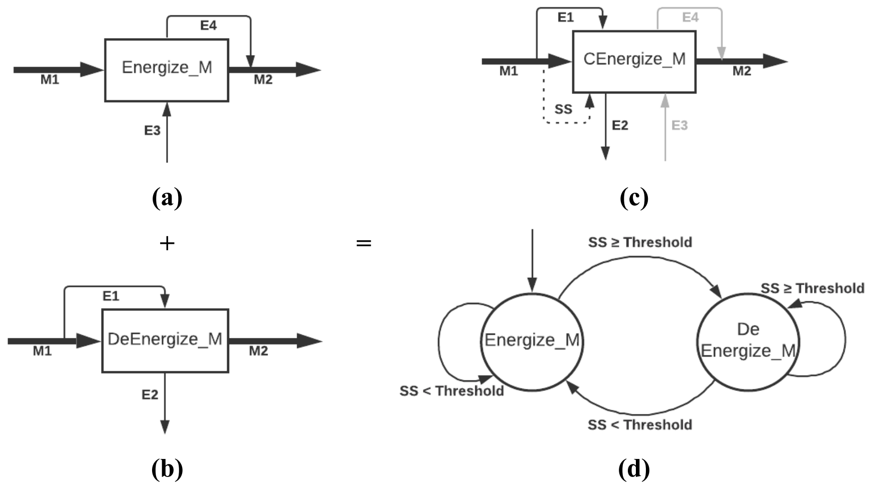


Figure 9: Templates and FSA of CEnergize_M

Table 13: Formal Definition of CEnergize_M

```
Class CEnergize_M : Verb, Energize_M, DeEnergize_M
// inheriting from higher-level classes
{
// Obtain parameter value from material flow
Status_Signal = M_in.Energy.Parameter.Value;

// Determine state and mode
if (Status_Signal ≥ Threshold)
   {
      Verb.State = DeEnergize_M;
      E1.type ∈ E;
      E2.type = E1.type;
      E3.type = ϕ;
      E4.type = ϕ;
   }
else
   {
      Verb.State = Energize_M;
      E1.type = ϕ;
      E2.type = ϕ;
      E3.type ∈ E;
      E4.type = E3.type;
   }
}
```

Per [82], Energize_M receives an energy flow and a material flow at the input and adds the former as a carried flow of the latter at output. DeEnergize_M does the opposite: it receives a material flow carrying an energy flow and outputs the two flows separately. In addition, Layer 1 of ConMod describes the base class Verb, from which all verbs are inherited. The formal definition of CEnergize_M (Table 13) therefore inherits from these higher-level definitions, so that their internal details need not be repeated in Table 13. The new information in CEnergize_M is the rules of mode transition. To this end, the FSA in Figure 9(d) shows that any time the status signal value is higher than a threshold, the mode of the function is DeEnergize_M, otherwise Energize_M. The formal definition captures these rules by switching the types of the incoming and outgoing energies between null (ϕ) and the user-defined energy type (see Table 13).

6.2 CStore_E

CStore_E combines the conjugate verbs Store_E and Supply_E defined in [82]. They operate only on energy flows, per definition. Examples of devices using this conjugacy are shown in Table 14 to justify its inclusion. The last column of this table lists the control signal parameter that drives the mode of the function. A spring, for example, will store or supply energy based on presence of an external load, and an engine flywheel will do so based on the crank angle (Store ME during power stroke, supply ME during other strokes). In each case, a material or an energy flow carries the control signal that determines the mode.

Table 14: Examples illustrating the need for CStore_E

Device	Energy type	Control Signal Type
Spring	PE	Force Vector, carried by an external material flow
Flywheel	ME	Crank Shaft Angle, carried by the rotational mechanical energy transferred from the crankshaft
Capacitor	EE	Direction of Current, carried by the electrical energy transferred from a battery source, or drawn from a load

Figure 10, which is formatted similar to Figure 9, shows the templates of Store_E and Supply_E from [82], and the template and FSA of the conjugate verb. Table 15 shows the formal definition of CStore_E. In this table, as with the previous case, the conjugate verb is inherited from previously defined abstract classes. δ is the FSA transition function that determines the resulting state upon receiving a signal. Based on this function, either the incoming energy is turned on and the outgoing energy is turned off (in the Store mode), or vice-versa (in the Supply mode).

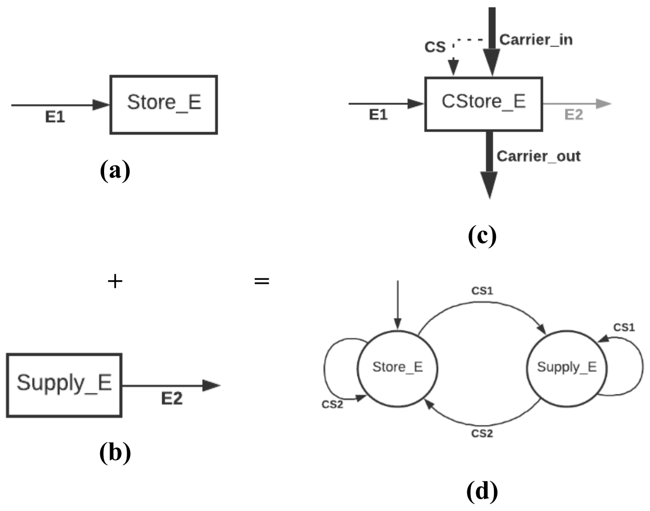


Figure 10: Templates and FSA of CStore_E

Table 15: Formal Definition of CStore_E

```
Class CStore_E : Verb, Store_E, Supply_E
// inheriting from higher-level classes
{
// Input and output flow types
   In_List = {E1, Carrier_in, CS};
   Out_List = {E2, Carrier_out};
   E1 ∈ E; E2 ∈ E;
   Carrier_in ∈ (M U E);
   Carrier_out ∈ (M U E);

// Grammar rules on Carrier Flow
   CS.Carrier = Carrier_in;
   Carrier_out.Type = Carrier_in.Type;

// Determine state and mode
   States_List = {Store_E, Supply_E};
   Signal_in_List = {CS1, CS2};

   if (𝜹(State, CS) == Store_E)
     Verb.State = Store_E;
     E2 = ϕ;
   else
     Verb.State = Supply_E;
     E1 = ϕ;
}
```

6.3 CDistribute_M

CDistribute_M combines the conjugate verbs Distribute_M and Couple_M defined in [83] and allows logical switching between these modes based on the number of material flow inputs. As defined, Distribute_M receives one material flow and breaks it into multiple identical flows, while Couple_M does the opposite. Figure 11 shows a pipe junction that could either couple or distribute the flows based on pressure gradients between the joint and the other end of each pipe.

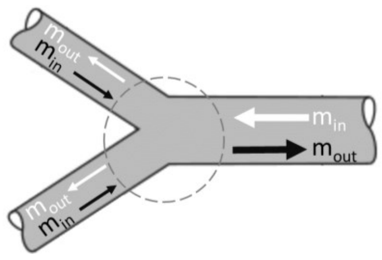


Figure 11: Possible inlet and outlet flow directions through a junction

As before, Figure 12 shows the templates of Distribute_M and Couple_M from [83], and the formal definition and FSA for CDistribute_M. Table 16 shows the formal definition of CDistribute_M. The modes can switch due to a status signal such as pressure differences or a control signal such as user commands. To keep these options open, the trigger is modeled based on the number of input material flows, n. When n = 1, the function's mode is Distribute_M and the number of output flows, m > 1. When n > 1, the mode is Couple_M, and m = 1. In all other cases, the function is not in either of these modes, and an error is returned.

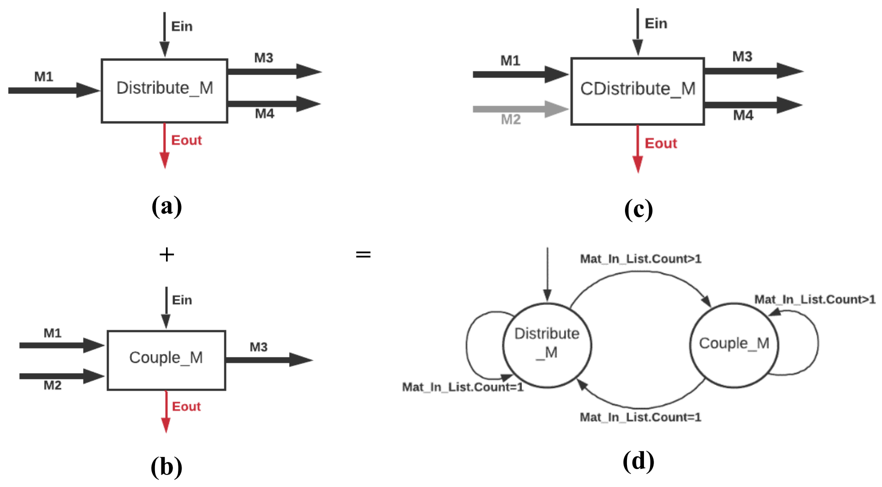


Figure 12: Templates and FSA of CDistribute_M

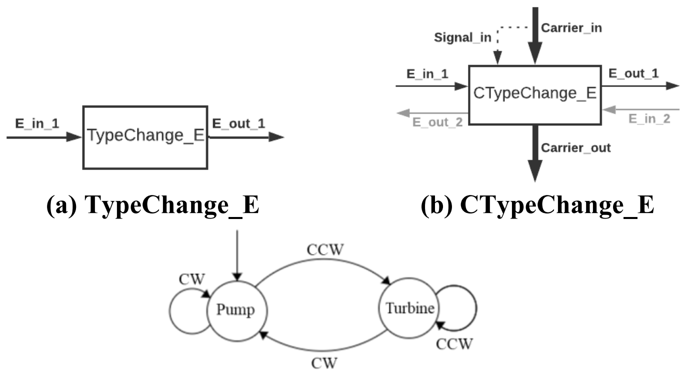
Table 16: Formal Definition of CDistribute_M

```
Class CDistribute_M: Verb, Distribute_M, Couple_M
{
//Flow Counts
n ∈ℤ+;     // Number of branches
Mat_In_List = {M_in1,…,M_inn};
Mat_Out_List = {M_out1,…,M_outm};
En_In_List = {E_in};
En_Out_List = {E_out};

//Determine state and mode
Loop through Mat_In_List
{
      if (n == 1)
            Mode = Distribute_M;
            Loop_through (Mat_In_List, i > 1)
            Mat_In_i.type = ϕ;
            m > 1;

      else if (n > 1)
            Mode = Couple_M;
            m = 1;
            Loop_through(Mat_Out_List, i > 1)
            Mat_out_i.type = ϕ;

      else ERROR;
}
```

6.4 CTypeChange_E

CTypeChange_E is the conjugate version of TypeChange_E described in [82]. Unlike the previous examples, CTypeChange_E does not combine two different verbs but describes a single function that can assume two opposing modes. Examples include devices such as the DC motor-generator and the pump-turbine shown in Figure 2 and rechargeable batteries that change energy types from electrical to chemical, or backward, depending on its mode: charging or discharging. Figure 13 shows the templates for the original verb

(TypeChange_E), the conjugate verb (CTypeChange_E), and the FSA. Table 17 shows the formal definition. The mode transition happens due to a user command, shown as Signal_in, following the FSA. The FSA in this example is stay in the pump mode if the user command is clockwise (CW), and in the turbine mode if it is counterclockwise (CCW). These keywords can be edited in the FSA template provided in ConMod, if necessary.



(a) TypeChange_E **(b) CTypeChange_E**

(c) FSA

Figure 13: Templates and FSA of CTypeChange_E

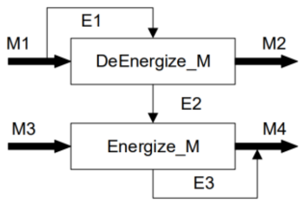
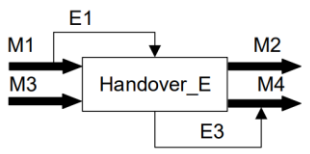
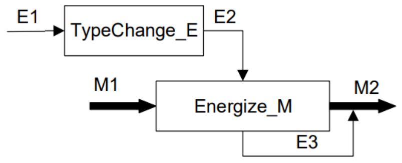
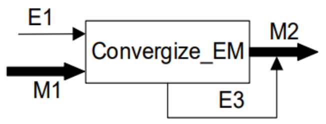
Table 17: Formal Definition of CTypeChange_E

```
Class CTypeChange_E : Verb, TypeChange_E
{
// Flow counts in input and output
  In_List   ={E_in_1, E_in_2 Carrier_in, Signal_in};
  Out_List  ={E_out_1, E_out_2, Carrier_out };

// Flow types in input and output
      E_in_1 ∈ E; E_out_1 ∈ E;
      E_in_2.subtype = E_out_1.subtype ∈ E;
      E_out_2.subtype = E_in_1.subtype ∈ E;
      Carrier_in.subtype ∈ (M ∪ E);
      Signal_in ∈ Control_Signal;

// Carrier Flow constraints
      Carrier_out.Subtype = Carrier_in.Subtype;
      Signal_in.Carrier = Carrier_in;

// State transition grammar
  State_List = { State_1, State_2 };
  State ∈State_List;

  if (State.Signal_in == State_1)
  {
      E_in_1 ∈ E; E_out_1 ∈ E;
      E_in_2 = ϕ; E_out_2 = ϕ;
  }
  else
  {
      E_in_1 = ϕ; E_out_1 = ϕ;
      E_in_2 ∈ E; E_out_2 ∈ E;
  }
}
```

Chapter 7
Representation: Extension of Conjugacy to Function Features

Function features are modeling entities that encapsulate multiple atomic functions to represent a complex concept. Due to the repeated use of certain clusters of basic functions within multiple models, function features were introduced to save modeling time and effort by combining the grammars of its constituent functions [81]. Recent research in this area have established a comprehensive feature set based on the physics- and logic- wise consistent language mentioned earlier for modeling thermal-fluid systems. A few of those function features are shown in Table 18.

Table 18 Examples of Function Features in Current Language

Expanded Version	Contracted Version
 E1 M1 DeEnergize_M M2 E2 M3 Energize_M M4 E3	
Handover_E [81]	
	
Convergize_EM [81]	

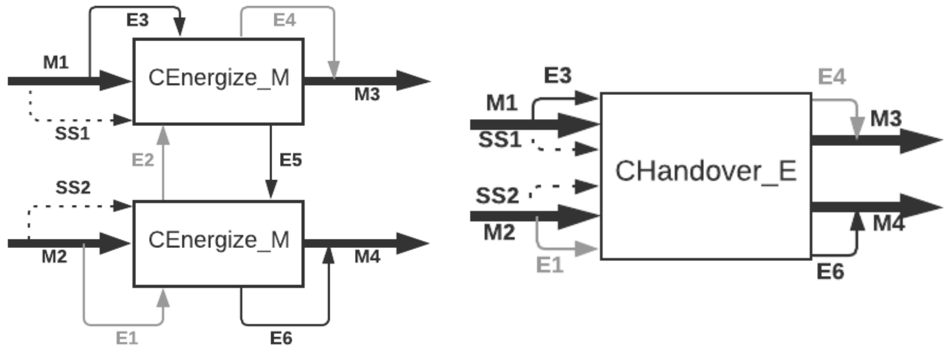
Chapter 6 took the vocabulary from this language to demonstrate that they either already support the modeling of conjugate behavior or could be combined/ altered to support modeling conjugate behavior. The idea of conjugacy could therefore be extended onto function features listed in Table 18 to support feature level conjugate behavior. Take the feature Change_M for example. Change_M is composed of three primary functions: DeEnergize_M, TypeChange_E, and Energize_M. It is used to model scenarios such as the flow of steam through a nozzle, free fall, and work done by a pump in raising a fluid against gravity, where the energy carried by a material flow changes its type. In the example of a fluid flow through a nozzle the thermal energy of the fluid (stored as enthalpy) gets converted into kinetic energy or vice versa based on the direction of fluid flow. Therefore, by modeling the Change_M feature with the conjugate versions of the functions that constitute Change_M, the function will be able to represent both modes and allow for system-level changes to the mode based on a control signal input.

Since features are combinations of two or more primary functions, this paper will not divulge into a comprehensive list of conjugate features. Instead, an approach to developing conjugate features is presented in sections 7.1 and 7.2 through the example of two conjugate features: CHandover_E and CConvergize_EM. CHandover_E utilizes a status signal to determine the mode of the feature and CConvergize_EM does so based on a control signal input.

7.1 CHandover_E

The *Handover_E* feature is pre-defined as a combination of two primitives *DeEnergize_M* and *Energize_M*. The feature represents energy transfer between two flows where one flow is de-energized, and the other flow is energized using the energy from the first flow. Previous research has already established the topological grammar rules for *Handover_E* and discussed its application in energy transfer devices such as heat exchangers [81]. However, its physics-wise consistency was not rigorously defined as *Handover_E*'s current formal definition can be erroneously used in modeling to allow energy transfer from low energy potentials to higher energy potentials. For example, a fluid at a lower temperature can be presently modeled to be de-energized and energize a fluid of a higher temperature. The second law of thermodynamics prohibits the transfer of energy against the temperature gradient without the addition of work. Since the feature is used to model energy transfer along a positive energy transfer gradient (the grammar restricts “work” input), the energy transfer should be automatically reversed based on the flow attributes. Thus, both the material flows into the feature should be allowed to either be energized or de-energized depending on the parameter values of equivalent energy types – which enter the function as status signals. *CHandover_E* could be defined in terms of CEnergize_M conjugate pair to (1) Allow physics-wise consistent modeling of energy transfer, and (2) represent energy transfer between the two material flows as a result of a change in operating mode.

Energize_M and *DeEnergize_M* constrain the input and output energies to be of the same subtypes, thereby constraining *CHandover_E*(read: Conjugate Handover Energy) to contain one energy type. Each energy type will have a parameter list which will have values and units associated with them. *Handover_E* will have to scan the values associated with energy attributes for both input material flows and de-energize the material flow with higher values of energy attributes and energize the material flow with lower values of energy attributes. Using this understanding, *CHandover_E* is formally defined in Table 19.

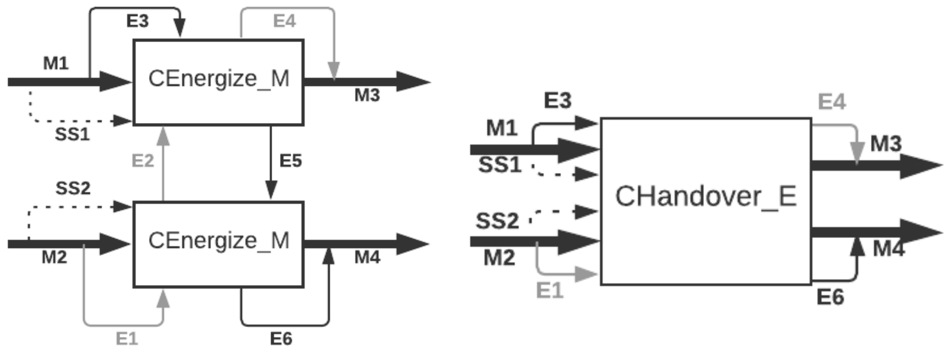


(a) Subgraph view **(b) collapsed view**

Figure 14: Templates for CHandover_E

Table 19: Formal definition of CHandover_E

```
Class CHandover_E: CEnergize_M;
//Determine Status Signals
SS1 = M1.Energy.Parameter.value;
SS2 = M2.Energy.Parameter.value;

//Determine state and mode
if (M1.Energy.Parameter.value > M2.Energy.Parameter.value)
{
      M1.HeadNode = Energize_M;
      M2.HeadNode = DeEnergize_M;
      E1 = E2  = E4 = ϕ;
}
else
{
      M1.HeadNode = DeEnergize_M;
      M2.HeadNode = Energize_M;
      E3 = E5  = E6 = ϕ;
}
```



(a) Subgraph view **(b) collapsed view**

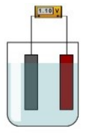
Figure 14 shows the expanded model for the *CHandover_E* with two operating modes alongside its graphical template. The active mode is shown in black and the inactive mode, its topological opposite, is shown in gray.

7.2 CConvergize_EM

The Convergize_EM (read Conjugate Convergize Energy Material) feature is defined using the primary functions TypeChange_E and Energize_M and it is used to model scenarios where an energy flow is converted from one type to another and then added onto a material flow. Example devices that represent this are shown in Table 20. Since Convergize_EM uses functions that cannot represent dual modal systems, all the examples in Table 20 can only be modeled in a single mode when they can operate in two distinctly opposite modes. A thermoelectric cooler can be used to generate electricity if a temperature difference (ThE) is an input to the system, a pump can be used to generate electricity (turbine mode) if the direction of angular rotation is reversed and electrochemical cells can produce electricity when they are used as batteries. Therefore, by introducing the conjugate function CConvergize_EM, the current language can be extended to support modeling and reasoning on dual mode systems. It should be noted that CConvergize_EM is not meant to replace Convergize_EM as there are devices that operate in a strictly singular mode and are physically impossible to be reversed. In a disc brake, for example, the rotational mechanical energy is converted into thermal energy (heat dissipated due to friction), and this process is irreversible. Therefore, the function CConvergize_EM should not be used to model this scenario as it is physically impossible to reverse heat dissipation caused by friction.

Table 20 Devices represented by Convergize_EM

From E Type	To E Type	Add to M	Device/ Process
EE	ThE	Air	Thermoelectric Cooler

EE	ME	Water	Pump
EE	ChE	Electrolyte	 Electrochemical Cells

CConvergize_EM will follow the template of Convergize_EM where TypeChange_E is followed by Energize_M, except their constituent functions will be replaced by their conjugate counterparts. It is important to recognize that this definition will not use the status signal input to CEnergize_M, instead the control signal input to CTypeChange_E will be used to determine the mode of the function. Control signals are more meaningful when modeling systems that CConvergize_EM represents as the change in mode of the feature is a result of the user physically altering the structure of the device. The information carried by the material flow (status signal input to CEnergize_M) is a result of the change in the structural state of the device. CConvergize_EM can therefore only exist in two distinct states: Change the type of an energy flow and add that energy to a material flow (State 1), or extract an energy flow from a material flow and then change its type (State 1). If CTypeChange_E is in State_1, then the initial modeling construct is instantiated. The grammar for this construct is similar to that of Convergize_EM and it ensures that the output flow from CTypeChange_E is the input energy flow into Energize_M (active mode in Figure 16). If the state variable equals State_2 in CTypeChange_E, then the output energy flow from the DeEnergize_M is the input to the CTypeChange_E function (greyed, inactive, mode in Figure 16). The graphical template for this feature is given in Figure 16 along with the decomposed version in Figure 15.

Table 21 Formal Definition of CConvergize_EM

```
Class CConvergize_EM: CTypeChange_E, Energize_M,
DeEnergize_M.

if (CTypeChange_E. State = State_1)
    E2.HeadNode.State = Energize_M;
    E4 = E5 = E6 = ϕ;
else
    E4.HeadNode.State = DeEnergize_M;
    E1 = E2 = E3 = ϕ;
```

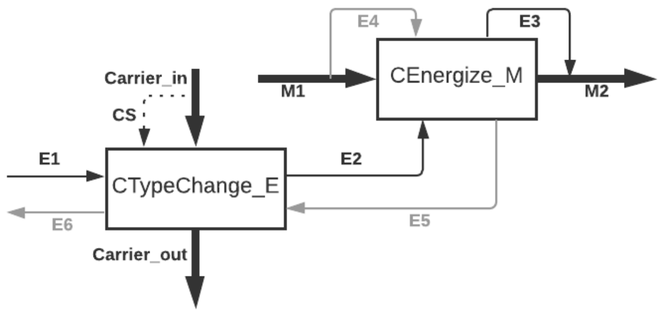


Figure 15: Expanded representation of CConvergize_EM

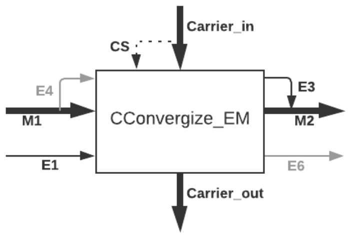


Figure 16: Contracted Representation of CConvergize_EM

Chapter 8
Reasoning: Implementation of FSA based reasoning to ConMod

In this section, the three new function verbs and the FSA-based capturing of states are implemented in software, and they are used to validate the representation's ability to model multi-state and multi-mode devices and reason with that additional information – which was the research gap mentioned in Chapter 1. For this purpose, the Actuate_E verb is implemented onto ConMod, the concept modeling software for the language discussed in Chapter 3. Implementing this new verb on an existing platform demonstrates the computational realization of the verbs and the FSAs and shows the extendibility of the platform.

8.1 Implementation of Actuate_E

ConMod is developed on the C++ language and already has a class hierarchy that includes base classes such as Verb, from which Actuate_E was derived (see Table 6). It also has the capability to construct function models with blocks, material, energy, and signal flows defined with plain-English names (top pallet in the figure) or functional features such as Convert_E and Energize_M (right pallet). These lower-level classes and features are described in prior research [89]. ConMod supports function modeling with physics- and logic-wise consistent reasoning, in both qualitative and quantitative manners (left panel of the figure) [30]. The implementation of Actuate_E on ConMod involves writing a class including associated dialog functions and graphic objects that embody the pseudocode of Table 6. Figure 17 shows the dialogs and an instance of Actuate_E on the modeling space of ConMod. The file selection dialog is used to select a plain-text description of the FSA (see Section 8.3), the other dialog is used to declare the type of carrier flow for the incoming signal, and the graphics in the modeling space shows the instantiation of the Actuate_E class in the model. The icon for Actuate_E is seen in the bottom-right corner of the figure. The remainder of this section illustrates the reasoning needed to perform causal deductions on flow actuation and stoppage on single function blocks and system-level models.

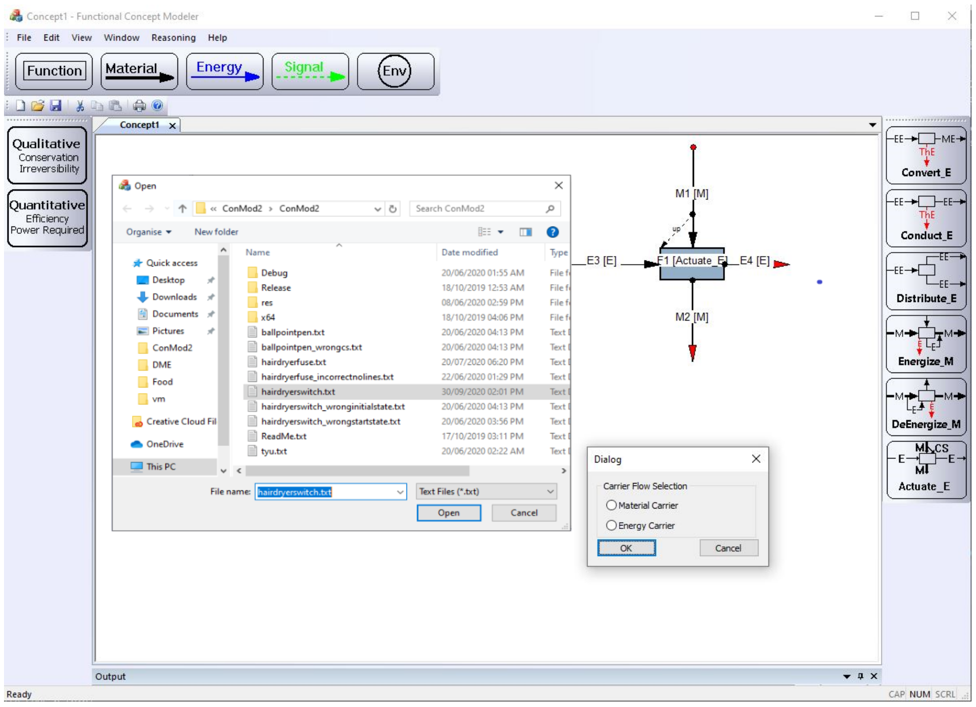


Figure 17: ConMod user interface showing Actuate_E object and dialogs

8.2 Implementation of CEnergize_M

Figure 18 shows the implementation of the formal definition of CEnergize_M, discussed in Chapter 6.1, onto the ConMod modeling platform. ConMod supports function modeling using basic entities such as blocks and arrows through the tools shown in the right pallet of the software. It also offers qualitative and quantitative reasoning on function models through the reasoning toolbar on the right. CEnergize_M was added to the left toolbar of function features (second from the bottom). Implementing CEnergize_M to ConMod involves writing a function feature class alongside dialog functions to input the signal flows and the threshold parameter, and drawing the graphical template that reflects the state of the function (Energize_M or DeEnergize_M). Figure 12 shows the dialog box asking for the threshold parameter input and the graphical template for the CEnergize_M function in an active Energize_M mode. The mode of the function is determined through the grammar in Table 13 which compares the status signal to the threshold parameter. In Section 8.5.3, we demonstrate the reasoning efficacy of this modeling language, where the model of a pump-turbine system can simulate the repercussions of changing a status signal value in the form of the system shifting from the turbine and generator mode to the motor and pump mode, which are conjugates of each other.

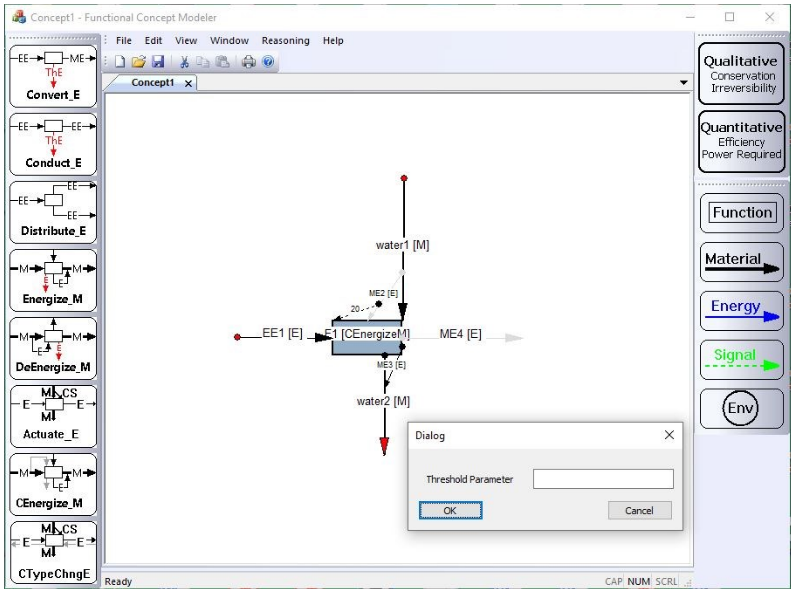


Figure 18 Implementation of CEnergize_M onto ConMod

8.3 Implementation of FSAs for Function Modeling

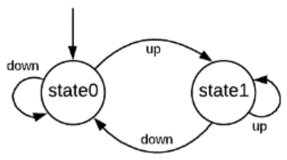
For enabling the modeler to declare the FSA rules during modeling, ConMod 2.0 is augmented with two capabilities: (1) FSAs are described as a class that can be instantiated into a model (pseudo code shown in Table 22), and (2) a user-friendly plain-text protocol for entering the data for the FSAs (example shown in Table 23). The first line of Table 22 says that FSAs are a quintuple containing a set of states, control signals, an initial state, a set of final states, and a transition matrix. The subsequent lines are explained by the comments within the table.

Table 22: Pseudocode for the FSA class

```
$\boldsymbol{FSA = \{S, C, state_i, \{state_f\}, T\}}$    // an FSA is defined with states, control signals, an initial state, a set of possible final states, and a transition matrix
$\boldsymbol{S = \{state_1, \dots, state_n\}}$    // n is the total number of states (S) in the FSA
$\boldsymbol{C = \{CS_1, \dots, CS_m\}}$    // m is the number of control signals (CS)
$\boldsymbol{T = \{line_1, line_2, \dots line_j, \dots\ line_k\}}$    // k is the total number of lines in the transition matrix, T
$\boldsymbol{line_j = \{state_p, command_r, state_q\}}$    // each line in the matrix must have a start state, an end state, and a command that transition the system from the start state to the end state
$\boldsymbol{state_i \in S}$    // the initial state of the FSA must be a valid state
$\boldsymbol{state_f \subset S}$    // the final states are a subset of the total available states
$\boldsymbol{state_p \in S}$    // in each line of the matrix, the start state must be a valid state
$\boldsymbol{state_q \in S}$    // in each line of the matrix, the end state must be a valid state
$\boldsymbol{CS_r \in C}$    // in each line of the matrix, the command must be valid
```

Table 23 (left) shows the plain-text data for declaring FSAs, using the light switch example (right). The first two lines declares the sets of valid states and commands. The third line declares the initial state at which the FSA is instantiated into the model by default. The subsequent lines constitute the state transition matrix and are formatted as line 5 of Table 22. For example, line 4 of Table 23 says that starting at state0, an “up” command takes the device to state1, etc. Since FSAs represent not just devices but device classes, the same text file can be used for an entire class that shares a behavior, with keywords such as up and state0 replaced with more meaningful terms, if necessary, without changing the data format. Designers can create libraries of these text files to represent different device classes. During modeling, the program ask the user to select a text file for each function that needs an FSA (such as Actuate_E in this example).

Table 23: FSA representation of a light switch

`S = {state0, state1};`
`C = {up, down};`
`state0;`
`state0, up, state1;`
`state0, down, state0;`
`state1, up, state1;`
`state1, down, state0;`
	

8.4 Algorithms to Determine the End State of a Function Receiving Control Signals

This section presents the algorithms that use the FSAs to determine the end state of a function and its outgoing in response to control signals. With FSAs now described formally, this reasoning can be accomplished in two steps, as described below.

8.4.1 Algorithm for Checking FSA Validity

Since the FSA text files are subject to manual editing, the program must check for data validity before using them for modeling or reasoning. This section presents the conditions and algorithm for this purpose. Collectively, these checks ensure that the FSA has the necessary and sufficient information for determining final states and that the model never enters an invalid state.

Initial state condition: The initial state declared in line 3 of the text file must belong to the list of states, S, declared in line 1.

State name check: The start and end state names in each line of the state-transition matrix (element 1 and 3 in line 4 onward in the text file) must belong to the list S declared in line 1.

Command name check: The command name in each line of the matrix (second element of the line) must belong to the list C declared in line 2.

Completeness check: For each combination of a state $\boldsymbol{state_p}$ and control signal $\boldsymbol{CS_j}$, there must be exactly one line in the matrix whose first element is $\boldsymbol{state_p}$ and the second element is $\boldsymbol{CS_j}$, whereas the last element (destination state) can be any member within $\boldsymbol{S}$. This check ensures that there is a transition rule available for processing any command starting at any state. It also has a consequence that the number of lines in the matrix ($\boldsymbol{k}$) is the product of the number of states ($\boldsymbol{n}$) and control signals ($\boldsymbol{m}$). For the FSA in Table 23, $\boldsymbol{n = 2}$, $\boldsymbol{m = 2}$, thus $\boldsymbol{k = 4}$.

Figure 19 shows the algorithm for implementing the above checks. The steps listed above are highlighted in the chart for ease of reference. Table 24 shows the error messages produced by this algorithm.

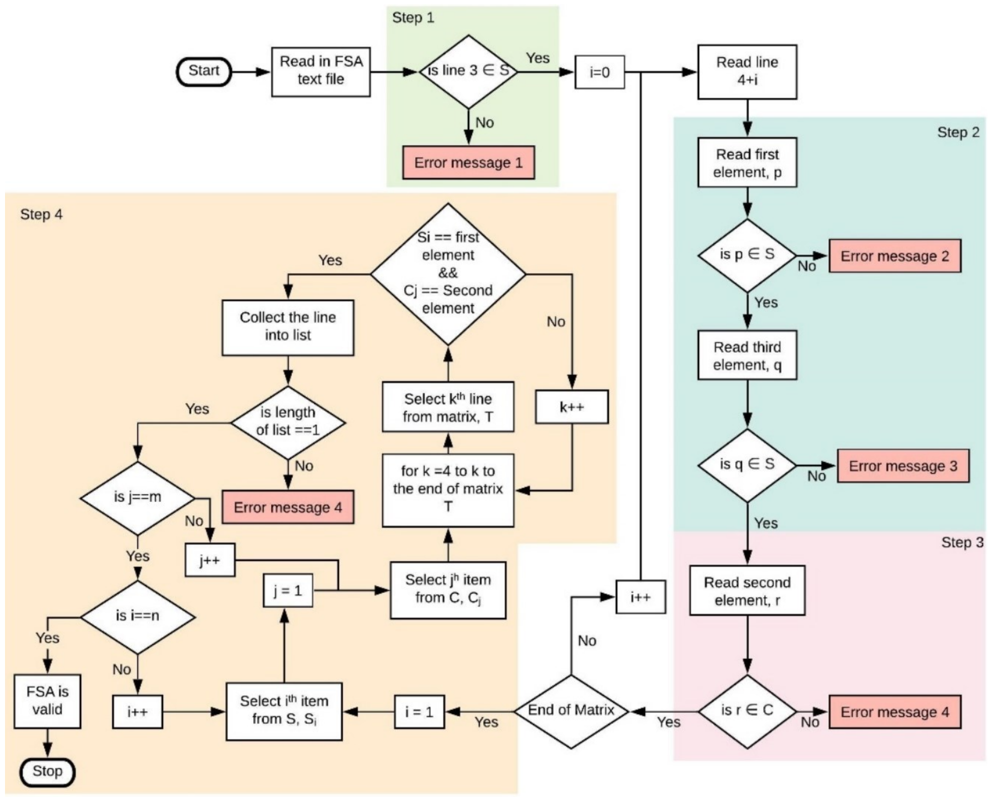


Figure 19: Algorithm for checking FSA validity

Table 24: List of Error Messages

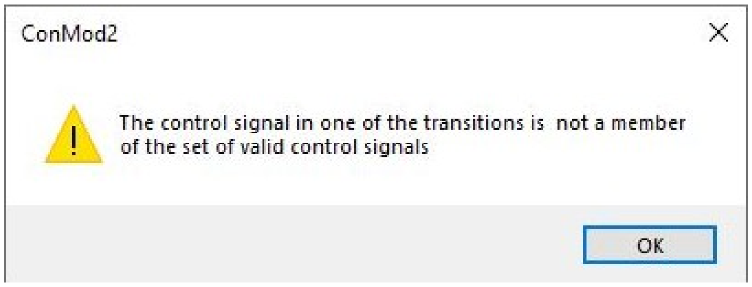
Error message 1	The initial state is not a member of the set of valid states.
Error message 2	A start state in one of the transitions is not a member of the set of valid states.
Error message 3	A end state in one of the transitions is not a member of the set of valid states.
Error message 4	The control signal in one of the transitions is not a member of the set of valid control signals.

The above algorithm was implemented onto ConMod 2.0. Figure 20 shows an error message issued by the program because of an incorrect FSA file input. The FSA represents a ball-point pen switch, which has only one control signal resulting in two lines of state transitions, as shown in Table 25 (left). The incorrect file on the right includes a control signal "unpress" that does not belong in the list C of line 2, which prompts error message 4 from Table 24.

Table 25: Correct and incorrect FSA for a switch ballpoint pen

Correct File	Incorrect File

S = {state0, state1};
C = {press};
state1;
state1, press, state0;
state0, press, state1;
	S = {state0, state1};
	C = {press};
	state1;
	state1, press, state0;
	state0, unpress, state1;


ConMod2

The control signal in one of the transitions is not a member of the set of valid control signals

Figure 20: An error message prompted in response to incorrect FSA input

8.4.2 Algorithm for Finding the Function's End State

This section presents the algorithms implemented on ConMod to determine the final state of a function and its output flows in response to a user-supplied control signal (Figure 21). This algorithm is executed only if the FSA validity checks pass. Upon receiving a control signal from the user, the program checks the current state of the function and looks for a line in the state transition matrix that has the current state in the first element and the received signal in the second element. Such a line is guaranteed to be available due to the completeness check mentioned above. Once the line is found, the third element of the line gives the new current state (i.e., end state) of the transition.

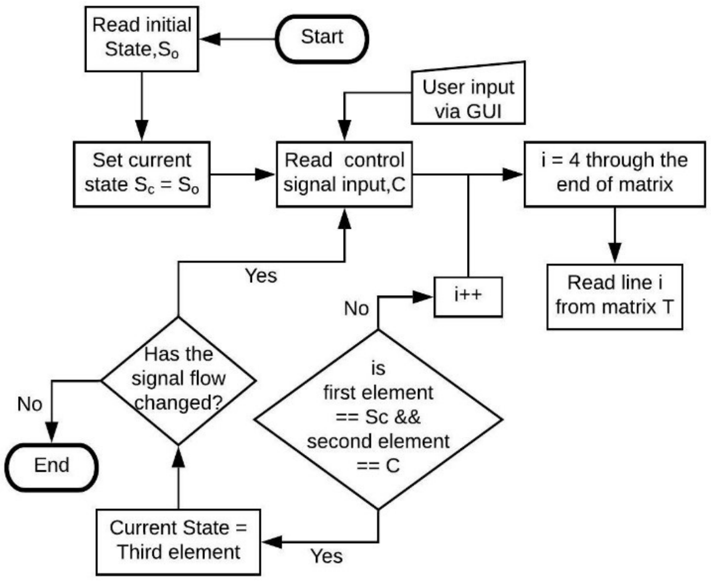
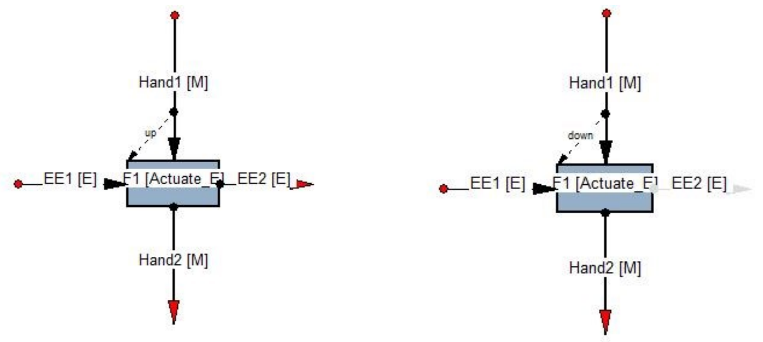

Start
Read initial State,S_o
Set current state $S_c = S_o$
User input via GUI
Read control signal input,C
i = 4 through the end of matrix
Read line i from matrix T
i++
No
is first element == Sc && second element == C
Yes
Current State = Third element
Has the signal flow changed?
Yes
No
End

Figure 21: Algorithm for finding end state of the function

These resulting states lead to consequences on the output flows. For example, in a light switch, if the signal is "up", the resulting current state S_c = state1 according to Figure 21, and the output energy flow exists in the same form as the input energy, according to the definition of the verb Actuate_E in Table 6. Otherwise, `Sc = state0`, therefore `E_out_primary.subtype = ϕ` in Table 6, and the flow ceases to exist (the function acts as the Stop verb in the Functional Basis). Figure 22 shows these two states of Actuate_E produced by ConMod, operating under the FSA of the light switch shown in Table 23. Their difference is that the received input signal is "up" for the left figure, thus the outgoing EE2 flow is actuated (shown in black), while on the right figure the signal is "down", which causes EE2 to cease (shown in light gray). Thus, it is now demonstrated that the FSA-based modeling of states can control a function's behavior or operational mode. In the next section, this same effect is extended on the whole model.



(a) signal = "up" actuates EE2 **(b) signal = "down" stops EE2**

Figure 22: Effect of state change of the Actuate_E function

8.5 Algorithm for Propagating the Effect of a Control Signal Through the Model

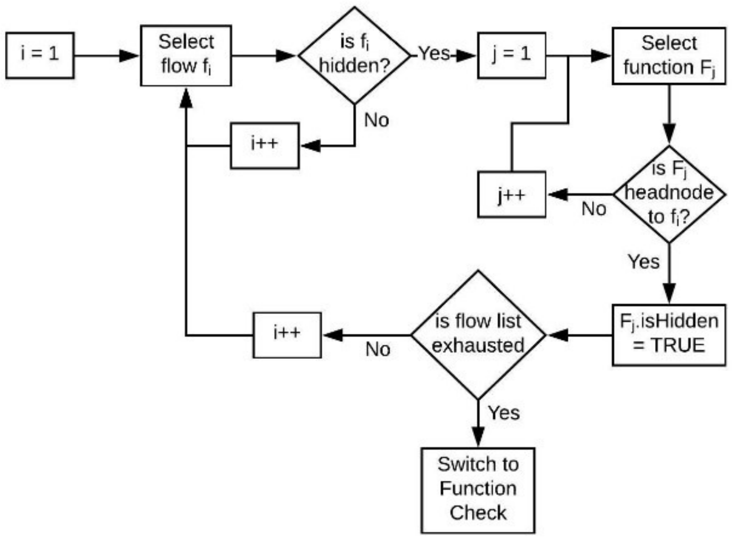
8.5.1 Algorithm for Propagating the Downstream Causal Effect

As the output flows of signal-processing functions such as Actuate_E are actuated or ceased according to the algorithm of Figure 21, the downstream functions that receive those flows are impacted accordingly, which in turn impacts the outputs of those functions. A causal propagation thus trickles down the remainder of the model based on its topology. This section presents the algorithm for computing that propagation (Figure 23).

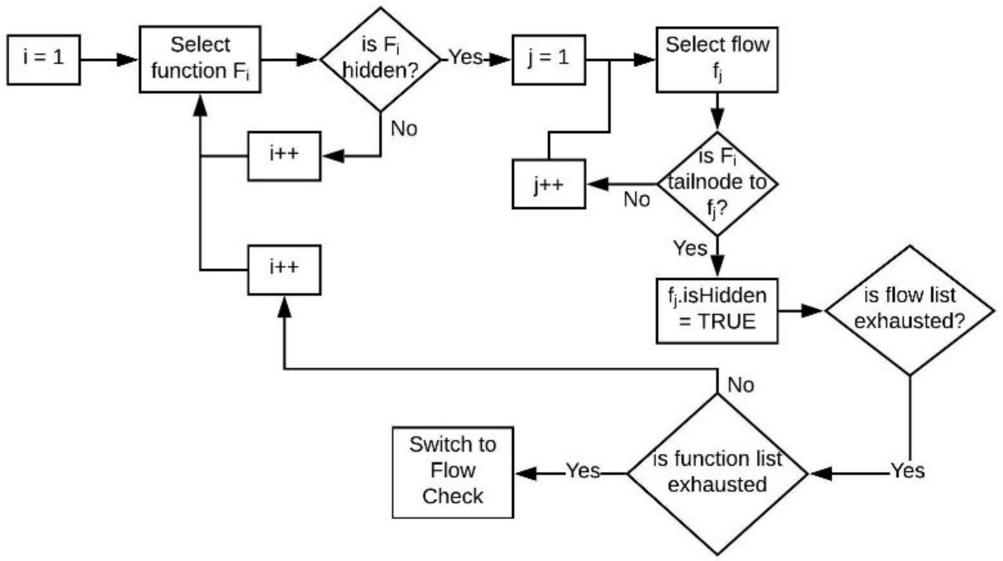
On the graphics side of ConMod, the cessation of a flow is shown by turning it light grey. Internally, this transition is controlled by a Boolean variable "isHidden" attached to the functions and flows. When this variable is set to TRUE, the colors are light grey, and the objects are assumed to have ceased.

The propagation is triggered whenever the value of a control signal in the model is changed, which changes the state (actuated/ceased) of flows from one or more signal-processing functions such as Actuate_E. Note that a model can contain multiple signal-processing functions. Starting here, the program looks for the function block located at the head of each flow that was originally added by the modeler but is currently in the ceased state and turns those blocks to the ceased state. This portion of the algorithm is shown in Figure 23(a). Once the head nodes of all ceased flows are ceased, the control moves to computing the output flows of those ceased functions and turns them into the ceased state, as shown in Figure 23(b). Subsequently, the program switches between these two algorithms, until no output flow from a ceased function and no ceased flow with a head node is found.

When a reversal in the control signal to a signal-processing function causes a flow to resume, the opposite causal effect is propagated through the model, and some of the previously hidden blocks and arrows could return to existence. The difference between these two opposite operations is that a function block will cease if even one of its input flows is ceased, but it will need all its input flows to resume in order to resume itself. In the next section, we present a system-level function model created on ConMod using two Actuate_E functions and validate that these algorithms indeed create the desired effects explained here.



(a) Ceased flows causing their head node functions to cease



(b) Ceased functions causing their output flows to cease

Figure 23: Algorithm to perform causal reasoning on flow actuation and stopping

8.5.2 Algorithm for Propagating the Upstream Causal Effect

When a signal flow causes the mode of a conjugate function to switch, the upstream and downstream functions affected by the switch in flows ceasing or actuating must be captured. If the input energy flow to a CEnergize_M instance, which is currently in DeEnergize_M mode, is ceased, then all the upstream functions from CEnergize_M must be ceased. Consecutively, if the output flow from CEnergize_M in the Energize_M mode is ceased, then all the downstream functions from CEnergize_M must be ceased. Since the conjugate function can be used at any point within the model, allowances must be made for both upstream and downstream causal propagation. This section looks at the upstream causal effects through the example of CEnergize_M.

When CEnergize_M is in DeEnergize_M mode, all the upstream functions and flows must be ceased, and they would be actuated when the function is in the Energize_M mode. The algorithms for ceasing upstream flows and functions are shown in Figure 24 and Figure 25 respectively. If a changed signal belongs to CEnergize_M template, and the mode of the said template is Energize_M, the program loops through the list of flows and checks for hidden flows. If the flow is hidden, the program loops through the list of functions and checks if the hidden flow has a tailnode relation with a function – if it does, the function also ceases. This process is repeated till both the function and flow lists are exhausted and the program switches to check for ceased functions. The algorithm for ceased functions (Figure 25) loops through the function list to look for hidden functions, if a function is hidden, then it loops through the flow list to obtain all flows that have a headnode relationship with that function and ceases them. Once both the function and flow lists are exhausted, the algorithm switches back to flow check. Subsequently, these two checks are repeated until no input flow from a ceased function and no ceased flow with a tailnode is found.

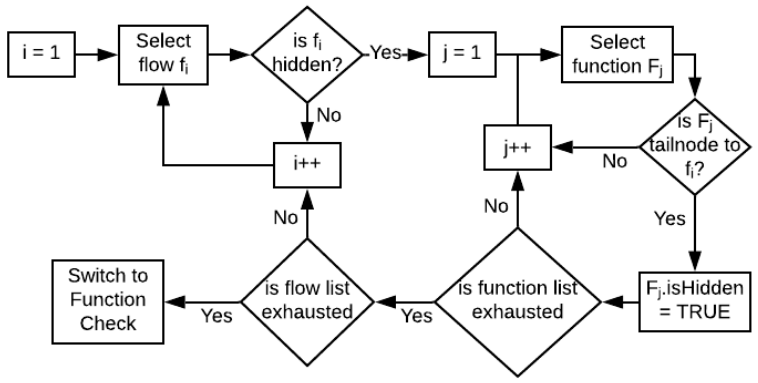


Figure 24 Algorithm to cease upstream flows

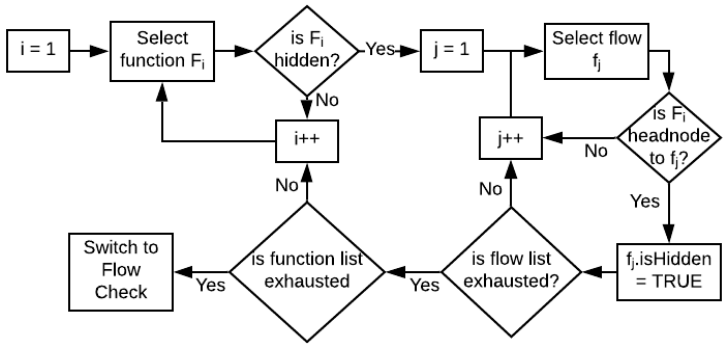
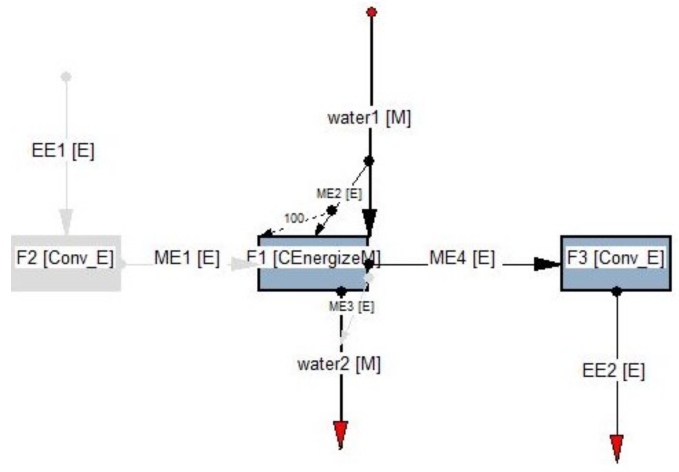


Figure 25 Algorithm to cease upstream functions

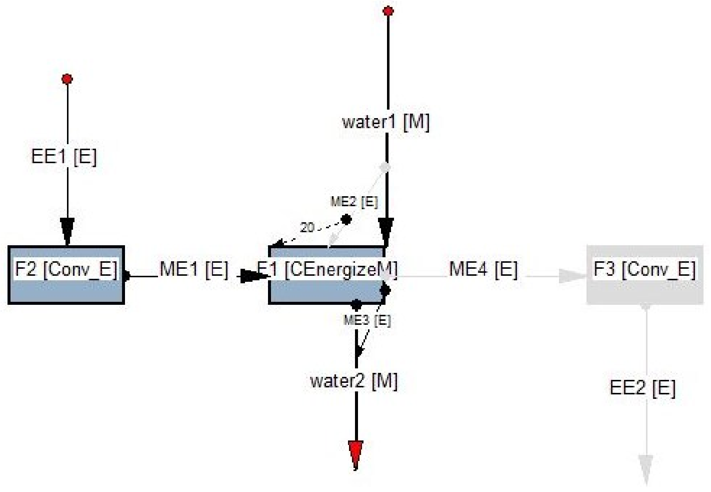
In the Energize_M mode, the algorithm will actuate the ceased flows by changing the isHidden variable to false to create an opposite causal effect on the function model by resurfacing the hidden functions and flows. Additionally, the program runs the algorithm for the downstream propagation of causal effects that is presented in section 8.5.1. The difference between the two algorithms is that the program looks for headnode relations in the flow check and the tailnode relations in the function check.

8.5.3 Demonstration of Causal Propagation

To demonstrate the efficacy of the algorithms, a pump-turbine system discussed in Figure 2 is modeled using CEnergize_M function – shown in Figure 26. In this case, the threshold parameter is set to 50. When the value of the signal is greater than 50, the system is in the turbine-generator mode and in DeEnergize_M mode where all the upstream functions are ceased and the downstream functions are actuated (Figure 26a). When the value of the signal is less than the threshold, and model simulates that the system switches to the motor-pump mode, where the downstream functions are ceased and the upstream functions are actuated (Figure 26b). Thus, the causal propagation ensures that in each scenario only one mode is active as dictated by the logical rules prescribed in the formal definitions of the function. Chapter 9 further exemplifies the downstream causal propagation through the example of a hairdryer.



(a) Turbine Mode



(b) Pump Mode

Figure 26 Pump-turbine system modeled in ConMod

Chapter 9
Reasoning: FSA-Based Reasoning Supported by the Formalism

The research gap identified in the beginning of this paper was the inability to capture multiple states and modes of a device in the same function model and to perform reasoning to determine the state and mode resulting in response to control signals. This section illustrates the ability of the new verbs and the FSA approach to address this gap.

We use a hairdryer product for this illustration, because it has the essential characters of a complex system in a relatively simple structure: it involves multiple energy types (electrical, thermal, kinetic, and others as losses), a material flow (air), and signal-processing components such as fuses and switches, connected in a reasonably complex topology (Figure 27).

This model is constructed on ConMod using the formal language of function modeling proposed earlier and uses verbs such as Convert_E and Energize_M defined in that research [77]. It also uses two instances of Actuate_E defined in the current paper. The Convert_E objects describe conversion of energy types by the motor (block F2 in model) and heater (F10). Energize_M is used to show the addition of energy flows to material flows and is used to describe the fan rotor (F3) and the heater (F4), which add mechanical and thermal energy to the air, respectively. The hexagons represent Environment objects and in effect define the system boundary. Up to this point, the model is based on the prior modeling practice of [77].

The two Actuate_E verbs represent a binary switch (on/off) and a fuse, respectively. For the switch (F1), the human hand (material) carries in a control signal of value “up” to the function, which allows the electrical energy flow EE2 to the motor. The fuse (F9) senses the temperature of Air3 exiting the heater (F4) and is designed to shut off EE4 supplied to the heater (F10) when it exceeds a critical value Tc. For clarity, note that both F10 and F4 are functions executed by the same heater, for producing heat and for adding it to the air, respectively. At present, T < Tc, as shown by the "<" symbol on the signal entering F9. Thus, EE4 is actuated and the heater is on. Hence, this model describes the system's normal operating mode, where the fan is propelling the air and the heater is heating it. If the modeler wrote correct logical behaviors in the FSAs of these two Actuate_E functions, this model will be able to support various causal and logical reasoning, as shown next.

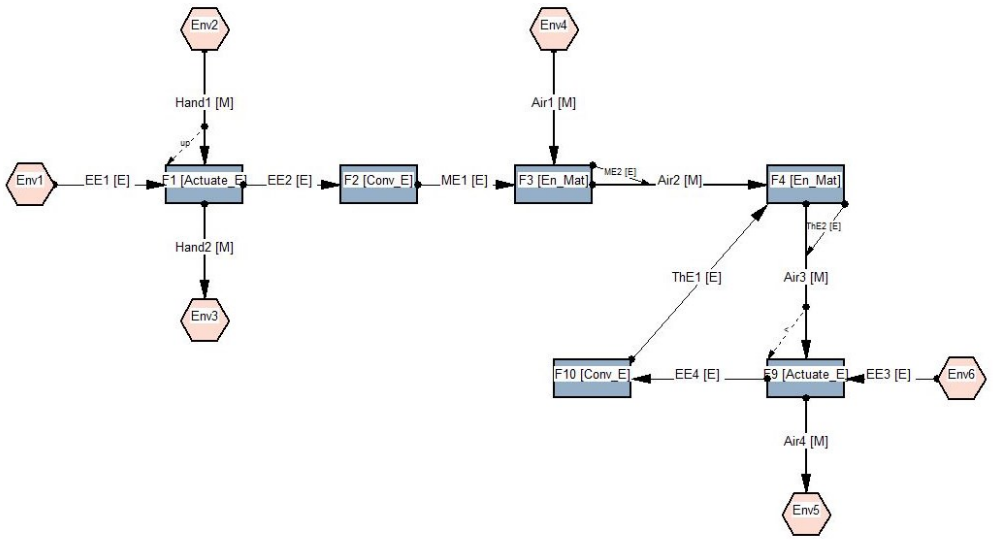


Figure 27: System-level model of a hairdryer in its normal operating mode

Assume that the modeler wants to analyze the effects of Air3 being overheated above Tc. The cause of this event is outside the scope of this reasoning, and could include events such as the incoming Air1 being already too hot or the flow rate of the air being reduced by an obstruction, thus reducing the heat removal rate from the heater, etc. In our case, the designer enters a value of T > Tc in the flow properties dialog of Air3, thus initiating the causal propagation. If the FSA of the fuse is declared correctly, once Air3 carries the information T > Tc to the fuse, the FSA should use the line in the transformation matrix that says "state1, T>Tc, state0" to push the function F9 to state0, where it shuts off EE4 by setting `EE4.IsHidden = true`. As a result, by propagation, the Convert_E function of the heater (F10), which is the recipient of EE4, should cease next. Further, ThE1 should be set to `ThE1.IsHidden = true`. Propagating this effect further down ThE1, the Energize_M function of the heater (F4) should next cease, which should finally prevent the heat ThE2 from being added to Air3. In effect, overheating of Air3 should shut off only the heating subsystems, without impacting the motor and fan subsystems. The device should thus switch from the normal mode to the cold-shot mode. The above hypothetical exercise sets the expected behavior of the program if T > Tc was entered as a signal.

Figure 28 shows the resulting state of the model of Figure 27 upon entering T > Tc. This reasoning is implemented in ConMod. As seen in this figure, the functions and flows comprising the heating subsystem in the bottom-right corner of the figure are ceased in this model, shown by their light grey color – exactly replicating the hypothetical scenario above. It is therefore validated here that the formal definitions of the signal-processing verbs and the FSA-based declaration of state transitions of those verb, as presented in this paper, are able to describe multi-state and multi-mode systems in the single function model and use that information to reason the propagation of the effects of control signals.

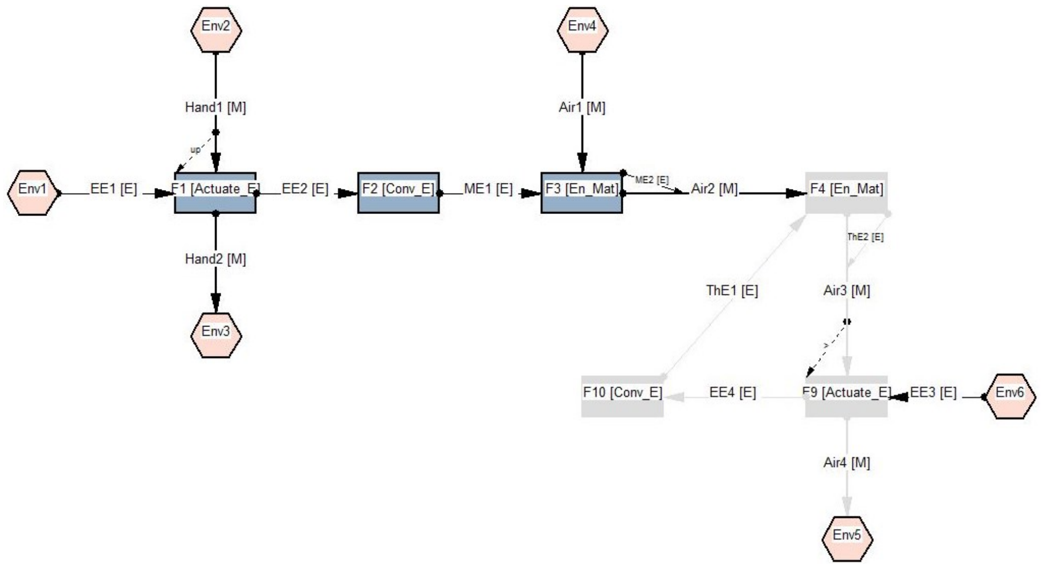


Figure 28: Resulting state of the model of Figure 27 after receiving signal T > Tc

Such as capability can be quite useful in design, especially for checking what-if scenarios and evaluating design alternatives. For example, consider how the system would behave differently if the fuse were replaced by a thermostat that resumed the EE2 flow when the temperature returned below Tc. Such a thermostat could still be described by Actuate_E, although with a different FSA that resembles the light switch of Table 4—the only difference being that the signal values are "C = {>, <}" instead of the "C = {up, down}". In this case, the model would be able to additionally predict that when the temperature of Air3 returned below Tc, the greyed-out subgraph of Figure 28 would be turned back on. By extension, it could indicate to the designer that if nothing were altered, the system would keep toggling between the normal mode and the cold-shot mode, automatically and indefinitely. Such predictions of causal behavior based on physics and formal logical definitions of flows and functions was not supported in graph-based function models previously and the illustration above addresses that gap.

Chapter 10
Reasoning: Conjugacy-Based Reasoning Supported by the Formalism

This section illustrates the reasoning and modeling abilities of the proposed conjugate functions and features using the example of a direct expansion geothermal heat-pump (GHP). We chose this system as it presents multiple conjugate functions and, at the system level, switched between opposite modes: heating and cooling. Figure 29 shows the schematic of the GHP. Figure 30 and Figure 31 shows its function model drawn using the conjugate verbs of Chapter 3. Note that in both the heating and the cooling modes, the system runs a vapor compression refrigeration cycle, only the direction of net heat transfer changes.

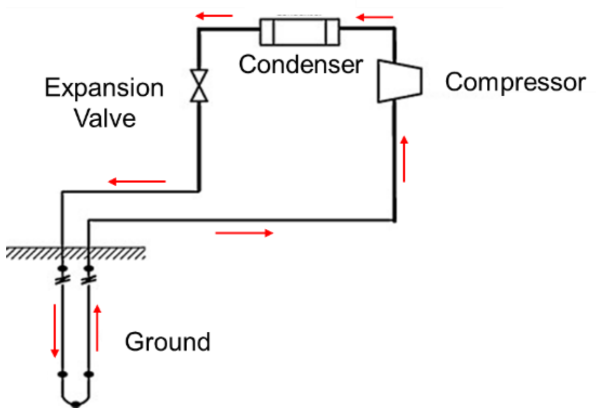


Figure 29: Schematic of a geothermal heat pump

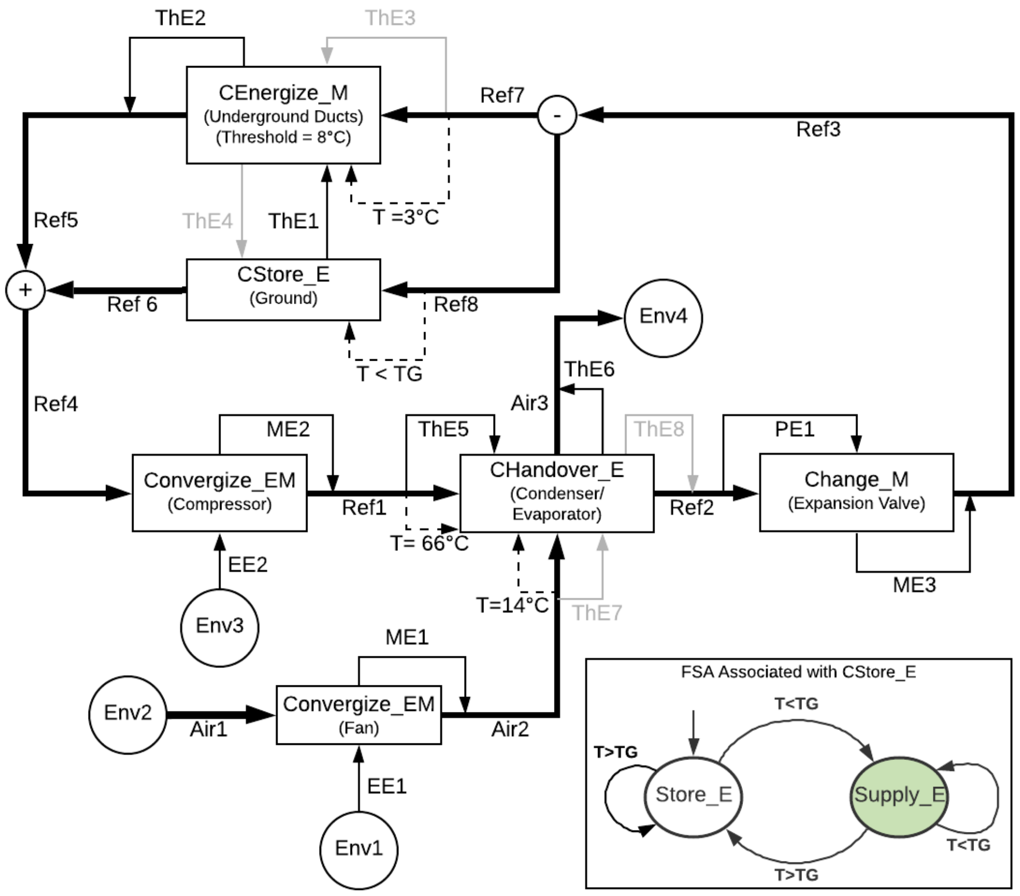


Figure 30: Function model of a GHP in heating mode

In the heating mode, the system transfers heat from a colder ground to a warmer room against the temperature gradient and, therefore, consumes work at the compressor according to the second law of thermodynamics. The processes are as follows:

Evaporator: First, the refrigerant absorbs heat from the ground while passing through the underground evaporator ducts and comes to a boil. This effect is captured by two functions: (1) CStore_E, where the ground supplies heat as a source in the Supply_E mode, and (2) CEnergize_M, where that heat energizes the refrigerant in the Energize_M mode.

Compressor: Next, the vapor enters the compressor, shown by the Convergize_EM block, where a motor first converts electrical energy (EE2) to mechanical work (ME2) using TypeChange_E, and ME2 is then added to the vapor by the compressor using Energize_M. Convergize_M was introduced in [81].

Condenser: Next, the vapor, now hotter than the room air upon compression, enters the condenser where it "hands over" its heat to the room air using the CHandover_E block operating in the so-called "fluid-to-room" mode. The room air is driven by an electric fan, which uses another Convergize_EM block that operates like the compressor.

Expander: Finally, the refrigerant, now in a supercooled state after rejecting heat, enters the expansion valve, which changes its thermodynamic state without net energy or work transfer, shown by the Change_M block.

In the cooling mode, the system transfers heat from a colder room to a hotter ground, once again against the temperature gradient, and thus consumes work. This situation can be modeled by simply flipping the three conjugate functions of the ground (CStore_E), underground ducts (CEnergize_M), and the condenser (CHandover_E) in the model. The arrow direction of the refrigerant flows will reverse, while the air flow directions will remain intact. In this case, the ground becomes a heat sink running in the Store_E mode (instead of a source), the coils submerged in the ground becomes the condenser running in the DeEnergize_M mode (instead of the evaporator) so that the hotter refrigerant can reject heat to it, and the coil in the path of the room air becomes the evaporator running in the Energize_M mode (instead of the condenser), so that it absorbs heat while evaporating. Since the refrigerant flow directions are flipped, the system still follows the evaporator-compressor-condenser-expander order that is needed for the vapor compression cycle. This resulting model is shown in Figure 31.

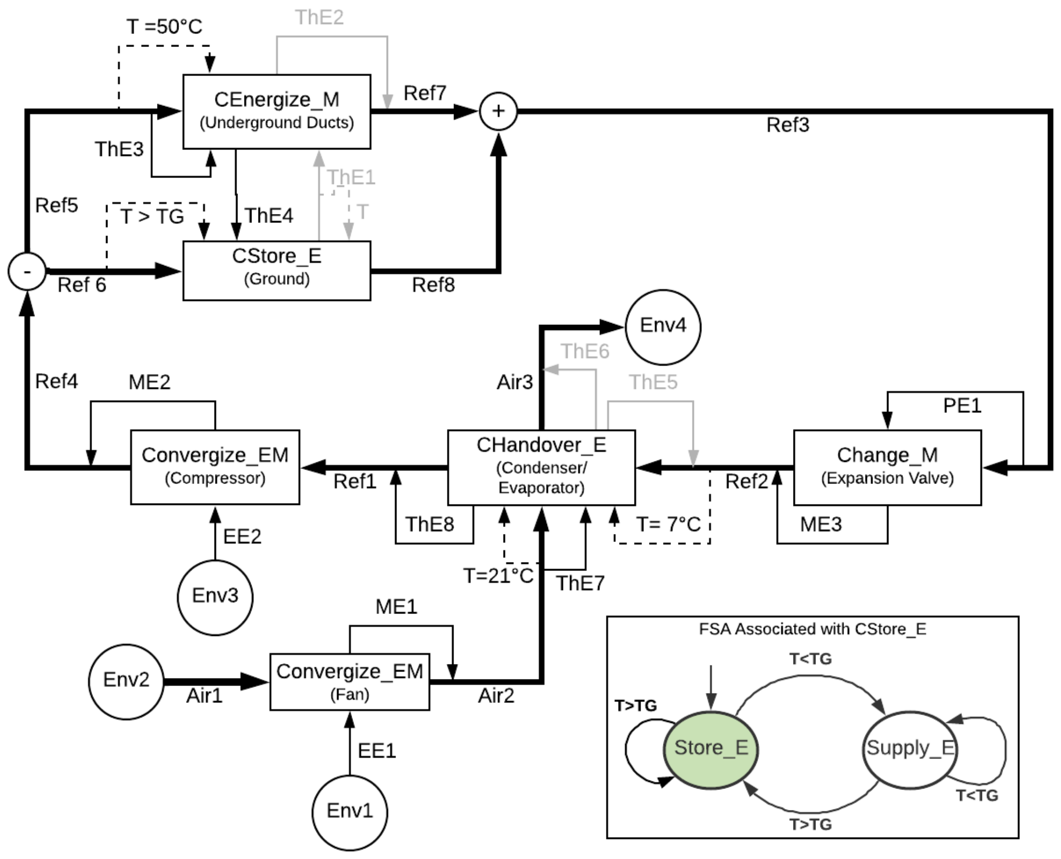


Figure 31: Function model of a GHP in cooling mode

The signal flows that control the transitions between the above two modes are shown in the function model. CEnergize_M of the underground coils needs a status signal of the refrigerant's temperature to determine the direction of heat transfer based on a user-defined threshold. Similarly, CHandover_E senses the temperatures of the refrigerant and the room air to determine the direction of heat transfer based on the temperature gradient. CStore_E needs a FSA input, which can come from the user. Thus, we have demonstrated that incorporating conjugate verbs and features in function models can enable the model to predict and simulate the reversal of a system's functional mode in response to a user-given signal or a change in the temperature gradient between the room and the ground, using function models.

Further, when properly extended, this approach could assess the difference in efficiency of multiple modes. Most devices operating in dual modes tend to be more efficient in one mode than in the other. The current ConMod language supports assigning quantitative details to flows, especially it uses the product of and effort term (F_{in} and F_{out}) and a rate term (R_{in} and R_{out}) associated with energy flows to capture their power, similar to Bond Graphs [77]:

$$\boldsymbol{\eta = \frac{F_{out}R_{out}}{F_{in}R_{in}} \times 100\%}$$

Since the quantitative information needed to calculate the status signals such as temperature differences must already be present in these model in order for the conjugate behavior to manifest, it follows that by adding the effort and rate terms to the flows, the whole system's efficiency in each mode, and their difference, could be computed.

In summary, it is demonstrated that conjugate functions can convey more information than single-mode functions. Such a representation captures multi-modal subsystems such as heat exchangers and thermal reservoirs in a single graphical unit while offering logic-based simulation of their mode transition, which could be implemented on the computer. Additionally, feature-level implementation of conjugacy makes models more meaningful and the language more powerful since the features encapsulate complex conjugate behavior in easy-to-model units. Thus, the extension of the ConMod formal language shown in this paper—including conjugate functions, conjugate features, FSAs, and the use of signals to trigger mode transitions—collectively augments the efficacy of function modeling.

Chapter 11
Conclusions

This chapter outlines the summary of contributions of this research, answers to the research questions, and the impact on the state of the art and future work.

11.1 Summary of Contributions

This thesis makes several novel contributions to the state of function-based design:

1. It introduces a rigorous formalism of modeling device states and state transitions within the function modeling context based on finite state automata (FSAs) described in formal language and automata theory. While the need for modeling device states has been previously recognized, such an approach was not presented for the graph-based function models.

2. It presents a formal and computationally rigorous definitions of three function verbs that actuate or regulate energy flows in response to signals: Actuate_E, Regulate_E_Discrete, and Regulate_E_Continuous (summarized in Table 26). The verbs collectively cover devices that were conventionally described by Actuate and Regulate within earlier vocabularies such as the Functional Basis. The vocabulary and grammar included in the new formal definitions are built upon a previously validated physics-based language of function representation, and thus enlarges the scope of formal, physics-consistent function modeling.

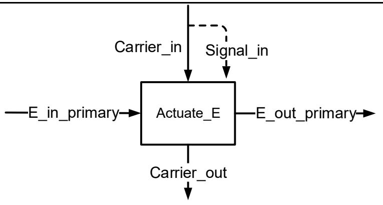
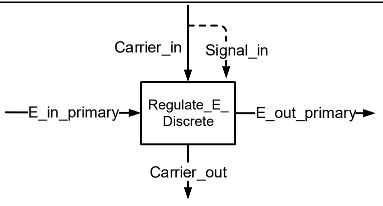
3. It institutes four rigorously defined conjugate verbs (summarized in Table 26) that change their operating modes by performing logical reasoning on the incoming signal flows. By doing so, they augment the current function modeling formalism by enabling the modeling of multi-mode systems.

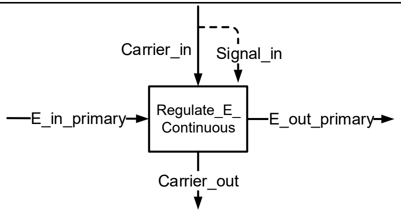
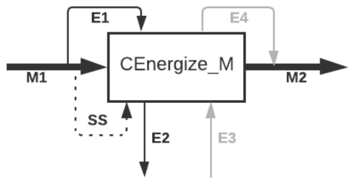
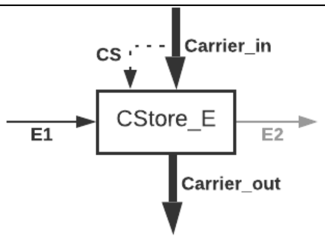
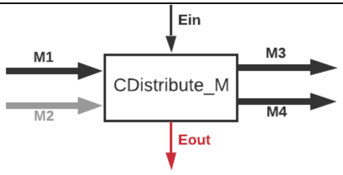
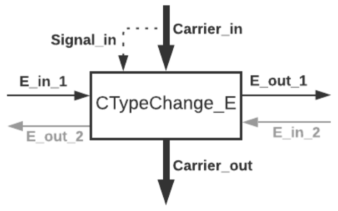
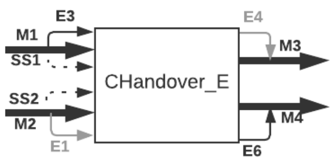
4. It presents an approach to integrate conjugate functions into conjugate features and illustrates that by formalizing two conjugate features: CHandover_E and CConvergize_EM (summarized in Table 26).

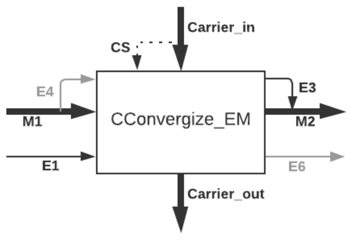
5. It validates that the formal definitions of the signal-processing verbs and the FSA-based declaration of state transitions of those verb are able to describe multi-state and multi-mode systems in the single function model and use that information to reason the propagation of the effects of control signals. The main contribution of this validation is the extension to the current ConMod software.

6. It demonstrates the reasoning efficacy of the conjugate function representation using a system level model that is built using the conjugate verbs defined in this paper. by demonstrating the idea of conjugate functions at a function-level, feature-level, and in the system-level, the scalability of this approach is demonstrated.

Table 26 Summary of vocabulary

Verb	Textual Definition	Graphical Template
Actuate_E	Commence or stop an energy flow based on a control signal input.	 Carrier_in Signal_in E_in_primary Actuate_E E_out_primary Carrier_out
Regulate_E_ Discrete	Change the quantity of an energy flow in discrete steps based on a control signal input.	 Carrier_in Signal_in E_in_primary Regulate_E_ Discrete E_out_primary Carrier_out

Regulate_E_ Continuous	Change the quantity of an energy flow continuously based on a control signal input.	 Carrier_in Signal_in E_in_primary Regulate_E_ Continuous E_out_primary Carrier_out
CEnergize_M	Add or remove energy to/from a material flow based on a status signal and a threshold parameter.	
CStore_E	Store or supply an energy flow based on a control signal input.	
CDistribute_M	Distribute or Couple a material flow based on the number of active inputs to the function.	
CTypeChange_E	Change the subtype of an energy flow and reverse the change based on a control signal input.	 Signal_in Carrier_in E_in_1 E_out_1 CTypeChange_E E_out_2 E_in_2 Carrier_out
CHandover_E	Transfer energy from one material flow to another or vice-versa based on a positive energy-transfer gradient.	 E3 E4 M1 M3 SS1 CHandover_E SS2 M4 M2 E1 E6

CConvergize_EM	Change the subtype of an energy flow and add it to a material flow or remove an energy flow from a material flow and change its type, based on a control signal input.	 Carrier_in CS E4 M1 CConvergize_EM E3 M2 E1 E6 Carrier_out

11.2 Answers to Research Questions

In this section, the research questions identified in Chapter 1 are answered. The research tasks are addressed first and then the status of each research hypothesis are discussed below:

RQ-1.1	How can the formal language capture effect of state-change of a device on the function model?
Task 1	Formally define functions that capture the state-changes of a device
Answer	The functions defined in Chapter 5, Chapter 6, and Chapter 7 have at-least two modes of operation and they can be used to catpture the effect of state-change of a device.
RQ-1.2	Is the proposed formalism computable?
Task 2	Implement FSA based function modeling to ConMod
Answer	The function Actuate_E was implemented onto ConMod. Algorithms to reason on FSA inputs (Section 8.4.1 and 8.4.2) and to determine function state ensure that the proposed formalism is computable.

RQ-1.3	Does the formalism support causal dependencies of functions on each other?
Task 3	Demonstrate causal reasoning on function models
Answer	The algorithm in Section 8.5 propagates the effect of a change in the state of the flow on the system-level model. Causal-dependencies of changing flow attributes on a function model is then demonstrated in Chapter 9 through the example of a hairdryer.
RH-1	Integration of FSAs that capture device state into the formal definitions of functions will ensure that the resulting function model is reflective of the device state.
Status	TRUE.
	System-level models constructed using the verbs developed in Chapter 5 in Chapter 9 demonstrate how the proposed vocabulary can be used to capture the device state.
RQ-2.1	How many conjugate verbs can be found in the ConMod language?
Task 4	Discover Conjugate functions from the vocabulary of the ConMod language
Answer	Chapter 4.2 narrows the scope of this formalism for conjugate verbs. In doing so, it examines the vocabulary of the current ConMod language to determine all possible conjugate functions.
RQ-2.2	Is the idea of conjugacy extendable to functional features?
Task 5	Develop conjugate features

Answer	Chapter 7 shows how the idea of conjugacy can be extended to higher level verbs by introducing two conjugate features CHandover_E and CConvergize_EM.
RQ-2.3	Can conjugacy be used to reason on the operating mode of a system?
Task 6	Demonstrate mode-based reasoning on function models
Answer	Chapter 10 demonstrates how mode-based reasoning can be applied to system-level models through the example of a geothermal heat-pump.
RH-2	Development of a thorough vocabulary of conjugate functions and features allow for functions to capture more than one mode of operation of a device
Status	TRUE.
	System-level models constructed using the verbs developed in Chapter 7 and Chapter 8 in Chapter 10 demonstrate how the proposed vocabulary can be used to capture multiple operating modes in a single function model.
RH-3	Identification of the state of the function, state of the flow, and propagating the effect of a change in the state of a flow through the function model based on headnode relations will achieve the above reasoning tasks.
Status	TRUE.
	Algorithms in Chapter 8 ensures that all the reasoning tasks are completed.

11.3 Impact on the State of Art

The formalism proposed in this thesis widens the function modeling perspective by introducing multi-state functions and a method to incorporate FSA-based descriptions of devices into function models. Additionally, the idea of multi-state functions is also extrapolated into function features that exist in a higher level. By doing so, the scope of function modeling is extended to include multiple operating modes of a system in a single model. The alternative being using more than one function model to represent each operating mode without having a mean to capture the effect of how the system transitions between the said modes. This ensures that complex systems including control units can be meaningfully represented using function models. Additionally, the proposed formalism is capable of determining how the functions would change its operating mode based on the state of a flow (active or hidden) and propagate that effect on a system-level model. This increases the modeling efficacy of function modeling and provides more useful qualitative and quantitative information regarding each operating mode as illustrated in Chapter 9 and Chapter 10. Lastly, it extends the current ConMod vocabulary with additional functional and feature-level verbs that are summarized in Table 26, which allows for logic based modeling of systems that require signal-flows as an input – hitherto unsupported by most function modleing representations.

11.4 Future Work

Going forward, this research can be extended in three main directions: Implementation of the research onto ConMod, Validation of the research through human-subject studies and its employment in education. Each of these approaches are described in further detail below.

1. Implementation: The entire vocabulary proposed by this formalism (Table 26) can be implemented in the ConMod software. Additional reasoning algorithms, such as reasoning on failure propagation paths, quantitative reasoning on modes, and reasoning on complexity of system-level models based on the number of operating

modes facilitated could be implemented onto the software program to provide more meaningful user interactions. A pre-requistite to this is a comprehensive set of flow attributes for each flow type.

2. Validation: Once the software program is fully capable of supporting modal reasoning, human subject studies could be conducted to test the modeling efficacy of the proposed formalism. Pertinent questions such as: Does the inclusion of multi-state functions prompt users to generate more complex solutions to design problems? Does the inclusion of FSA-based function representations make function models more appropriate for use in industry?

3. Education: ConMod software could be used in teaching students enrolled in design courses about generating adaptive solutions to design problems. It can also be employed in education of complex systems to graphically teach the causal relationships between flows, functions and the operating mode of the system.

This thesis presents a formal method to incorporate finite-state-automata into function modeling in anticipation that doing so will better the representation of complex systems in function models. It is expected that this research, in conjugation with the ConMod software, will be extended to further support multi-modal representation of systems in function modeling to provide automated support in the conceptual design phase of the engineering design process.

References

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

[2] Ullman, D. G., 1992, *The Mechanical Design Process*, McGraw-Hill.

[3] Otto, K., and Wood, K., 2001, *Product Design : Techiniques in Reverse Engineering and New Product Development*, Prentice-Hall, Upper Saddle River, NJ.

[4] Ulrich, K., and Eppinger, S., 2007, *Product Design and Development*.

[5] Bohm, M. R., Stone, R. B., Simpson, T. W., and Steva, E. D., 2008, "Introduction of A Data Schema to Support A Design Repository," Computer Aided Design, 40(7), pp. 801–811.

[6] Nagel, R. L., Vucovich, J. P., Stone, R. B., and McAdams, D. a., 2007, "Signal Flow Grammar From the Functional Basis," Guidelines for a Decision Support Method Adapted to NPD Processes.

[7] Sen, C., Caldwell, B. W., Summers, J. D., and Mocko, G. M., 2010, "Evaluation of the Functional Basis Using an Information Theoretic Approach," Artificial Intelligence for Engineering Design Analysis and Manufacturing, AIEDAM, 24(1), pp. 87–105.

[8] Stone, R. B., Tumer, I. Y., and Van Wie, M., 2005, "The Function-Failure Design Method," Journal of Mechanical Design, 127(3), pp. 397–407.

[9] Nagel, R. L., Perry, K., Stone, R. B., and McAdams, D. A., 2009, "Functioncad: A Functional Modeling Application Based on the Function Design Framework," *ASME 2009 IDETC/ CIE*, San Diego, California, August 30–September 2, 2009, DETC2009-87010, pp. 591–600.

[10] Chowdhury, A., Mao, X., Sen, C., and Venkatanarasimhan, L. N. A., 2019, "Finite-State Automata-Based Representation of Device States for Function Modeling and Formal Definitions of Signal-Processing Functions," *ASME 2019 IDETC/ CIE*, Anaheim, California, August 18–21, 2019 , DETC2019-98248, p. V001T02A016.

[11] Liu, C., Hildre, H. P., Zhang, H., and Rølvåg, T., 2016, "Product Architecture Design of Multi-Modal Products," Research in Engineering Design, 27, pp. 331–346.

[12] Weaver, J. M., Wood, K. L., and Jensen, D., 2008, “Transformation Facilitators: A Quantitative Analysis of Reconfigurable Products and Their Characteristics,” *Proceedings of the ASME Design Engineering Technical Conference*, Brooklyn, New York, August 3–6, 2008, DETC2008-49891, pp. 351–366.

[13] Far, B. H., and Elamy, A. H., 2006, “Functional Reasoning Theories: Problems and Perspectives,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 19(2), pp. 75–88.

[14] Chandrasekaran, B., and Josephson, J. R., 2000, “Function in Device Representation,” Engineering with Computers, 16(3–4), pp. 162–177.

[15] Chakrabarti, A., 2001, “Sharing in Design-Categories, Importance, and Issues.,” International Conference on Engineering Design (ICED), 1, pp. 21–23.

[16] Yildirim, U., Campean, F., and Williams, H., 2017, “Function Modeling Using the System State Flow Diagram,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 31(4), pp. 413–435.

[17] Xu, C., Yao, Z., Gupta, S. K., Gruninger, M., and Sriram, R., 2005, “Towards Computer-Aided Conceptual Design of Mechatronic Devices with Multiple Interaction-States,” *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference - DETC2005*, American Society of Mechanical Engineers Digital Collection, pp. 455–467.

[18] Chowdhury, A., and Venkatanarasimhan, L. N., 2020, “A Formal Representation of Conjugate Verbs in Function Modeling,” *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference* , Virtual, Online.

[19] Stone, R. B., and Wood, K. L., 2000, “Development of a Functional Basis for Design,” Journal of Mechanical Design, 122(4), pp. 359–370.

[20] Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., and Wood, K. L., 2002, “A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts,” Research in Engineering Design - Theory, Applications, and Concurrent Engineering, 13(2), pp. 65–82.

[21] Gero, J. S., and Kannengiesser, U., 2004, “The Situated Function-Behaviour-Structure Framework,” Design Studies, 25(4), pp. 373–391.

[22] Van Eck, D., McAdams, D. A., and Vermaas, P. E., 2008, “Functional Decomposition in Engineering: A Survey,” *2007 Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference, DETC2007*, American Society of Mechanical Engineers Digital Collection, pp. 227–236.

[23] Deng, Y. M., Britton, G. A., and Tor, S. B., 2000, “Constraint-Based Functional Design Verification for Conceptual Design,” CAD Computer Aided Design, 32(14), pp. 889–899.

[24] Stone, R. B., and Wood, K. L., 2000, “Development of a Functional Basis for Design,” Journal of Mechanical Design, 122(4), pp. 359–370.

[25] Otto, K. N., and Wood, K. L., 1996, “A Reverse Engineering and Redesign Methodology for Product Evolution,” Proceedings of the 1996 ASME Design Theory and Methodology Conference, 96.

[26] Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T., 1996, “Supporting Conceptual Design Based on the Function-Behavior-State Modeler,” Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 10(04), pp. 275–288.

[27] Kitamura, Y., and Mizoguchi, R., 2004, “Ontology-Based Systematization of Functional Knowledge,” Journal of Engineering Design, 15(4), pp. 327–351.

[28] Goel, A. K., and Bhatta, S. R., 2004, “Use of Design Patterns in Analogy-Based Design,” Advanced Engineering Informatics, 18(2), pp. 85–94.

[29] Bracewell, R. H., and Sharpe, J. E. E., 1996, “Functional Descriptions Used in Computer Support for Qualitative Scheme Generation—‘Schemebuilder,’” Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 10(4), pp. 333–345.

[30] Sen, C., Summers, J. D., and Mocko, G. M., 2013, “Physics-Based Reasoning in Conceptual Design Using a Formal Representation of Function Structure Graphs,” Journal of Computing and Information Science in Engineering, 13(1), pp. 011008–12.

[31] Kurtoglu, T., and Tumer, I. Y., 2008, “A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems,” ASME Journal of Mechanical Design (JMD), 130(5), pp. 51401–51408.

[32] Stone, R. B., Irem, A. E., Ae, Y. T., and Stock, M. E., “Linking Product Functionality to Historic Failures to Improve Failure Analysis in Design.”

[33] O’Halloran, B. M., Papakonstantinou, N., Giammarco, K., and Van Bossuyt, D. L., 2017, “A Graph Theory Approach to Functional Failure Propagation in Early Complex Cyber-Physical Systems (CCPSs),” INCOSE International Symposium, 27(1), pp. 1734–1748.

[34] Mcadams, D. A., and Wood, K. L., 2002, “A Quantitative Similarity Metric for Design-by-Analogy,” ASME Journal of Mechanical Design (JMD), 124(2), pp. 173–182.

[35] Agyemang, M., Linsey, J., and Turner, C. J., 2017, “Transforming Functional Models to Critical Chain Models via Expert Knowledge and Automatic Parsing Rules for Design Analogy Identification,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 31(4), pp. 501–511.

[36] Liu, L., Li, Y., Xiong, Y., and Cavallucci, D., 2020, “A New Function-Based Patent Knowledge Retrieval Tool for Conceptual Design of Innovative Products,” Computers in Industry, 115, p. 103154.

[37] Mathieson, J. L., Shanthakumar, A., Sen, C., Arlitt, R., Summers, J. D., and Stone, R., 2011, “Complexity as a Surrogate Mapping between Function Models and Market Value,” *Proceedings of the ASME Design Engineering Technical Conference*, American Society of Mechanical Engineers Digital Collection, pp. 55–64.

[38] Gill, A. S., Summers, J. D., and Turner, C. J., 2017, “Comparing Function Structures and Pruned Function Structures for Market Price Prediction: An Approach to Benchmarking Representation Inferencing Value,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 31(4), pp. 550–566.

[39] Mokhtarian, H., Coatanéa, E., and Paris, H., 2017, “Function Modeling Combined with Physics-Based Reasoning for Assessing Design Options and Supporting Innovative Ideation,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 31(4), pp. 476–500.

[40] Murphy, A. R., Banks, H. D., Nagel, R. L., and Linsey, J. S., 2019, “Graduate Students’ Mental Models: An Investigation into the Role of Function in Systems Understanding,” *Proceedings of the ASME Design Engineering Technical Conference*, American Society of Mechanical Engineers (ASME).

[41] Iwasaki, Y., Vescovi, M., Fikes, R., and Chandrasekaran, B., 1995, “Causal Functional Representation Language with Behavior-Based Semantics,” Applied Artificial Intelligence an International Journal, 9(1), pp. 5–31.

[42] Lai, K., and Wilson, W. R. D., 1989, "FDL—A Language for Function Description and Rationalization in Mechanical Design," Journal of Mechanisms Transmissions and Automation in Design, 111(1), pp. 117–123.

[43] Hundal, M. ., 1990, "A Systematic Method for Developing Function Structures, Solutions and Concept Variants," Mechanism and Machine Theory, 25(3), pp. 243–256.

[44] Murdock, J. W., Szykman, S., and Sriram, R. D., 1997, "An Information Modeling Framework to Support Design Databases and Repositories," 1997 ASME Design Engineering Technical Conferences, 97, pp. 14–17.

[45] Kurfman, M. A., Stone, R. B., Rajan, J. R., and Wood, K. L., 2001, "Functional Modeling Experimental Studies," *Proceedings of DETC2001, DETC2001/DTM-21709*, Pittsburgh, PA.

[46] Sridharan, P., and Campbell, M. I., 2004, "A Grammar for Function Structures," *Proc. of the ASME 2004 Intl. Design Engineering Technical Conf. and Computers and Information in Engineering Conf., IDETC/CIE'04*, pp. 41–55.

[47] Sridharan, P., and Campbell, M. I., 2005, "A Study on the Grammatical Construction of Function Structures," Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 19(3), pp. 139–160.

[48] Kirschman, C. F., Fadel, G. M., and Jara-Almonte, C. C., 1998, "Classifying Functions for Mechanical Design," Journal of Mechanical Design, 120(3), pp. 475–482.

[49] Bohm, M. R., Vucovich, J. P., and Stone, R. B., 2008, "Using a Design Repository to Drive Concept Generation," Journal of Computing and Information Science in Engineering, 8(1).

[50] Bohm, M. R., and Stone, R. B., 2004, "Product Design Support: Exploring a Design Repository System," *ASME 2004 International Mechanical Engineering Congress and Exposition*, ASME, pp. 55–65.

[51] Bohm, M. R., Vucovich, J. P., and Stone, R. B., 2005, "Capturing Creativity: Using a Design Repository to Drive Concept Innovation," *ASME 2005 IDETC/CIE*, ASME, Long Beach, California, September 24–28, 2005, DETC2005-85105, pp. 331–342.

[52] Caldwell, B. W., Thomas, J. E., Sen, C., Mocko, G. M., and Summers, J. D., 2012, "The Effects of Language and Pruning on Function Structure Interpretability," Journal of Mechanical Design, Transactions of the ASME, 134(6).

[53] Caldwell, B. W., Sen, C., Mocko, G. M., and Summers, J. D., 2011, "An Empirical Study of the Expressiveness of the Functional Basis," Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 25(03), pp. 273–287.

[54] Sen, C., Caldwell, B. W., Summers, J. D., and Mocko, G. M., 2010, "Topological Information Content and Expressiveness of Function Models in Mechancial Design," Journal of Computing and Information Science in Engineering, 10(3), pp. 381–394.

[55] Volker, J., 2009, *Learning Expressive Ontologies*, IOS Press.

[56] Kurtoglu, T., Swantner, A., and Campbell, M. I., 2010, "Automating the Conceptual Design Process: 'From Black Box to Component Selection,'" Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 24(1), pp. 49–62.

[57] Mao, X., and Sen, C., 2018, "Physics-Based Semantic Reasoning for Function Model Decomposition," ASME International.

[58] McAdams, D. A., Stone, R. B., and Wood, K. L., 1999, "Functional Interdependence and Product Similarity Based on Customer Needs," Research in Engineering Design - Theory, Applications, and Concurrent Engineering, 11(1), pp. 1–19.

[59] Gill, A. S., and Sen, C., 2020, "Evolutionary Approach to Function Model Synthesis: Development of Parameterization and Synthesis Rules," *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE2020*, Virtual, Online.

[60] Mikes, A., Edmonds, K., Stone, R. B., and Dupont, B., 2020, "Optimizing an Algorithm for Data Mining a Design Repository to Automate Functional Modeling," *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE2020* , Virtual, Online.

[61] Gero, J. S., and Kannengiesser, U., 2000, "Towards a Situated Function-Behaviour-Structure Framework as the Basis of a Theory of Designing," *Workshop on Development and Application of Design Theories in AI in Design Research, Sixth International Conference on Artificial Intelligence in Design*, Worcester, MA.

[62] Mao, X., and Sen, C., 2019, "Semantic and Qualitative Physics-Based Reasoning on Plain-English Flow Terms for Generating Function Model Alternatives," Journal of Computing and Information Science in Engineering, 20(4), p. 041006.

[63] Sen, C., Summers, J. D., and Mocko, G. M., 2011, “Exploring Potentials for Conservational Reasoning Using Topologic Rules of Function Structure Graphs,” Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, 9, pp. 377–388.

[64] Umeda, Y., Ishii, M., Yoshioka, M., Shimomura, Y., and Tomiyama, T., 1996, “Supporting Conceptual Design Based on the Function-Behavior-State Modeler,” Artificial Intelligence for Engineering, Design, Analysis and Manufacturing, 10(04), pp. 275–288.

[65] Chen, Y., Zhang, Z., Huang, J., and Xie, Y., 2013, “Toward a Scientific Ontology Based Concept of Function,” Artificial Intelligence for Engineering Design, Analysis and Manufacturing: AIEDAM, 27(3), pp. 241–248.

[66] Xu, C., Gupta, S. K., Yao, Z., Gruninger, M., and Sriram, R. D., 2005, “Toward Computer-Aided Conceptual Design of Mechatronic Devices with Multiple Interaction-States,” *ASME 2005 IDETC/CIE*, Long Beach, CA, September 24-28, 2005, DETC2005-85410, pp. 455–467.

[67] Umarikar, A. C., and Umanand, L., 2005, “Modelling of Switching Systems in Bond Graphs Using the Concept of Switched Power Junctions,” Journal of the Franklin Institute, 342(2), pp. 131–147.

[68] Mosterman, P. J., and Biswas, G., 1994, *Behavior Generation Using Model Switching A Hybrid Bond Graph Modeling Technique*.

[69] Hrovat, D., and Tobler, W. E., 1991, “Bond Graph Modeling of Automotive Power Trains,” Journal of the Franklin Institute, 328(5–6), pp. 623–662.

[70] Deur, J., Ivanović, V., Assadian, F., Kuang, M., Tseng, E. H., and Hrovat, D., *Bond Graph Modeling of Automotive Transmissions and Drivelines*.

[71] Eisenbart, B., Gericke, K., and Blessing, L., 2013, “Adapting the IFM Framework to Functional Approaches across Disciplines,” *Proceedings of the International Conference on Engineering Design, ICED*, pp. 163–172.

[72] Buur, J., Andreasen, and Myrup, M., 1990, *A Theoretical Approach to Mechatronics Design*, Technical University of Denmark, Lyngby, Denmark.

[73] Nagel, R. L., Stone, R. B., Hutcheson, R. S., McAdams, D. A., and Donndelinger, J. A., 2008, “Function Design Framework (FDF): Integrated Process and Function Modeling for Complex Systems,” *ASME 2008 IDETC/CIE*, pp. 273–286.

[74] Liu, C., Hildre, H. P., Zhang, H., and Rølvåg, T., 2015, “Conceptual Design of Multi-Modal Products,” Research in Engineering Design, 26(3), pp. 219–234.

[75] Sen, C., Summers, J. D., and Mocko, G. M., 2012, "A Formal Representation of Function Structure Graphs for Computer-Directed Modeling and Conservation-Based Reasoning," ASME Journal of Computing and Information Science in Engineering, JCISE, 13(2), p. 21001.

[76] Sen, C., 2016, "Feature-Based Computer Modeling and Reasoning on Mechanical Functions," *Proceedings of the IDETC/CIE*, Charlotte, North Carolina, August 21-24, 2016, ASME Paper No. DETC2016-60353, p. V01BT02A008.

[77] Sen, C., 2011, *A Formal Representation of Mechanical Functions to Support Physics-Based Computational Reasoning in Early Mechanical Design*, Clemson University.

[78] A Venkatanarasimhan, L. N., and Chowdhury, A., 2020, "A Vocabulary of Function Features for Computer Aided Modeling of Thermal-Fluid Systems," *Proceedings of the ASME 2020 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference IDETC/CIE2020*, Virtual, Online.

[79] Venkatanarasimhan, L. N. A., Mao, X., Chowdhury, A., and Sen, C., 2019, "Physics-Based Function Features for a Set of Material-Processing Verbs," *Proceedings of the ASME Design Engineering Technical Conference*, American Society of Mechanical Engineers (ASME).

[80] Sen, C., 2016, "Feature-Based Computer Modeling and Reasoning on Mechanical Functions," *Proceedings of the ASME Design Engineering Technical Conference*, American Society of Mechanical Engineers (ASME), Charlotte, North Carolina, August 21–24, 2016, DETC2016-60353, p. V01BT02A008.

[81] Venkatanarasimhan, L. N. A., Mao, X., Chowdhury, A., and Sen, C., 2019, "Physics-Based Function Features for a Set of Material-Processing Verbs," *ASME 2019 IDETC/ CIE*, Anaheim, California, August 18–21, 2019, DETC2019-98343, p. V001T02A031.

[82] Sen, C., Summers, J. D., and Mao, X., 2019, "A Physics-Based Formal Vocabulary of Energy Verbs for Function Modeling," *ASME 2019 IDETC/ CIE*, Anaheim, California, August 18–21, 2019 , DETC2019-98502, p. V001T02A069.

[83] Mao, X., 2019, "Semantic and Qualitative Physics-Based Formal Reasoning for Functional Decomposition in Mechanical Design," Florida Institute of Technology.

[84] Hopcroft, J., Motwani, R., and Ullman, J., 2008, *Introduction to Automata Theory, Languages, and Computation*, Pearson Education India, New York, NY, USA.

[85] Sen, C., Summers, J., and Mocko, G., 2011, *A Protocol to Formalise Function Verbs to Support Conservation-Based Model Checking*.

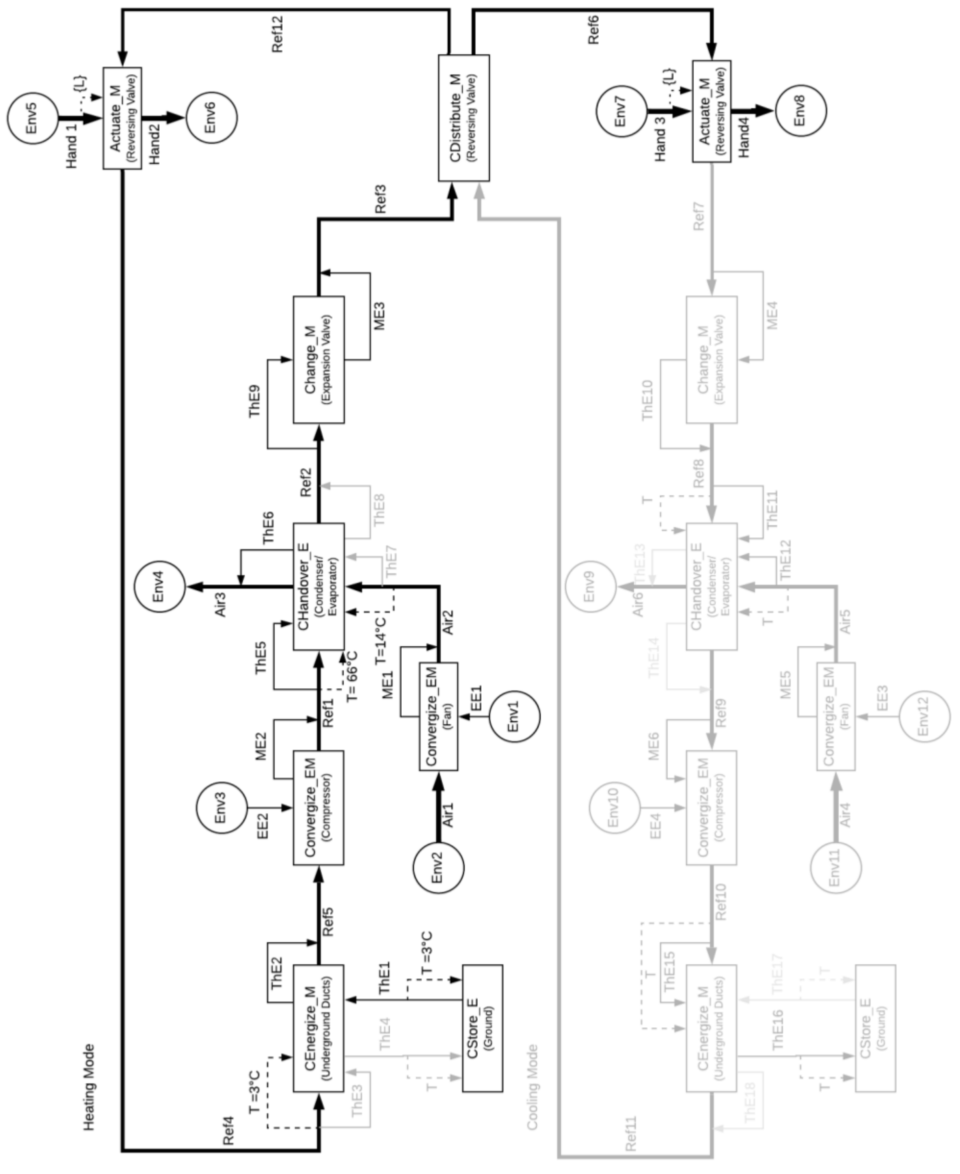
[86] Summers, J. D., and Mocko, G. M., 2013, "Of Function Structure Graphs for Physics-Based Reasoning," 13(June), pp. 1–13.

[87] Teig, N., and Scherer, R., 2016, "Bringing Formal and Informal Reasoning Together-a New Era of Assessment?," Frontiers in Psychology, 7(JUL), p. 1097.

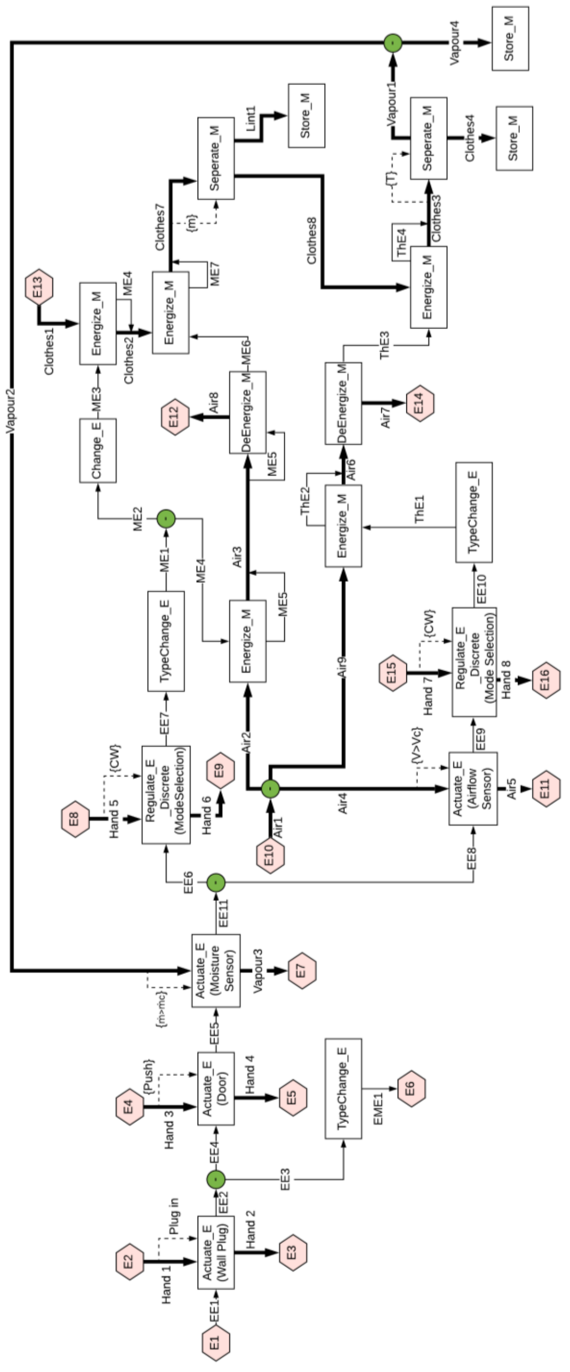
[88] Nagel, R. L., Vucovich, J. P., Stone, R. B., and Mcadams, D. A., 2008, "A Signal Grammar to Guide Functional Modeling of Electromechanical Products," Journal of Mechanical Design, 130(5).

[89] Sen, C., Summers, J. D., and Mocko, G. M., 2011, "A Protocol to Formalise Function Verbs to Support Conservation-Based Model Checking," Journal of Engineering Design, 22(11–12), pp. 765–788.

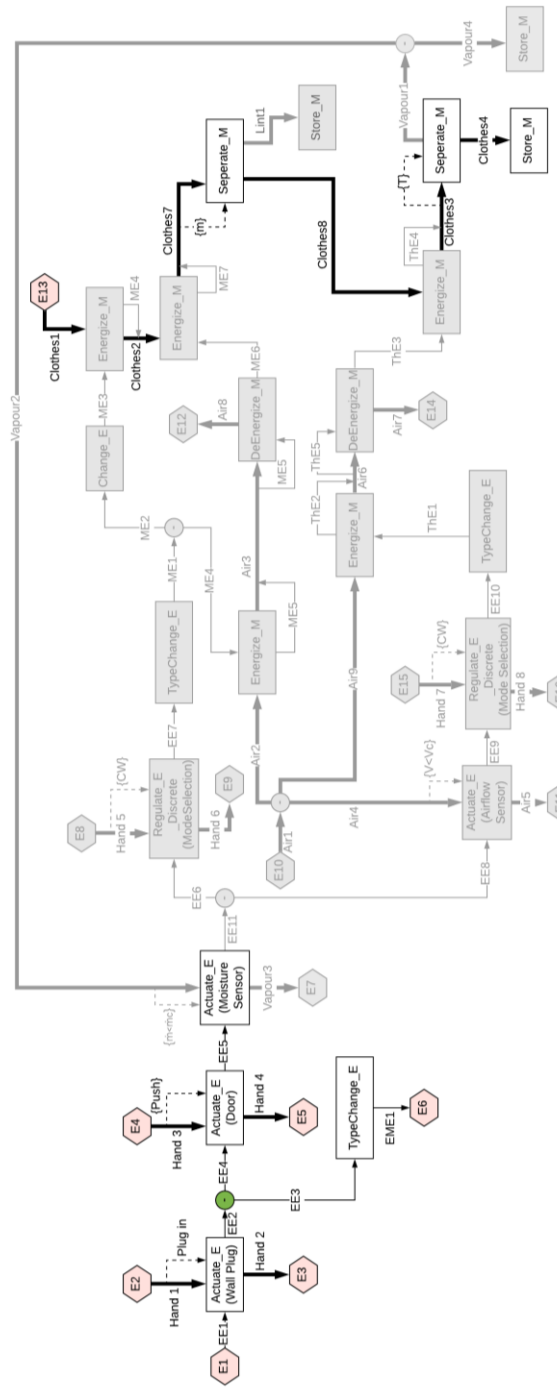
Appendix A
Multi-Mode Function Models



Geothermal Heatpump in Both Heating and Cooling Mode



Function Model of a Clothes Dryer



Function Model of Clothes Dryer When Drying is Complete

Appendix B
Header files for ConMod 2.0

```
#pragma once
#include "Template.h"
#include <fstream>
#include <iostream>
#include <iomanip>
#include <vector>
#include <string>
#include <algorithm>
#include <iterator>
#include <sstream>

// ActuateE_Template dialog
class ActuateE_Template : public CDialogEx, public CTemplate
{
        DECLARE_DYNAMIC(ActuateE_Template)

public:
        ActuateE_Template(CWnd* pParent = NULL,
                CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL,
                CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE = NULL,
                CString* pCounterString_InS = NULL,
                CString* pCounterString_InCarrier = NULL,
                CString* pCounterString_OutCarrier = NULL, int ReasOpt = 0);   //
standard constructor
        virtual ~ActuateE_Template();


// Dialog Data
#ifdef AFX_DESIGN_TIME
        enum { IDD = IDD_ACTUATE_E_TEMPLATE };
#endif

protected:

public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE;
        CSignal* pSignal_InS;
        CEnergy* pEnergy_In_CarrierE;
        CEnergy* pEnergy_Out_CarrierE;
        CMaterial* pMaterial_In_CarrierM;
        CMaterial* pMaterial_Out_CarrierM;
        int ReasoningOption;
        bool CarrierIsMaterial = false;
        bool CarrierIsEnergy = false;
        CString InitialState;
        CString CurrentState = NULL;
        CString CurrentCS;
        int Mode;
        enum { Actuate, DeActuate };
        int Row, Coloumn;
        // individual name and value elements
        HWND hWnd;
        //void open_file(HWND hWnd);

        //bool StateRecognized = false, CSRecognized = false;
        //void OnFileOpen(CString CSinput, CString State0);
        bool EisActuated = false;
        CString* EInID = new CString;
        CString* EOutID = new CString;
        void ModeIsActuate();
        void ModeIsDeActuate();
        CPoint EInHP, EInTP, EOutHP, EOutTP;

        DECLARE_MESSAGE_MAP()
        afx_msg void OnBnClickedCarrierM();
        afx_msg void OnBnClickedCarrierE();
        afx_msg void OnBnClickedOk();
};
```

```
#pragma once
#include "Template.h"

// Conduct_E_Template dialog
class CConduct_E_Template : public CDialog, public CTemplate
{
        DECLARE_DYNAMIC(CConduct_E_Template)

public:
        CConduct_E_Template(CWnd* pParent = NULL, CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL, CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE = NULL, CString* pCounterString_OutE_Res =
NULL, int ReasOpt = 0);

        virtual ~CConduct_E_Template();

// Dialog Data

        enum { IDD = IDD_CONDUCT_E_TEMPLATE };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()


public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE;
        CEnergy* pEnergy_OutE_Res;
        int ReasoningOption;
};
```

```
#include "Function.h"
#include "Env.h"
#include "Edge.h"
#include "Material.h"
#include "Energy.h"
#include "Signal.h"
#include "Convert_E_Template.h"
#include "Conduct_E_Template.h"
#include "Energize_M_Template.h"
#include "Distribute_E_Template.h"
#include "DeEn_M_Template.h"
#include "ActuateE_Template.h"
#pragma once

class CConMod2Doc : public CDocument
{
protected: // create from serialization only
        CConMod2Doc();
        DECLARE_DYNCREATE(CConMod2Doc)

public:
        CList<CElement*, CElement*> CElementList;                // list of all elements
of all types - reconstructed everytime OnDraw is called
        CList<CNode*, CNode*> CNodeList; // List of Function blocks - appended upon
ADD_FUNCTION, removed upon DELETE
        CList<CEdge*, CEdge*> CEdgeList;                          // List of flow arrows
(edges) of all kinds - appended upon ADD_EDGE, removed upon DELETE
        CList<CElement*, CElement*> PreselectionList;

        CList<CFunction*, CFunction*> CFunctionList;
        CList<CEnv*, CEnv*> CEnvList;
        CList<CMaterial*, CMaterial*> CMaterialList;
        CList<CMaterial*, CMaterial*> CMaterialList_IN_TEMP;
        CList<CMaterial*, CMaterial*> CMaterialList_OUT_TEMP;
        CList<CEnergy*, CEnergy*> CEnergyList;
        CList<CEnergy*, CEnergy*> CEnergyList_IN_TEMP;
        CList<CEnergy*, CEnergy*> CEnergyList_OUT_TEMP;
        CList<CSignal*, CSignal*> CSignalList;
        CList<CSignal*, CSignal*> CSignalList_IN_TEMP;
        CList<CSignal*, CSignal*> CSignalList_OUT_TEMP;


        CList<CTemplate*, CTemplate*> CTemplateList;
        // List of all templates of Layer 2

                         // The main purpose of this list is to store the "template"
instances, while the              individual

                         // elements in the templates, such as functions and flows, are
stored in the    CElementList.

                         // By storing the templates in this separate list, it will be
easier to delete         them

                         // during application exit (Destructor of the View class).

        CList<CFunction*, CFunction*> CConvert_E_Function_List;
        CList<CConvert_E_Template*, CConvert_E_Template*> CConvert_E_Template_List;
        CList<CFunction*, CFunction*> CConduct_E_Function_List;
        CList<CConduct_E_Template*, CConduct_E_Template*> CConduct_E_Template_List;

        CList<CFunction*, CFunction*> CEnergize_M_Function_List;
        CList<CEnergize_M_Template*, CEnergize_M_Template*> CEnergize_M_Template_List;

        CList<CFunction*, CFunction*> CDistribute_E_Function_List;
        CList<CDistribute_E_Template*, CDistribute_E_Template*>
        CDistribute_E_Template_List;

        CList<CFunction*, CFunction*> CDeEn_M_Function_List;
        CList<CDeEn_M_Template*, CDeEn_M_Template*> CDeEn_M_Template_List;

        CList<CFunction*, CFunction*> ActuateE_Function_List;
        CList<ActuateE_Template*, ActuateE_Template*> ActuateE_Template_List;
        CList<ActuateE_Template*, ActuateE_Template*> CurrState_List;

// Attributes
public:

// Operations
public:

// Overrides
public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
#ifdef SHARED_HANDLERS
        virtual void InitializeSearchContent();
        virtual void OnDrawThumbnail(CDC& dc, LPRECT lprcBounds);
#endif // SHARED_HANDLERS

// Implementation
public:
        virtual ~CConMod2Doc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

// Generated message map functions
protected:
        DECLARE_MESSAGE_MAP()

#ifdef SHARED_HANDLERS
        // Helper function that sets search content for a Search Handler
        void SetSearchContent(const CString& value);
#endif // SHARED_HANDLERS
};
```

```
#pragma once
#include "afxmt.h"
#include "geometry.h"
```

```
#define SELECTION_RADIUS 20
class CConMod2View : public CView, public CGeometry
{
protected: // create from serialization only
        CConMod2View();
        DECLARE_DYNCREATE(CConMod2View)

        // Attributes
public:
        CConMod2Doc* GetDocument() const;

        // Operations
public:

        // Overrides
public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

        // Implementation
public:
        virtual ~CConMod2View();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif

protected:

        //===========================================================================
        //===========================================================================
        // END OF WIZARD-GENERATED CODE
        //===========================================================================
        //===========================================================================

        //===========================================================================
        // SELECTION OF REASONING OPTIONS - THREE MAIN TYPES
        //===========================================================================

public:
        int ReasoningOption;
        enum {
                QUALITATIVE_CONSERVATION,
                QUALITATIVE_IRREVERSIBILITY,
                QUANTITATIVE_EFFICIENCY,
                QUANTITATIVE_POWERREQUIRED
        };

        //===========================================================================
        // SELECTION OF MESSAGE HANDLER FUNCTIONS THROUGH ENUMERATED WHAT-TO-DO LIST
        //===========================================================================

public:
        int WhatToDo;
        enum
        {
                ESCAPE,
                ADD_FUNCTION,
                ADD_MATERIAL,
                ADD_ENERGY,
                ADD_SIGNAL,
                ADD_ENV,
                ADD_CONVERT_E_TEMPLATE,
                ADD_CONDUCT_E_TEMPLATE,         // Add more todo items here
                ADD_ENERGIZE_M_TEMPLATE,        // Add more todo items here
                ADD_DISTRIBUTE_E_TEMPLATE,              // Add more todo items here
                ADD_DEEN_M_TEMPLATE,            // Add more todo items here
                ADD_ACTUATEE_TEMPLATE           // Add more todo items here

        };

        //============================================================================
        // MAIN MENU - REASONING OPTION SELECTION MESSAGE HANDLER FUNCTION
        //============================================================================

public:
        afx_msg void OnQualitativeConservation();
        afx_msg void OnQualitativeIrreversibility();
        afx_msg void OnQuantitativeEfficiency();
        afx_msg void OnQuantitativePowerRequired();

        //============================================================================
        // PRIMITIVES TOOLBAR MESSAGE HANDLER FUNCTIONS
        //============================================================================

public:
        void Handler_SaveFile(void);

public:
        void Handler_AddFunction(void);
        void Handler_AddMaterial(void);
        void Handler_AddEnergy(void);
        void Handler_AddSignal(void);
        void Handler_AddEnv(void);
        void Handler_EditCut(void);

        //============================================================================
        // FEATURES TOOLBAR MESSAGE HANDLER FUNCTIONS
        //============================================================================

public:
        void Handler_AddConvert_E_Template(void);
        void Handler_AddConduct_E_Template(void);
        void Handler_AddEnergize_M_Template(void);
        void Handler_AddDistribute_E_Template(void);
        void Handler_AddDeEn_M_Template(void);
        void Handler_AddActuateE_Template(void);

        //============================================================================
        // REASONING TOOLBAR MESSAGE HANDLER FUNCTIONS
        //============================================================================

public:
        void Handler_Qualitative(void);
        void Handler_Quantitative(void);
        void Handler_Causal(void);

        //==========================================================================
        // FUNCTIONS FOR ADDING INSTANCES TO THE MODEL
        //==========================================================================
public:
        int Counter_F;
        int Counter_Env;
        int Counter_M;
        int Counter_E;
        int Counter_S;
        CString CounterString;

public:
        void AddFunction(void);
        void AddMaterial(void);
        void AddEdge_Dynamic(void);
        void AddEnergy(void);
        void AddSignal(void);
        void AddEnv(void);

public:
        void AddConvert_E_Template(void);
        void AddConduct_E_Template(void);
        void AddEnergize_M_Template(void);
        void AddDistribute_E_Template(void);
        void AddDeEn_M_Template(void);
        void AddActuateE_Template(void);


        // The following four members are used during construction of the dynamic
        // instance of edges, and to pass their values to the final instance.
        CElement* pTailElemDynamic;
        CElement* pHeadElemDynamic;
        bool TailNodeSelected;

        //==========================================================================
        // FUNCTIONS FOR SELECTING INSTANCES FROM THE MODEL TO DO EDIT OPERATIONS
        //==========================================================================

public:
        void Preselect(CPoint* pMouseTip);                // Preselection of elements by
mouse hover
        void Highlight(CElement* pElement);      // Change color when preselected
        void UnHighlight(CElement* pElement);    // Reset color when released from
preselection
        void SelectElement(CElement* pElement);  // Finally select one element from the
presel list
        void ScrollThroughPreselection();                 // Scrolling through
preselected elements
        POSITION ScrollPosition;                 // Current position within
PreselectionList that is selected
        enum { NONE, TAIL, CENTER, HEAD };                // Grab handle locations
        CElement* pElementToBeDeleted;
        CElement* pSelectedElement;                       // Pointer to store the
currently selected element
```

```
//==============================================================================
// FUNCTIONS FOR STORING OBJECT POINTERS AND DETERMINING THEIR TYPE

//==============================================================================
// For basic elements (Layer 1) - Node, Edge, Function, Env, M, E, and S

bool ElementIsNode(CElement* pElement);
// TRUE if pSelectedElement is a member of CNodeList
bool ElementIsFunction(CElement* pElement);
// TRUE if pSelectedElement is a member of CFunctionList
bool ElementIsEnv(CElement* pElement);
// TRUE if pSelectedElement is a member of CEnvList
bool ElementIsEdge(CElement* pElement);
// TRUE if pSelectedElement is a member of CEdgeList
bool ElementIsMaterial(CElement* pElement);
// TRUE if pSelectedElement is a member of CEdgeList
bool ElementIsEnergy(CElement* pElement);
// TRUE if pSelectedElement is a member of CEdgeList
bool ElementIsSignal(CElement* pElement);
// TRUE if pSelectedElement is a member of CEdgeList

POSITION NodeIndexInNodeList;
// Gets set by SelectedElementIsNode so that it could be removed
POSITION FunctionIndexInFunctionList;
// Gets set by SelectedElementIsFunction so that it could be removed
POSITION EnvIndexInEnvList;
// Gets set by SelectedElementIsEnv so that it could be removed
POSITION EdgeIndexInEdgeList;
// Gets set by SelectedElementIsEdge so that it could be removed
POSITION MaterialIndexInMaterialList;
// Gets set by SelectedElementIsEdge so that it could be removed
POSITION EnergyIndexInEnergyList;
// Gets set by SelectedElementIsEdge so that it could be removed
POSITION SignalIndexInSignalList;
// Gets set by SelectedElementIsEdge so that it could be removed

bool ElementIsConvert_E_Function(CElement* pElement);
bool ElementIsConvert_E_Template(CElement* pElement);

bool ElementIsConduct_E_Function(CElement* pElement);
bool ElementIsConduct_E_Template(CElement* pElement);

bool ElementIsEnergize_M_Function(CElement* pElement);
bool ElementIsEnergize_M_Template(CElement* pElement);

bool ElementIsDistribute_E_Function(CElement* pElement);
bool ElementIsDistribute_E_Template(CElement* pElement);

bool ElementIsDeEn_M_Function(CElement* pElement);
bool ElementIsDeEn_M_Template(CElement* pElement);

bool ElementIsActuateE_Function(CElement* pElement);
bool ElementIsActuateE_Template(CElement* pElement);

POSITION Convert_E_Function_IndexInConvert_E_Function_List;
POSITION Convert_E_Template_IndexInConvert_E_Template_List;

POSITION Conduct_E_Function_IndexInConduct_E_Function_List;
```

```
        POSITION Conduct_E_Template_IndexInConduct_E_Template_List;

        POSITION Energize_M_Function_IndexInEnergize_M_Function_List;
        POSITION Energize_M_Template_IndexInEnergize_M_Template_List;

        POSITION Distribute_E_Function_IndexInDistribute_E_Function_List;
        POSITION Distribute_E_Template_IndexInDistribute_E_Template_List;

        POSITION DeEn_M_Function_IndexInDeEn_M_Function_List;
        POSITION DeEn_M_Template_IndexInDeEn_M_Template_List;

        POSITION ActuateE_Function_IndexInDeActuateE_Function_List;
        POSITION ActuateE_Template_IndexInActuateE_Template_List;


        void EmptyAllTempLists();
        //==============================================================================
        // FUNCTIONS FOR EDIT OPERATIONS ON INSTANCES WITHIN THE MODEL
        //==============================================================================
public:
        void MoveConnectDynamic();
        void MoveConnect();
        void DetachEdgesFromElement(CElement* pElement);
        void DeleteElement(CElement* pElement);
        // The following four members stores the topology of a flow terminal
        // (head or tail) that is moved by the MoveConnectDynamic function to a temp
        // storage, so that the point can be reassigned in the case the operation
        // was illegal.  The storage code is in the MoveConnectDynamic function.
        // The reassignment code is in OnDraw (during gramamr chekcs).
        CElement* pRememberHeadElement;
        CPoint RememberHeadPoint;
        CElement* pRememberTailElement;
        CPoint RememberTailPoint;

        //==============================================================================
        // PARAMETERS AND FLAGS FOR CONTROLLING AND SIGNALLING MOUSE POINTS AND BUTTONS
        //==============================================================================

public:
        //  Parameters
        CPoint MouseLDownPoint;
        CPoint MouseLUpPoint;
        CPoint MouseRDownPoint;
        CPoint MouseRUpPoint;
        CPoint MouseMovePoint;

        // Flags
        bool LButtonIsDown;
        bool RButtonIsDown;

        //==============================================================================
        // MESSAGE HANDLING FUNCTIONS FOR MOUSE EVENTS
        //==============================================================================

public:                                 // Mouse Button and Move Functions
        afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnRButtonDown(UINT nFlags, CPoint point);
        afx_msg void OnRButtonUp(UINT nFlags, CPoint point);
        afx_msg void OnMouseMove(UINT nFlags, CPoint point);
        afx_msg void OnMButtonUp(UINT nFlags, CPoint point);

public:
        afx_msg BOOL OnEraseBkgnd(CDC* pDC);     // Flicker elimination
        afx_msg void OnLButtonDblClk(UINT nFlags, CPoint point);

        //============================================================================
        // CONSERVATION CHECKING FUNCTIONS - TOPOLOGICAL CONSERVATION (WITHOUT VOCAB)
        // REFERE TO:  ICED-2011 PAPER
        //============================================================================

        bool GrammarCheckRequired;

        CString Msg_OrphanFlow;
        CString Msg_BarrenFlow;
        CString Msg_OneInManyOut_M;
        CString Msg_OneInManyOut_E;
        CString Msg_ManyInOneOut_M;
        CString Msg_ManyInOneOut_E;
        CString Msg_ManyInManyOut;
        CString Msg_MissingResidualEnergy;
        CString Msg_MaterialChangeWithoutEnergy;
        CString StartState = NULL;
        CString StartingState;
        bool SignalIsChanged = false;
        void Set_OrphanFlowMsg();
        void Set_BarrenFlowMsg();
        void Set_OneInManyOutMsg_M();
        void Set_OneInManyOutMsg_E();
        void Set_ManyInOneOutMsg_M();
        void Set_ManyInOneOutMsg_E();
        void Set_ManyInManyOutMsg();
        void Set_MissingResidualEnergyMsg();
        void Set_MaterialChangeWithoutEnergyMsg();
        void ComposeQualitativeMessage();

        //============================================================================
        // Quantitative Reasoning Methods
        //============================================================================

        void VerifyPositivePowerOfFlows();
        void VerifyEnergyBalanceOfFunctions();
        void ComputeEfficiency();
        void ComposeQuantitativeMessage();
        bool ContinueReasoning;
        void ComposeCausalMessage();
        //int DetermineState(CString CSinput);
        int DetermineState(CString CSinput, ActuateE_Template* actuatedfunc);
        void CausalReasoning(ActuateE_Template* actuatedfunc);
        int Mode;
        enum { Actuate, DeActuate };
        bool EisActuated = false;


        // Generated message map functions
protected:
        afx_msg void OnFilePrintPreview();
        afx_msg void OnContextMenu(CWnd* pWnd, CPoint point);
        DECLARE_MESSAGE_MAP()

};
```

```
#pragma once
#include "Template.h"

// Convert_E_Template dialog
class CConvert_E_Template : public CDialog, public CTemplate
{
        DECLARE_DYNAMIC(CConvert_E_Template)

public:
        CConvert_E_Template(CWnd* pParent = NULL, CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL, CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE = NULL, CString* pCounterString_OutE_Res =
NULL, int ReasOpt = 0);

        virtual ~CConvert_E_Template();

// Dialog Data
        enum { IDD = IDD_CONVERT_E_TEMPLATE };


protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE;
        CEnergy* pEnergy_OutE_Res;
        int ReasoningOption;
};
```

```
#pragma once
#include "Template.h"

// DeEn_M_Template dialog
class CDeEn_M_Template : public CDialog, public CTemplate
{
        DECLARE_DYNAMIC(CDeEn_M_Template)

public:
        CDeEn_M_Template(CWnd* pParent = NULL,
                CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL,
                CString* pCounterString_InM = NULL,
                CString* pCounterString_OutM = NULL,
                CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE = NULL, int ReasOpt = 0);   // standard
constructor
```

```
        virtual ~CDeEn_M_Template();

// Dialog Data
        enum { IDD = IDD_DEEN_M_TEMPLATE };


protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE;
        CMaterial* pMaterial_InM;
        CMaterial* pMaterial_OutM;
        int ReasoningOption;
};
```

```
#pragma once
#include "Template.h"

// Distribute_E_Template dialog
class CDistribute_E_Template : public CDialog, public CTemplate
{
        DECLARE_DYNAMIC(CDistribute_E_Template)

public:
public:
        CDistribute_E_Template(CWnd* pParent = NULL,
                CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL,
                CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE1 = NULL,
                CString* pCounterString_OutE2 = NULL, int ReasOpt = 0);   // standard
constructor

        virtual ~CDistribute_E_Template();


// Dialog Data
        enum { IDD = IDD_DISTRIBUTE_E_TEMPLATE };


protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE1;
        CEnergy* pEnergy_OutE2;
        int ReasoningOption;
};
```

```
#pragma once
#include "element.h"
//#include "math.h"
#include "node.h"

#define EDGE_HEAD_SIZE 20
#define EDGE_HEAD_HALF_ANGLE 0.25 // Radians
class CEdge : public CElement
{
public:
        CEdge(void);
        CEdge(CPoint TailClick, CPoint HeadClick);
        ~CEdge(void);

        // Head construction data
        CPoint HeadLeftVertex, HeadRightVertex;
        double HeadSize, HalfHeadAngle;
        CPoint HeadVertexArray[3];

        // Topological information
        void ComputeAnchorPoints();
        void AttachEdgeToNearestAnchor();
        void ResetGeometricCenter();     // Makes sure that the GeometricCenter is reset
between the
                                                                           // Tail and
Head points, when an arrow is moved by grabbing
                                                                           // Those
terminal points
        bool ThisFlowIsIncomingBaggage;
        bool ThisFlowIsOutgoingBaggage;

        // Drawing data
        int StemThickness;
        int StemLineFont;
        enum { NONE, THIN, MEDIUM, THICK };
        int FontSize;
        void DrawOnDC(CDC* pDC);
};
```

```
#pragma once
#include "geometry.h"

#define GENERIC_PEN_R 0
#define GENERIC_PEN_G 0
#define GENERIC_PEN_B 0

#define GENERIC_BRUSH_R 0
#define GENERIC_BRUSH_G 0
#define GENERIC_BRUSH_B 0

#define DANGLING_BRUSH_R 255
#define DANGLING_BRUSH_G 0
#define DANGLING_BRUSH_B 0

#define PRESELECTION_PEN_R 200
#define PRESELECTION_PEN_G 0
#define PRESELECTION_PEN_B 200
```

```
#define HIDDEN_PEN_R 220
#define HIDDEN_PEN_G 220
#define HIDDEN_PEN_B 220

#define SELECTION_PEN_R 0
#define SELECTION_PEN_G 200
#define SELECTION_PEN_B 0

#define RESIDUAL_PEN_R 255
#define RESIDUAL_PEN_G 0
#define RESIDUAL_PEN_B 0

#define GENERIC_FONT_SIZE 16
#define BAGGAGE_FONT_SIZE 12

class CElement :
        public CGeometry/*, public CDialog*/
{
public:
        CElement(void);
        virtual ~CElement(void); // Must be virtual, so that individual desctrutors of
                                                                // the derived classes
are called when CConModView's
                                                                // destructor tries to
close the session

                                                                // PARAMETERS OVERRIDEN
IN BOTH CNode AND CEdge CLASSES
        bool IsHighlighted;
        bool IsSelected;
        bool IsResidual;
        bool IsHidden;

        CPoint GeometricCenter;
        CPoint Anchors[16];
        CPoint AnchorsForBaggageFlows[16];

        //CString GivenName;             // Unnecessary - the individual classes need
their own
        // GivenName attribute, because the dilaog constructor
        // needs a GivenName that is not inherited.


        int PenR, PenG, PenB;
        int BrushR, BrushG, BrushB;

        int GrabHandle;          // Stores where (Head, Tail, Center) an element is
grabbed by the mouse
        virtual void DrawOnDC(CDC* pDC);

        //int ReasoningOption;
        enum {
                QUALITATIVE_CONSERVATION,
                QUALITATIVE_IRREVERSIBILITY,
                QUANTITATIVE_EFFICIENCY,
                QUANTITATIVE_POWERREQUIRED
        };
        // PARAMETERS OVERRIDEN IN CEDGE:  TOPOLOGY DATA
        CPoint TailPoint, HeadPoint;
        int HeadBrushR, HeadBrushG, HeadBrushB;
        int TailBrushR, TailBrushG, TailBrushB;
        CElement* pHeadElem;
        CElement* pTailElem;
};
```

```
#pragma once
#include "Template.h"


// Energize_M_Template dialog
class CEnergize_M_Template : public CDialog, public CTemplate
{
        DECLARE_DYNAMIC(CEnergize_M_Template)

public:
        CEnergize_M_Template(CWnd* pParent = NULL,
                CPoint InsertionPoint = (500, 500),
                CString* pCounterString_F = NULL,
                CString* pCounterString_InM = NULL,
                CString* pCounterString_OutM = NULL,
                CString* pCounterString_InE = NULL,
                CString* pCounterString_OutE = NULL, int ReasOpt = 0);
        virtual ~CEnergize_M_Template();

// Dialog Data
        enum { IDD = IDD_ENERGIZE_M_TEMPLATE };


protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public: // Instances that comprise the Convert_E template
        CFunction* pFunctionBlock;
        CEnergy* pEnergy_InE;
        CEnergy* pEnergy_OutE;
        CMaterial* pMaterial_InM;
        CMaterial* pMaterial_OutM;
        int ReasoningOption;
};
```

```
#pragma once
#include "afxcmn.h"
#include "afxwin.h"
#include "edge.h"

// CEnergy dialog

class CEnergy :
        public CEdge, public CDialog
{
        DECLARE_DYNAMIC(CEnergy)
public:
        CEnergy(CWnd* pParent = NULL,
                CPoint TailClick = (0, 0, 0),
                CPoint HeadClick = (100, 100, 0),
                CString* pCounterString = NULL,
                int ReasOpt = QUALITATIVE_CONSERVATION);   // standard constructor

        virtual ~CEnergy();

        // Dialog Data
        enum { IDD = IDD_ENERGY };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public:

        CList<CEnergy*, CEnergy*> ChildList;
        CList<CEnergy*, CEnergy*> ParentList;
        CString GivenName;
        void DrawOnDC(CDC*pDC);
        int UI_IsResidual;
        BOOL OnInitDialog();
        void OnOK();
        CTreeCtrl* pEnergyTaxonomy;
        HTREEITEM hEnergyType;
        CString EnergyTypeName;

        // Quantitative data members
        double Power;
        double UI_ForceTerm, UI_RateTerm;
        int ReasoningOption;

        afx_msg void OnBnClickedCheck1();
};
```

```
#pragma once
#include "node.h"

#define ENV_SIZE 25

#define ENV_BRUSH_R 255
#define ENV_BRUSH_G 220
#define ENV_BRUSH_B 210

// CEnv dialog
class CEnv :
        public CNode, public CDialog
{
        DECLARE_DYNAMIC(CEnv)

public:
        CEnv(CWnd* pParent = NULL, CPoint InsertionPoint = (500, 500, 0), CString*
pCounterString = NULL);   // standard constructor
```

```
        virtual ~CEnv();

        // Dialog Data
        enum { IDD = IDD_ENV };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()
public:
        // Environment name within block
        CString GivenName;

        // Drawing functions
        void ComputeBlockCoordinates();
        void DrawOnDC(CDC* pDC);
        afx_msg void OnEnChangeEdit1();
};
```

```
#pragma once
#include "node.h"

#define BLOCK_LENGTH 80
#define BLOCK_HEIGHT 40

#define FUNCTION_BRUSH_R 150
#define FUNCTION_BRUSH_G 175
#define FUNCTION_BRUSH_B 200

// CFunction dialog

class CFunction :
        public CNode, public CRect, public CDialog
{
        DECLARE_DYNAMIC(CFunction)

public:
        CFunction(CWnd* pParent = NULL, CPoint InsertionPoint = (500, 500, 0), CString*
pCounterString = NULL);   // standard constructor
        virtual ~CFunction();

        // Dialog Data
        enum { IDD = IDD_FUNCTION };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()
public:

        // Function name within block
        CString GivenName;

        // Drawing functions
        void ComputeBlockCoordinates();
        void DrawOnDC(CDC* pDC);
        bool ElementIsHidden = false;
        // Quantitative data
        double Efficiency;
};
```

```
#pragma once
#include "math.h"

class CGeometry
{
public:
        CGeometry(void);
        ~CGeometry(void);

        // Member Functions
        int RoundToInteger(long n, int t);
        CPoint SnapToGrid(CPoint p);
        long distance(CPoint p1, CPoint p2);
        CPoint* InterpolatePoints(CPoint p1, CPoint p2, double ratio);
};
```

```
// MainFrm.h : interface of the CMainFrame class
//

#pragma once
#include "FileView.h"
#include "ClassView.h"
#include "OutputWnd.h"
#include "PropertiesWnd.h"

class CMainFrame : public CMDIFrameWndEx
{
        DECLARE_DYNAMIC(CMainFrame)
public:
        CMainFrame();

// Attributes
public:

// Operations
public:

// Overrides
public:
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual BOOL LoadFrame(UINT nIDResource, DWORD dwDefaultStyle =
WS_OVERLAPPEDWINDOW | FWS_ADDTOTITLE, CWnd* pParentWnd = NULL, CCreateContext* pContext =
NULL);

// Implementation
public:
        virtual ~CMainFrame();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:  // control bar embedded members
        CMFCMenuBar       m_wndMenuBar;
        CMFCToolBar       m_wndToolBar;
        CMFCStatusBar     m_wndStatusBar;
        CMFCToolBarImages m_UserImages;
        CFileView         m_wndFileView;
        CClassView        m_wndClassView;
        COutputWnd        m_wndOutput;
        CPropertiesWnd    m_wndProperties;

protected:
        CMFCToolBar m_primitivesToolBar;
        CMFCToolBar m_reasoningToolBar;
        CMFCToolBar     m_featuresToolBar;

// Generated message map functions
protected:
        afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
        afx_msg void OnWindowManager();
        afx_msg void OnViewCustomize();
        afx_msg LRESULT OnToolbarCreateNew(WPARAM wp, LPARAM lp);
        afx_msg void OnApplicationLook(UINT id);
        afx_msg void OnUpdateApplicationLook(CCmdUI* pCmdUI);
        afx_msg void OnSettingChange(UINT uFlags, LPCTSTR lpszSection);
        DECLARE_MESSAGE_MAP()

        BOOL CreateDockingWindows();
        void SetDockingWindowIcons(BOOL bHiColorIcons);
};
```

```
#pragma once
#include "edge.h"

// CMaterial dialog
class CMaterial :
        public CEdge, public CDialog
{
        DECLARE_DYNAMIC(CMaterial)

public:
        CMaterial(CWnd* pParent = NULL,
                CPoint TailClick = (0, 0, 0),
                CPoint HeadClick = (100, 100, 0),
                CString* pCounterString = NULL,
                int ReasOpt = QUALITATIVE_CONSERVATION);   // standard constructor
        virtual ~CMaterial();

        // Dialog Data
        enum { IDD = IDD_MATERIAL };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public:
        CList<CMaterial*, CMaterial*> ChildList;
        CList<CMaterial*, CMaterial*> ParentList;
        CString GivenName;
        void DrawOnDC(CDC*pDC);
        int UI_IsResidual;
        BOOL OnInitDialog();
        void OnOK();
        CTreeCtrl* pMaterialTaxonomy;
        HTREEITEM hMaterialType;
        CString MaterialTypeName;
        int ReasoningOption;
};
```

```
#pragma once
#include "element.h"
class CNode :
        public CElement
{
public:
        CNode(void);
        virtual ~CNode(void);    // Must be virtual, so that individual desctrutors of
                                                        // the derived classes are
called when CConModView's
                                                        // destructor tries to close
the session

                                                        // Parameters to check for
dangling functions and env instances
        bool NoInputAttached;
        bool NoOutputAttached;

        void ComputeBlockCoordinates();
};
```

```
#pragma once
#include "edge.h"

// CSignal dialog

class CSignal :
        public CEdge, public CDialog
{
        DECLARE_DYNAMIC(CSignal)

public:
        CSignal(CWnd* pParent = NULL, CPoint TailClick = (0, 0, 0), CPoint HeadClick =
(100, 100, 0), CString* pCounterString = NULL);   // standard constructor
        virtual ~CSignal();

        // Dialog Data
        enum { IDD = IDD_SIGNAL };

protected:
        virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV support

        DECLARE_MESSAGE_MAP()

public:
        //CList<CSignal*, CSignal*> ChildList;
        //CList<CSignal*, CSignal*> ParentList;
        CString GivenName;
        void DrawOnDC(CDC*pDC);
};
```

```
#pragma once

#include "Element.h"
#include "Function.h"
#include "Env.h"
#include "Material.h"
#include "Energy.h"
#include "Signal.h"

#define TEMPLATE_FLOW_LENGTH 120

// Template
class CTemplate : public CElement
{
//      DECLARE_DYNAMIC(CTemplate)

public:
        CTemplate();
        virtual ~CTemplate();

protected:
        //DECLARE_MESSAGE_MAP()
};

/*
This is a high-level abstract class for all templates of the second layer.
The purpose is to provide one identity so that instances all Layer-2 versb, such as
Covnert_E and Energize_M, can be stored in a single list called CTemplateList,
declared in the Doc class as usual.  The template instances are not used in the
model in their own identity, they are only required to instnatiate the elements
such as functions and flows WITHIN the templates using one instance call in
View, such as in functions AddCovnert_E.  After that, the elements are used, while
the template instance must be deleted.  To facilitate this delete, the templates
are stored in this CTemplateList, which is emptied during application exit (View class
desctrictor).
*/
```

Appendix C
Source files for ConMod 2.0

```
// ActuateE_Template.cpp : implementation file
//

#include "stdafx.h"
#include "ConMod2.h"
#include "ActuateE_Template.h"
#include "afxdialogex.h"


// ActuateE_Template dialog
IMPLEMENT_DYNAMIC(ActuateE_Template, CDialogEx)

ActuateE_Template::ActuateE_Template(CWnd* pParent /*= NULL*/,
        CPoint InsertionPoint /*= (500, 500)*/,
        CString* pCounterString_F /*= NULL*/,
        CString* pCounterString_InE /*= NULL*/,
        CString* pCounterString_OutE /*= NULL*/,
        CString* pCounterString_InS /*= NULL*/,
        CString* pCounterString_InCarrier/*= NULL*/,
        CString* pCounterString_OutCarrier /*= NULL*/, int ReasOpt)
        : CDialogEx(IDD_ACTUATE_E_TEMPLATE, pParent), ReasoningOption(ReasOpt)
{
        // Pointer to Function class calls for a new function block
        pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);

        //  Calculations for head and tail node locations on the graphics window
        CPoint TailOfInE(InsertionPoint.x - 1.5*TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
        CPoint HeadOfOutE(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
        CPoint TailOfInCarrierE(InsertionPoint.x, InsertionPoint.y -
1.5*TEMPLATE_FLOW_LENGTH);
        CPoint TailOfInCarrierM(InsertionPoint.x, InsertionPoint.y -
1.5*TEMPLATE_FLOW_LENGTH);
        CPoint HeadOfOutCarrierE(InsertionPoint.x, InsertionPoint.y +
TEMPLATE_FLOW_LENGTH);
        CPoint HeadOfOutCarrierM(InsertionPoint.x, InsertionPoint.y +
TEMPLATE_FLOW_LENGTH);

        // Pointers to InE, OutE, InS, InM_carrier, OutM_carrier, InE_carrier, and out E
carrier for flows entering/leaving function
        pEnergy_InE = new CEnergy(NULL, TailOfInE, InsertionPoint, pCounterString_InE,
this->ReasoningOption);
        pEnergy_OutE = new CEnergy(NULL, InsertionPoint, HeadOfOutE,
pCounterString_OutE, this->ReasoningOption);
        pSignal_InS = new CSignal(NULL, InsertionPoint, InsertionPoint,
pCounterString_InS);
        pMaterial_In_CarrierM = new CMaterial(NULL, TailOfInCarrierM, InsertionPoint,
pCounterString_InCarrier, this->ReasoningOption);
        pMaterial_Out_CarrierM = new CMaterial(NULL, InsertionPoint, HeadOfOutCarrierM,
pCounterString_OutCarrier, this->ReasoningOption);
        pEnergy_In_CarrierE = new CEnergy(NULL, TailOfInCarrierE, InsertionPoint,
pCounterString_InCarrier, this->ReasoningOption);
```

```
        pEnergy_Out_CarrierE = new CEnergy(NULL, InsertionPoint, HeadOfOutCarrierE,
pCounterString_InCarrier, this->ReasoningOption);

        // calls for dialog box
        DoModal();

        // head node of the signal flow goes to function block
        pSignal_InS->pHeadElem = pFunctionBlock;
        pEnergy_InE->pHeadElem = pFunctionBlock;
        pEnergy_OutE->pTailElem = pFunctionBlock;
        if (EisActuated == false)
        {
                pEnergy_OutE->IsHidden = true;
                ModeIsDeActuate();
        }
        else
        {
                ModeIsActuate();
        }
}

ActuateE_Template::~ActuateE_Template()
{
}

void ActuateE_Template::ModeIsActuate()
{
        //if (EisActuated == true)
        {
                EisActuated = true;
                pEnergy_OutE->IsHidden = false;
        }
}
void ActuateE_Template::ModeIsDeActuate()
{
        //if (EisActuated == false)
        {
                EisActuated = false;
                pEnergy_OutE->IsHidden = true;
        }
}



BEGIN_MESSAGE_MAP(ActuateE_Template, CDialogEx)
        ON_BN_CLICKED(IDC_CARRIER_M, &ActuateE_Template::OnBnClickedCarrierM)
        ON_BN_CLICKED(IDC_CARRIER_E, &ActuateE_Template::OnBnClickedCarrierE)
        ON_BN_CLICKED(IDOK, &ActuateE_Template::OnBnClickedOk)
END_MESSAGE_MAP()

// ActuateE_Template message handlers


void ActuateE_Template::OnBnClickedCarrierM()
{
        CarrierIsMaterial = true;
        CarrierIsEnergy = false;
        // Head- and tail-node relationships are established
        pMaterial_In_CarrierM->pHeadElem = pFunctionBlock;
        pSignal_InS->pTailElem = pMaterial_In_CarrierM;
        pMaterial_Out_CarrierM->pTailElem = pFunctionBlock;
}


void ActuateE_Template::OnBnClickedCarrierE()
{
        CarrierIsEnergy = true;
        CarrierIsMaterial = false;
        // Head- and tail-node relationships are established
        pEnergy_In_CarrierE->pHeadElem = pFunctionBlock;
        pSignal_InS->pTailElem = pEnergy_In_CarrierE;
        pEnergy_Out_CarrierE->pTailElem = pFunctionBlock;
}



void ActuateE_Template::OnBnClickedOk()
{
        // TODO: Add your control notification handler code here
}
```

```
#include "stdafx.h"
#include "MainFrm.h"
#include "ClassView.h"
#include "Resource.h"
#include "ConMod2.h"

class CClassViewMenuButton : public CMFCToolBarMenuButton
{
        friend class CClassView;

        DECLARE_SERIAL(CClassViewMenuButton)

public:
        CClassViewMenuButton(HMENU hMenu = NULL) : CMFCToolBarMenuButton((UINT)-1,
hMenu, -1)
        {
        }

        virtual void OnDraw(CDC* pDC, const CRect& rect, CMFCToolBarImages* pImages,
BOOL bHorz = TRUE,
                BOOL bCustomizeMode = FALSE, BOOL bHighlight = FALSE, BOOL bDrawBorder =
TRUE, BOOL bGrayDisabledButtons = TRUE)
        {
                pImages = CMFCToolBar::GetImages();

                CAfxDrawState ds;
                pImages->PrepareDrawImage(ds);

                CMFCToolBarMenuButton::OnDraw(pDC, rect, pImages, bHorz, bCustomizeMode,
bHighlight, bDrawBorder, bGrayDisabledButtons);

                pImages->EndDrawImage(ds);
        }
};

IMPLEMENT_SERIAL(CClassViewMenuButton, CMFCToolBarMenuButton, 1)
```

```
//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

CClassView::CClassView()
{
        m_nCurrSort = ID_SORTING_GROUPBYTYPE;
}

CClassView::~CClassView()
{
}

BEGIN_MESSAGE_MAP(CClassView, CDockablePane)
        ON_WM_CREATE()
        ON_WM_SIZE()
        ON_WM_CONTEXTMENU()
        ON_COMMAND(ID_CLASS_ADD_MEMBER_FUNCTION, OnClassAddMemberFunction)
        ON_COMMAND(ID_CLASS_ADD_MEMBER_VARIABLE, OnClassAddMemberVariable)
        ON_COMMAND(ID_CLASS_DEFINITION, OnClassDefinition)
        ON_COMMAND(ID_CLASS_PROPERTIES, OnClassProperties)
        ON_COMMAND(ID_NEW_FOLDER, OnNewFolder)
        ON_WM_PAINT()
        ON_WM_SETFOCUS()
        ON_COMMAND_RANGE(ID_SORTING_GROUPBYTYPE, ID_SORTING_SORTBYACCESS, OnSort)
        ON_UPDATE_COMMAND_UI_RANGE(ID_SORTING_GROUPBYTYPE, ID_SORTING_SORTBYACCESS,
OnUpdateSort)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CClassView message handlers

int CClassView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
        if (CDockablePane::OnCreate(lpCreateStruct) == -1)
                return -1;

        CRect rectDummy;
        rectDummy.SetRectEmpty();

        // Create views:
        const DWORD dwViewStyle = WS_CHILD | WS_VISIBLE | TVS_HASLINES | TVS_LINESATROOT
| TVS_HASBUTTONS | WS_CLIPSIBLINGS | WS_CLIPCHILDREN;

        if (!m_wndClassView.Create(dwViewStyle, rectDummy, this, 2))
        {
                TRACE0("Failed to create Class View\n");
                return -1;      // fail to create
        }

        // Load images:
        m_wndToolBar.Create(this, AFX_DEFAULT_TOOLBAR_STYLE, IDR_SORT);
        m_wndToolBar.LoadToolBar(IDR_SORT, 0, 0, TRUE /* Is locked */);

        OnChangeVisualStyle();

        m_wndToolBar.SetPaneStyle(m_wndToolBar.GetPaneStyle() | CBRS_TOOLTIPS |
CBRS_FLYBY);
```

```
	m_wndToolBar.SetPaneStyle(m_wndToolBar.GetPaneStyle() & ~(CBRS_GRIPPER | 
CBRS_SIZE_DYNAMIC | CBRS_BORDER_TOP | CBRS_BORDER_BOTTOM | CBRS_BORDER_LEFT | 
CBRS_BORDER_RIGHT));

	m_wndToolBar.SetOwner(this);

	// All commands will be routed via this control , not via the parent frame:
	m_wndToolBar.SetRouteCommandsViaFrame(FALSE);

	CMenu menuSort;
	menuSort.LoadMenu(IDR_POPUP_SORT);

	m_wndToolBar.ReplaceButton(ID_SORT_MENU, 
CClassViewMenuButton(menuSort.GetSubMenu(0)->GetSafeHmenu()));

	CClassViewMenuButton* pButton =  DYNAMIC_DOWNCAST(CClassViewMenuButton, 
m_wndToolBar.GetButton(0));

	if (pButton != NULL)
	{
		pButton->m_bText = FALSE;
		pButton->m_bImage = TRUE;
		pButton->SetImage(GetCmdMgr()->GetCmdImage(m_nCurrSort));
		pButton->SetMessageWnd(this);
	}

	// Fill in some static tree view data (dummy code, nothing magic here)
	FillClassView();

	return 0;
}

void CClassView::OnSize(UINT nType, int cx, int cy)
{
	CDockablePane::OnSize(nType, cx, cy);
	AdjustLayout();
}

void CClassView::FillClassView()
{
	HTREEITEM hRoot = m_wndClassView.InsertItem(_T("FakeApp classes"), 0, 0);
	m_wndClassView.SetItemState(hRoot, TVIS_BOLD, TVIS_BOLD);

	HTREEITEM hClass = m_wndClassView.InsertItem(_T("CFakeAboutDlg"), 1, 1, hRoot);
	m_wndClassView.InsertItem(_T("CFakeAboutDlg()"), 3, 3, hClass);

	m_wndClassView.Expand(hRoot, TVE_EXPAND);

	hClass = m_wndClassView.InsertItem(_T("CFakeApp"), 1, 1, hRoot);
	m_wndClassView.InsertItem(_T("CFakeApp()"), 3, 3, hClass);
	m_wndClassView.InsertItem(_T("InitInstance()"), 3, 3, hClass);
	m_wndClassView.InsertItem(_T("OnAppAbout()"), 3, 3, hClass);

	hClass = m_wndClassView.InsertItem(_T("CFakeAppDoc"), 1, 1, hRoot);
	m_wndClassView.InsertItem(_T("CFakeAppDoc()"), 4, 4, hClass);
	m_wndClassView.InsertItem(_T("~CFakeAppDoc()"), 3, 3, hClass);
	m_wndClassView.InsertItem(_T("OnNewDocument()"), 3, 3, hClass);

	hClass = m_wndClassView.InsertItem(_T("CFakeAppView"), 1, 1, hRoot);
```

```
        m_wndClassView.InsertItem(_T("CFakeAppView()"), 4, 4, hClass);
        m_wndClassView.InsertItem(_T("~CFakeAppView()"), 3, 3, hClass);
        m_wndClassView.InsertItem(_T("GetDocument()"), 3, 3, hClass);
        m_wndClassView.Expand(hClass, TVE_EXPAND);

        hClass = m_wndClassView.InsertItem(_T("CFakeAppFrame"), 1, 1, hRoot);
        m_wndClassView.InsertItem(_T("CFakeAppFrame()"), 3, 3, hClass);
        m_wndClassView.InsertItem(_T("~CFakeAppFrame()"), 3, 3, hClass);
        m_wndClassView.InsertItem(_T("m_wndMenuBar"), 6, 6, hClass);
        m_wndClassView.InsertItem(_T("m_wndToolBar"), 6, 6, hClass);
        m_wndClassView.InsertItem(_T("m_wndStatusBar"), 6, 6, hClass);

        hClass = m_wndClassView.InsertItem(_T("Globals"), 2, 2, hRoot);
        m_wndClassView.InsertItem(_T("theFakeApp"), 5, 5, hClass);
        m_wndClassView.Expand(hClass, TVE_EXPAND);
}

void CClassView::OnContextMenu(CWnd* pWnd, CPoint point)
{
        CTreeCtrl* pWndTree = (CTreeCtrl*)&m_wndClassView;
        ASSERT_VALID(pWndTree);

        if (pWnd != pWndTree)
        {
                CDockablePane::OnContextMenu(pWnd, point);
                return;
        }

        if (point != CPoint(-1, -1))
        {
                // Select clicked item:
                CPoint ptTree = point;
                pWndTree->ScreenToClient(&ptTree);

                UINT flags = 0;
                HTREEITEM hTreeItem = pWndTree->HitTest(ptTree, &flags);
                if (hTreeItem != NULL)
                {
                        pWndTree->SelectItem(hTreeItem);
                }
        }

        pWndTree->SetFocus();
        CMenu menu;
        menu.LoadMenu(IDR_POPUP_SORT);

        CMenu* pSumMenu = menu.GetSubMenu(0);

        if (AfxGetMainWnd()->IsKindOf(RUNTIME_CLASS(CMDIFrameWndEx)))
        {
                CMFCPopupMenu* pPopupMenu = new CMFCPopupMenu;

                if (!pPopupMenu->Create(this, point.x, point.y, (HMENU)pSumMenu-
>m_hMenu, FALSE, TRUE))
                        return;

                ((CMDIFrameWndEx*)AfxGetMainWnd())->OnShowPopupMenu(pPopupMenu);
                UpdateDialogControls(this, FALSE);
        }
```

```
}

void CClassView::AdjustLayout()
{
	if (GetSafeHwnd() == NULL)
	{
		return;
	}

	CRect rectClient;
	GetClientRect(rectClient);

	int cyTlb = m_wndToolBar.CalcFixedLayout(FALSE, TRUE).cy;

	m_wndToolBar.SetWindowPos(NULL, rectClient.left, rectClient.top, 
rectClient.Width(), cyTlb, SWP_NOACTIVATE | SWP_NOZORDER);
	m_wndClassView.SetWindowPos(NULL, rectClient.left + 1, rectClient.top + cyTlb + 
1, rectClient.Width() - 2, rectClient.Height() - cyTlb - 2, SWP_NOACTIVATE | 
SWP_NOZORDER);
}

BOOL CClassView::PreTranslateMessage(MSG* pMsg)
{
	return CDockablePane::PreTranslateMessage(pMsg);
}

void CClassView::OnSort(UINT id)
{
	if (m_nCurrSort == id)
	{
		return;
	}

	m_nCurrSort = id;

	CClassViewMenuButton* pButton =  DYNAMIC_DOWNCAST(CClassViewMenuButton, 
m_wndToolBar.GetButton(0));

	if (pButton != NULL)
	{
		pButton->SetImage(GetCmdMgr()->GetCmdImage(id));
		m_wndToolBar.Invalidate();
		m_wndToolBar.UpdateWindow();
	}
}

void CClassView::OnUpdateSort(CCmdUI* pCmdUI)
{
	pCmdUI->SetCheck(pCmdUI->m_nID == m_nCurrSort);
}

void CClassView::OnClassAddMemberFunction()
{
	AfxMessageBox(_T("Add member function..."));
}

void CClassView::OnClassAddMemberVariable()
{
	// TODO: Add your command handler code here
```

```
}

void CClassView::OnClassDefinition()
{
        // TODO: Add your command handler code here
}

void CClassView::OnClassProperties()
{
        // TODO: Add your command handler code here
}

void CClassView::OnNewFolder()
{
        AfxMessageBox(_T("New Folder..."));
}

void CClassView::OnPaint()
{
        CPaintDC dc(this); // device context for painting

        CRect rectTree;
        m_wndClassView.GetWindowRect(rectTree);
        ScreenToClient(rectTree);

        rectTree.InflateRect(1, 1);
        dc.Draw3dRect(rectTree, ::GetSysColor(COLOR_3DSHADOW),
::GetSysColor(COLOR_3DSHADOW));
}

void CClassView::OnSetFocus(CWnd* pOldWnd)
{
        CDockablePane::OnSetFocus(pOldWnd);

        m_wndClassView.SetFocus();
}

void CClassView::OnChangeVisualStyle()
{
        m_ClassViewImages.DeleteImageList();

        UINT uiBmpId = theApp.m_bHiColorIcons ? IDB_CLASS_VIEW_24 : IDB_CLASS_VIEW;

        CBitmap bmp;
        if (!bmp.LoadBitmap(uiBmpId))
        {
                TRACE(_T("Can't load bitmap: %x\n"), uiBmpId);
                ASSERT(FALSE);
                return;
        }

        BITMAP bmpObj;
        bmp.GetBitmap(&bmpObj);

        UINT nFlags = ILC_MASK;

        nFlags |= (theApp.m_bHiColorIcons) ? ILC_COLOR24 : ILC_COLOR4;

        m_ClassViewImages.Create(16, bmpObj.bmHeight, nFlags, 0, 0);
```

```
	m_ClassViewImages.Add(&bmp, RGB(255, 0, 0));

	m_wndClassView.SetImageList(&m_ClassViewImages, TVSIL_NORMAL);

	m_wndToolBar.CleanUpLockedImages();
	m_wndToolBar.LoadBitmap(theApp.m_bHiColorIcons ? IDB_SORT_24 : IDR_SORT, 0, 0,
TRUE /* Locked */);
}
```

```
// Conduct_E_Template.cpp : implementation file
//

#include "stdafx.h"
#include "ConMod2.h"
#include "Conduct_E_Template.h"
#include "afxdialogex.h"


IMPLEMENT_DYNAMIC(CConduct_E_Template,CDialog)

CConduct_E_Template::CConduct_E_Template(CWnd* pParent /*= NULL*/, CPoint InsertionPoint
/*= (500,500)*/,
	CString* pCounterString_F /*= NULL*/, CString* pCounterString_InE /*= NULL*/,
	CString* pCounterString_OutE /*= NULL*/, CString* pCounterString_OutE_Res /*=
NULL*/,
	int ReasOpt)
	: CDialog(CConduct_E_Template::IDD, pParent)
	, ReasoningOption(ReasOpt)
{
	pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);

	CPoint TailOfInE(InsertionPoint.x - TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
	CPoint HeadOfOutE(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
	CPoint HeadOfOutE_Res(InsertionPoint.x, InsertionPoint.y +
TEMPLATE_FLOW_LENGTH);

	pEnergy_InE = new CEnergy(NULL, TailOfInE, InsertionPoint, pCounterString_InE,
this->ReasoningOption);
	pEnergy_OutE = new CEnergy(NULL, InsertionPoint, HeadOfOutE,
pCounterString_OutE, this->ReasoningOption);
	pEnergy_OutE_Res = new CEnergy(NULL, InsertionPoint, HeadOfOutE_Res,
pCounterString_OutE_Res, this->ReasoningOption);

	pEnergy_InE->pHeadElem = pFunctionBlock;
	pEnergy_OutE->pTailElem = pFunctionBlock;
	pEnergy_OutE_Res->pTailElem = pFunctionBlock;
	pEnergy_OutE_Res->UI_IsResidual = true;
	if (pEnergy_InE->IsHidden == true)
	{
		pEnergy_OutE->IsHidden = true;
	}
}

CConduct_E_Template::~CConduct_E_Template()
{
}

void CConduct_E_Template::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CConduct_E_Template, CDialog)
END_MESSAGE_MAP()


// Conduct_E_Template message handlers
// ConMod2Doc.cpp : implementation of the CConMod2Doc class
//

#include "stdafx.h"
// SHARED_HANDLERS can be defined in an ATL project implementing preview, thumbnail
// and search filter handlers and allows sharing of document code with that project.
#ifndef SHARED_HANDLERS
#include "ConMod2.h"
#endif

#include "ConMod2Doc.h"

#include <propkey.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CConMod2Doc

IMPLEMENT_DYNCREATE(CConMod2Doc, CDocument)

BEGIN_MESSAGE_MAP(CConMod2Doc, CDocument)
END_MESSAGE_MAP()


// CConMod2Doc construction/destruction

CConMod2Doc::CConMod2Doc()
{
        // TODO: add one-time construction code here

}

CConMod2Doc::~CConMod2Doc()
{
}

BOOL CConMod2Doc::OnNewDocument()
{
        if (!CDocument::OnNewDocument())
                return FALSE;

        // TODO: add reinitialization code here
        // (SDI documents will reuse this document)

        return TRUE;
}

// CConMod2Doc serialization
```

```
void CConMod2Doc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}
}

#ifdef SHARED_HANDLERS

// Support for thumbnails
void CConMod2Doc::OnDrawThumbnail(CDC& dc, LPRECT lprcBounds)
{
	// Modify this code to draw the document's data
	dc.FillSolidRect(lprcBounds, RGB(255, 255, 255));

	CString strText = _T("TODO: implement thumbnail drawing here");
	LOGFONT lf;

	CFont* pDefaultGUIFont = CFont::FromHandle((HFONT)
GetStockObject(DEFAULT_GUI_FONT));
	pDefaultGUIFont->GetLogFont(&lf);
	lf.lfHeight = 36;

	CFont fontDraw;
	fontDraw.CreateFontIndirect(&lf);

	CFont* pOldFont = dc.SelectObject(&fontDraw);
	dc.DrawText(strText, lprcBounds, DT_CENTER | DT_WORDBREAK);
	dc.SelectObject(pOldFont);
}

// Support for Search Handlers
void CConMod2Doc::InitializeSearchContent()
{
	CString strSearchContent;
	// Set search contents from document's data.
	// The content parts should be separated by ";"

	// For example:  strSearchContent = _T("point;rectangle;circle;ole object;");
	SetSearchContent(strSearchContent);
}

void CConMod2Doc::SetSearchContent(const CString& value)
{
	if (value.IsEmpty())
	{
		RemoveChunk(PKEY_Search_Contents.fmtid, PKEY_Search_Contents.pid);
	}
	else
	{
		CMFCFilterChunkValueImpl *pChunk = NULL;
		ATLTRY(pChunk = new CMFCFilterChunkValueImpl);
		if (pChunk != NULL)
```

```
                {
                        pChunk->SetTextValue(PKEY_Search_Contents, value, CHUNK_TEXT);
                        SetChunkValue(pChunk);
                }
        }
}

#endif // SHARED_HANDLERS

// CConMod2Doc diagnostics

#ifdef _DEBUG
void CConMod2Doc::AssertValid() const
{
        CDocument::AssertValid();
}

void CConMod2Doc::Dump(CDumpContext& dc) const
{
        CDocument::Dump(dc);
}
#endif //_DEBUG


// CConMod2Doc commands
```

```
#include "stdafx.h"
#include "MemDC.h"
#include "ConMod2.h"
#include "ConMod2Doc.h"
#include "ConMod2View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif


// CConMod2View

IMPLEMENT_DYNCREATE(CConMod2View, CView)

BEGIN_MESSAGE_MAP(CConMod2View, CView)
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, &CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, &CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, &CConMod2View::OnFilePrintPreview)
        ON_WM_CONTEXTMENU()

        // ConMod Main Menu - Reasoning Options Message Handlers
        ON_COMMAND(ID_QUALITATIVE_CONSERVATION,
&CConMod2View::OnQualitativeConservation)
        ON_COMMAND(ID_QUALITATIVE_IRREVERSIBILITY,
&CConMod2View::OnQualitativeIrreversibility)
        ON_COMMAND(ID_QUANTITATIVE_EFFICIENCY, &CConMod2View::OnQuantitativeEfficiency)
        ON_COMMAND(ID_QUANTITATIVE_POWERREQUIRED,
&CConMod2View::OnQuantitativePowerRequired)
```

```
        // ConMod PRIMITIVES Toolbar Commands
        ON_COMMAND(ID_ADD_FUNCTION, Handler_AddFunction)
        ON_COMMAND(ID_ADD_MATERIAL, Handler_AddMaterial)
        ON_COMMAND(ID_ADD_ENERGY, Handler_AddEnergy)
        ON_COMMAND(ID_ADD_SIGNAL, Handler_AddSignal)
        ON_COMMAND(ID_ADD_ENV, Handler_AddEnv)

        // ConMod FEATURES Toolbar Commands
        ON_COMMAND(ID_CONVERT_E, Handler_AddConvert_E_Template)
        ON_COMMAND(ID_CONDUCT_E_TEMPLATE, Handler_AddConduct_E_Template)
        ON_COMMAND(ID_ENERGIZE_M_TEMPLATE, Handler_AddEnergize_M_Template)
        ON_COMMAND(ID_DISTRIBUTE_E_TEMPLATE, Handler_AddDistribute_E_Template)
        ON_COMMAND(ID_DEEN_M_TEMPLATE, Handler_AddDeEn_M_Template)
        ON_COMMAND(ID_ACTUATE_E_TEMPLATE, Handler_AddActuateE_Template)

        // ConMod REASONING Toolbar Commands
        ON_COMMAND(ID_QUALITATIVE, Handler_Qualitative)
        ON_COMMAND(ID_QUANTITATIVE, Handler_Quantitative)


        // ConMod Mouse Event Commands
        ON_WM_LBUTTONDOWN()
        ON_WM_LBUTTONUP()
        ON_WM_RBUTTONDOWN()
        ON_WM_RBUTTONUP()
        ON_WM_MOUSEMOVE()
        ON_WM_LBUTTONDBLCLK()

        // Flicker prevention of the screen
        ON_WM_ERASEBKGND()

        // ConModKeyboard Event Commands
        ON_COMMAND(ID_EDIT_CUT, Handler_EditCut)

        ON_WM_MBUTTONUP()

END_MESSAGE_MAP()

// CConMod2View construction/destruction

CConMod2View::CConMod2View()
{
        ReasoningOption = QUALITATIVE_CONSERVATION;
        ContinueReasoning = true;

        WhatToDo = ESCAPE;
        LButtonIsDown = false;
        RButtonIsDown = false;
        pTailElemDynamic = NULL;
        pHeadElemDynamic = NULL;
        TailNodeSelected = false;
        pElementToBeDeleted = NULL;

        // Conservation Checking Messages
        Msg_OrphanFlow = "";
        Msg_BarrenFlow = "";
        Msg_OneInManyOut_M = "";
        Msg_OneInManyOut_E = "";
        Msg_ManyInOneOut_M = "";
        Msg_ManyInOneOut_E = "";
        Msg_ManyInManyOut = "";
        Msg_MissingResidualEnergy = "";
        Msg_MaterialChangeWithoutEnergy = "";

        Counter_F = 0;
        Counter_Env = 0;
        Counter_M = 0;
        Counter_E = 0;
        Counter_S = 0;

        GrammarCheckRequired = true;
}

CConMod2View::~CConMod2View()
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CElementList.GetHeadPosition(); pos != NULL; )
        {
                delete pDoc->CElementList.GetAt(pos);
                pDoc->CElementList.GetNext(pos);
        }
        pDoc->CFunctionList.RemoveAll();
        pDoc->CEnvList.RemoveAll();
        pDoc->CNodeList.RemoveAll();
        pDoc->CMaterialList.RemoveAll();
        pDoc->CEnergyList.RemoveAll();
        pDoc->CSignalList.RemoveAll();
        pDoc->CEdgeList.RemoveAll();
        pDoc->CElementList.RemoveAll();

}

BOOL CConMod2View::PreCreateWindow(CREATESTRUCT& cs)
{
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs
        cs.lpszClass = AfxRegisterWndClass(CS_DBLCLKS | CS_HREDRAW | CS_VREDRAW,
                AfxGetApp()->LoadCursor(IDC_CROSS), (HBRUSH)(COLOR_WINDOW + 1));

        return CView::PreCreateWindow(cs);
}

// CConMod2View drawing

void CConMod2View::OnDraw(CDC* dc)
{
        MemDC pDC(dc);
        CConMod2Doc* pDoc = GetDocument();
        ASSERT_VALID(pDoc);
        if (!pDoc)
                return;

        if (pDoc->CNodeList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CNodeList.GetHeadPosition(); pos != NULL; )
                {
                        pDoc->CNodeList.GetAt(pos)->NoInputAttached = true;
                        pDoc->CNodeList.GetAt(pos)->NoOutputAttached = true;

                        for (POSITION pos_inner = pDoc->CEdgeList.GetHeadPosition();
pos_inner != NULL; )
                        {
                                if (pDoc->CEdgeList.GetAt(pos_inner)->pTailElem == pDoc-
>CNodeList.GetAt(pos))
                                        pDoc->CNodeList.GetAt(pos)->NoOutputAttached =
false;
                                if (pDoc->CEdgeList.GetAt(pos_inner)->pHeadElem == pDoc-
>CNodeList.GetAt(pos))
                                        pDoc->CNodeList.GetAt(pos)->NoInputAttached =
false;
                                pDoc->CEdgeList.GetNext(pos_inner);
                        }

                        pDoc->CNodeList.GetNext(pos);
                }
        }

        //=========================================================================
        // Check for baggage flows (incoming and outgoing)
        //=========================================================================

        if (pDoc->CEdgeList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CEdgeList.GetHeadPosition(); pos != NULL; )
                {
                        if (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pHeadElem) &&
                                (pDoc->CEdgeList.GetAt(pos)->pHeadElem->pTailElem ==
pDoc->CEdgeList.GetAt(pos)->pTailElem) &&
                                ElementIsNode(pDoc->CEdgeList.GetAt(pos)->pTailElem))
                                pDoc->CEdgeList.GetAt(pos)->ThisFlowIsOutgoingBaggage =
true;
                        else pDoc->CEdgeList.GetAt(pos)->ThisFlowIsOutgoingBaggage =
false;

                        if (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pTailElem) &&
                                (pDoc->CEdgeList.GetAt(pos)->pTailElem->pHeadElem ==
pDoc->CEdgeList.GetAt(pos)->pHeadElem) &&
                                ElementIsNode(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
                                pDoc->CEdgeList.GetAt(pos)->ThisFlowIsIncomingBaggage =
true;
                        else pDoc->CEdgeList.GetAt(pos)->ThisFlowIsIncomingBaggage =
false;

                        pDoc->CEdgeList.GetNext(pos);
                }
        }

        //=========================================================================
        // Update the ReasoningOption variable in Energy flows, so that dialogs
        // show the correct reasoning option check box through ONInitDialog
        //=========================================================================
        if (pDoc->CEnergyList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
                {
                        pDoc->CEnergyList.GetAt(pos)->ReasoningOption = this-
>ReasoningOption;
                        pDoc->CElementList.GetNext(pos);
                }
        }

        //=======================================================================
        // Update the ReasoningOption variable in Material flows, so that dialogs
        // show the correct reasoning option check box through ONInitDialog
        //=======================================================================
        if (pDoc->CMaterialList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != NULL;
)
                {
                        pDoc->CMaterialList.GetAt(pos)->ReasoningOption = this-
>ReasoningOption;
                        pDoc->CMaterialList.GetNext(pos);
                }
        }
        //=======================================================================
        //=======================================================================
        // Redraw all elements
        //=======================================================================
        //=======================================================================

        if (pDoc->CElementList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CElementList.GetHeadPosition(); pos != NULL; )
                {
                        pDoc->CElementList.GetAt(pos)->DrawOnDC(pDC);
                        pDoc->CElementList.GetNext(pos);
                }
        }

        //=======================================================================
        //=======================================================================
        // ***************  APPLY GRAMMAR RULES  *****************
        //=======================================================================
        //=======================================================================

        //=======================================================================
        // Check for uniqueness of GivenName of FUNCTIONS
        //=======================================================================
        if (pDoc->CFunctionList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != pDoc-
>CFunctionList.GetTailPosition(); )
                {
                        if (pDoc->CFunctionList.GetAt(pos)->GivenName == pDoc-
>CFunctionList.GetTail()->GivenName)
                        {
                                AfxMessageBox(CString("ILLEGAL NAMING :: ABORTING
INSTANCE.\n\n***** ") +
                                        pDoc->CFunctionList.GetAt(pos)->GivenName +
                                        " *****\n\nFunction names must be unique.");
                                DeleteElement(pDoc->CFunctionList.GetTail());
                                return;
                        }
                        pDoc->CFunctionList.GetNext(pos);
                }
        }

        //=========================================================================
        // Check for uniqueness of GivenName of ENV
        //=========================================================================
        if (pDoc->CEnvList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CEnvList.GetHeadPosition(); pos != pDoc-
>CEnvList.GetTailPosition(); )
                {
                        if (pDoc->CEnvList.GetAt(pos)->GivenName == pDoc-
>CEnvList.GetTail()->GivenName)
                        {
                                AfxMessageBox(CString("ILLEGAL NAMING :: ABORTING
INSTANCE.\n\n***** ") +
                                        pDoc->CEnvList.GetAt(pos)->GivenName +
                                        " *****\n\nEnvironment names must be unique.");
                                DeleteElement(pDoc->CEnvList.GetTail());
                                return;
                        }
                        pDoc->CEnvList.GetNext(pos);
                }
        }

        //=========================================================================
        // Check for uniqueness of GivenName of MATERIAL
        //=========================================================================
        if (pDoc->CMaterialList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != pDoc-
>CMaterialList.GetTailPosition(); )
                {
                        if (pDoc->CMaterialList.GetAt(pos)->GivenName == pDoc-
>CMaterialList.GetTail()->GivenName)
                        {
                                AfxMessageBox(CString("ILLEGAL NAMING :: ABORTING
INSTANCE.\n\n***** ") +
                                        pDoc->CMaterialList.GetAt(pos)->GivenName +
                                        " *****\n\nMaterial flow names must be
unique.");
                                DeleteElement(pDoc->CMaterialList.GetTail());
                                return;
                        }
                        pDoc->CMaterialList.GetNext(pos);
                }
        }

        //=========================================================================
        // Check for uniqueness of GivenName of ENERGY
        //=========================================================================
        if (pDoc->CEnergyList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != pDoc-
>CEnergyList.GetTailPosition(); )
                {
                        if (pDoc->CEnergyList.GetAt(pos)->GivenName == pDoc-
>CEnergyList.GetTail()->GivenName)
```

```
                        {
                                AfxMessageBox(CString("ILLEGAL NAMING :: ABORTING 
INSTANCE.\n\n***** ") +
                                        pDoc->CEnergyList.GetAt(pos)->GivenName +
                                        " *****\n\nEnergy flow names must be unique.");
                                DeleteElement(pDoc->CEnergyList.GetTail());
                                return;
                        }
                        pDoc->CEnergyList.GetNext(pos);
                }
        }

        //========================================================================
        // Check for uniqueness of GivenName of NEW SIGNAL
        //========================================================================
        if (pDoc->CSignalList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CSignalList.GetHeadPosition(); pos != pDoc-
>CSignalList.GetTailPosition(); )
                {
                        if (pDoc->CSignalList.GetAt(pos)->GivenName == pDoc-
>CSignalList.GetTail()->GivenName)
                        {
                                AfxMessageBox(CString("ILLEGAL NAMING :: ABORTING 
INSTANCE.\n\n***** ") +
                                        pDoc->CSignalList.GetAt(pos)->GivenName +
                                        " *****\n\nSignal flow names must be unique.");
                                DeleteElement(pDoc->CSignalList.GetTail());
                                return;
                        }
                        pDoc->CSignalList.GetNext(pos);
                }
        }

        //========================================================================
        // Check for the Env-Flow-Env construct
        // Check for head node = tail node construct
        // Check for the Double-Carrier construct
        // Check for Wrong Carrier Hierarchy, e.g., E carrying M
        // Check for Carried head != Carrier head construct
        //========================================================================

        if (pDoc->CEdgeList.IsEmpty() == false)
        {
                for (POSITION pos = pDoc->CEdgeList.GetHeadPosition(); ((pos != NULL) && 
(GrammarCheckRequired)); )
                {
                        //==========================
                        // Check for the Env-Flow-Env construct
                        //==========================
                        if (ElementIsNode(pDoc->CEdgeList.GetAt(pos)->pTailElem) &&
                                (pDoc->CEdgeList.GetAt(pos)->pTailElem == pDoc-
>CEdgeList.GetAt(pos)->pHeadElem))
                        {
                                GrammarCheckRequired = false;    // This call is very 
important.

                // Without it, the same instance is attmepted to delete multiple
                // times and the system crashes because it does not find the

                // instance the second time around.
                                AfxMessageBox(_T("ILLEGAL TOPOLOGY :: ABORTING
OPERATION. \n\nA flow cannot have the same head and tail node.\n\n(Get real - This ain't
no FunctionCAD)"));

                                // There are two ways to create the Head = Tail
situation.
                                // (1) At creation time - by selecting the same node
twice
                                // (2) by dragging an existing signal end to two Env's
                                // These three situations are addressed here.

                                if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                        DeleteElement(pDoc->CEdgeList.GetTail());

                                if (WhatToDo == ESCAPE)         // Case 2
                                {
                                        pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
                                        pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                        pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                        pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                }

                                if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3
                                        DeleteElement(pDoc->CNodeList.GetTail());

                                return;
                        }

                        //==========================
                        // Check for the Env-Flow-Env construct
                        //==========================
                        if (ElementIsEnv(pDoc->CEdgeList.GetAt(pos)->pTailElem) &&
                                ElementIsEnv(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
                        {
                                GrammarCheckRequired = false;

                                AfxMessageBox(_T("ILLEGAL TOPOLOGY :: ABORTING
OPERATION. \n\nA flow cannot connect to Env's."));

                                // There are three ways to create the Env-flow-Env
construct:
                                // (1) by adding a flow between two Env instances,
                                // (2) by dragging an existing flow end to two Env's,
and
                                // (3) by adding an Env instance to the end of a flow
that
                                // already has an Env at the other end.  These three
situations
                                // are addressed here.
                                if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                        DeleteElement(pDoc->CEdgeList.GetTail());

                                if (WhatToDo == ESCAPE)          // Case 2
                                {
                                        pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
                                        pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                        pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                        pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                }

                                if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3

                                        DeleteElement(pDoc->CNodeList.GetTail());

                                return;
                        }

                        //===========================
                        // Check for the Double-Carrier construct
                        //===========================
                        if (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pTailElem) &&
                                ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
                        {
                                GrammarCheckRequired = false;
                                AfxMessageBox(_T("ILLEGAL TOPOLOGY :: ABORTING
OPERATION. \n\nA flow cannot have two carriers."));

                                // There are two ways of creating the double-carrier
construct:
                                // (1) At the time of adding a flow, two other flows can
be selected
                                // (2) by connecting the end of a flow to a carrier,
while the
                                // other end already has a carrier
                                // Both cases are addressed here.
                                if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                        DeleteElement(pDoc->CEdgeList.GetTail());

                                if (WhatToDo == ESCAPE)          // Case 2
                                {
                                        pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
                                        pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                        pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                        pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                }
                                if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3
                                        DeleteElement(pDoc->CNodeList.GetTail());

                                return;
                        }
                        //==========================
                        // Check for Wrong Carrier Hierarchy
                        //==========================
                        if (
                                ((ElementIsMaterial(pDoc->CEdgeList.GetAt(pos))) &&
                                ((ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
|| (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pTailElem))))
                                ||
                                ((ElementIsEnergy(pDoc->CEdgeList.GetAt(pos))) &&
                                ((ElementIsEnergy(pDoc->CEdgeList.GetAt(pos)-
>pTailElem)) ||
                                        (ElementIsSignal(pDoc->CEdgeList.GetAt(pos)-
>pTailElem)) ||
                                        (ElementIsEnergy(pDoc->CEdgeList.GetAt(pos)-
>pHeadElem)) ||
                                        (ElementIsSignal(pDoc->CEdgeList.GetAt(pos)-
>pHeadElem))))
                                ||
                                ((ElementIsSignal(pDoc->CEdgeList.GetAt(pos))) &&
                                ((ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
||
                                        ((ElementIsNode(pDoc->CEdgeList.GetAt(pos)-
>pTailElem)) || (ElementIsSignal(pDoc->CEdgeList.GetAt(pos)->pTailElem)))))
                                )
                        {
                                GrammarCheckRequired = false;

                                CString* pCarrierMessage = new CString;

                                if (ElementIsMaterial(pDoc->CEdgeList.GetAt(pos)))
                                        *pCarrierMessage = "Material cannot be carried
by another flow.  No, not even by another Material.";
                                //if (ElementIsEnergy(pDoc->CEdgeList.GetAt(pos)))
                                        *pCarrierMessage = "Energy can be carried by
Material only.  Not by another Energy, not by a Signal.";
                                if (ElementIsSignal(pDoc->CEdgeList.GetAt(pos)))
                                        *pCarrierMessage = "Signal must be carried by a
M or E.  It can go ONLY from its carrier to ONLY a Node.";

                                AfxMessageBox(_T("ILLEGAL CARRIER HIERACHY :: ABORTING
OPERATION.\n\n") + *pCarrierMessage);

                                delete pCarrierMessage;

                                if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                        DeleteElement(pDoc->CEdgeList.GetTail());

                                if (WhatToDo == ESCAPE)          // Case 2
                                {
                                        pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
```

```
                                                pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                                pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                                pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                        }

                                        if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3
                                                DeleteElement(pDoc->CNodeList.GetTail());

                                        return;
                                }

                                //==========================
                                // Check for Carried head != Carrier head construct FOR FLOW-to-
NODE BAGGAGE
                                //==========================
                                if (
                                        (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pTailElem))
&&
                                        (!ElementIsSignal(pDoc->CEdgeList.GetAt(pos))) &&
                                        (ElementIsNode(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
&&
                                        (pDoc->CEdgeList.GetAt(pos)->pHeadElem != pDoc-
>CEdgeList.GetAt(pos)->pTailElem->pHeadElem)
                                        )
                                {
                                        GrammarCheckRequired = false;
                                        AfxMessageBox(_T("ILLEGAL TOPOLOGY :: ABORTING
OPERATION. \n\nA carried Energy flow can be input to ONLY the function \nthat inputs its
carrier."));

                                        if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                                DeleteElement(pDoc->CEdgeList.GetTail());

                                        if (WhatToDo == ESCAPE)          // Case 2
                                        {
                                                pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
                                                pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                                pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                                pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                        }

                                        if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3
                                                DeleteElement(pDoc->CNodeList.GetTail());

                                        return;
                                }

                                //==========================
                                // Check for Carried head != Carrier head construct for NODE-to-
FLOW BAGGAGE
                                //========================
                                if (
                                        (ElementIsNode(pDoc->CEdgeList.GetAt(pos)->pTailElem))
&&
                                        (ElementIsEdge(pDoc->CEdgeList.GetAt(pos)->pHeadElem))
&&
                                        (pDoc->CEdgeList.GetAt(pos)->pTailElem != pDoc-
>CEdgeList.GetAt(pos)->pHeadElem->pTailElem)
                                        )
                                {
                                        GrammarCheckRequired = false;
                                        AfxMessageBox(_T("ILLEGAL TOPOLOGY :: ABORTING
OPERATION. \n\nA carried Energy flow can be added to a flow ONLY by the function \nthat
outputs its carrier."));

                                        if ((WhatToDo == ADD_MATERIAL) || (WhatToDo ==
ADD_ENERGY) || (WhatToDo == ADD_SIGNAL)) // Case 1
                                                DeleteElement(pDoc->CEdgeList.GetTail());

                                        if (WhatToDo == ESCAPE)         // Case 2
                                        {
                                                pDoc->CEdgeList.GetAt(pos)->pHeadElem =
pRememberHeadElement;
                                                pDoc->CEdgeList.GetAt(pos)->HeadPoint =
RememberHeadPoint;
                                                pDoc->CEdgeList.GetAt(pos)->pTailElem =
pRememberTailElement;
                                                pDoc->CEdgeList.GetAt(pos)->TailPoint =
RememberTailPoint;
                                        }

                                        if ((WhatToDo == ADD_ENV) || (WhatToDo == ADD_FUNCTION))
        // Case 3
                                                DeleteElement(pDoc->CNodeList.GetTail());

                                        return;
                                }

                                pDoc->CEdgeList.GetNext(pos);
                        }
        }
        GrammarCheckRequired = true;
}


// CConMod2View printing


void CConMod2View::OnFilePrintPreview()
{
#ifndef SHARED_HANDLERS
        AFXPrintPreview(this);
#endif
}

BOOL CConMod2View::OnPreparePrinting(CPrintInfo* pInfo)
{
	// default preparation
	return DoPreparePrinting(pInfo);
}

void CConMod2View::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add extra initialization before printing
}

void CConMod2View::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
	// TODO: add cleanup after printing
}



void CConMod2View::OnContextMenu(CWnd* /* pWnd */, CPoint point)
{
#ifndef SHARED_HANDLERS
	theApp.GetContextMenuManager()->ShowPopupMenu(IDR_POPUP_EDIT, point.x, point.y,
this, TRUE);
#endif
}


// CConMod2View diagnostics

#ifdef _DEBUG
void CConMod2View::AssertValid() const
{
	CView::AssertValid();
}

void CConMod2View::Dump(CDumpContext& dc) const
{
	CView::Dump(dc);
}

CConMod2Doc* CConMod2View::GetDocument() const // non-debug version is inline
{
	ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CConMod2Doc)));
	return (CConMod2Doc*)m_pDocument;
}
#endif //_DEBUG
// ============================================================================
// ============================================================================
// STANDARD TOOLBAR EVENT HANDLER - FILE SAVE
// ============================================================================
// ============================================================================

void CConMod2View::Handler_SaveFile(void)
{
	AfxMessageBox(_T("Save File."));
}

// ============================================================================
// ============================================================================
// MAIN MENU - REASONING OPTION MESSAGE HANDLER FUNCTIONS
// ============================================================================
// ===========================================================================
void CConMod2View::OnQualitativeConservation()
{
        ReasoningOption = QUALITATIVE_CONSERVATION;
        AfxMessageBox(_T("Reasoning Switched to: QUALITATIVE CONSERVATION."));
}

void CConMod2View::OnQualitativeIrreversibility()
{
        ReasoningOption = QUALITATIVE_IRREVERSIBILITY;
        AfxMessageBox(_T("Reasoning Switched to: QUALITATIVE IRREVERSIBILITY."));
}

void CConMod2View::OnQuantitativeEfficiency()
{
        ReasoningOption = QUANTITATIVE_EFFICIENCY;
        AfxMessageBox(_T("Reasoning Switched to: EFFICIENCY."));
}

void CConMod2View::OnQuantitativePowerRequired()
{
        ReasoningOption = QUANTITATIVE_POWERREQUIRED;
        AfxMessageBox(_T("Reasoning Switched to: POWER REQUIRED."));
}


// ===========================================================================
// ===========================================================================
// PRIMITIVES TOOLBAR EVENT HANDLER FUNCTIONS - ONLY FOR SETTING "WHAT TO DO"
// ===========================================================================
// ===========================================================================

void CConMod2View::Handler_AddFunction(void)
{
        WhatToDo = ADD_FUNCTION;
}

void CConMod2View::Handler_AddMaterial(void)
{
        WhatToDo = ADD_MATERIAL;
}

void CConMod2View::Handler_AddEnergy(void)
{
        WhatToDo = ADD_ENERGY;
}

void CConMod2View::Handler_AddSignal(void)
{
        WhatToDo = ADD_SIGNAL;
}

void CConMod2View::Handler_AddEnv(void)
{
        WhatToDo = ADD_ENV;
}

// ===========================================================================
// ===========================================================================
// FEATURES TOOLBAR EVENT HANDLER FUNCTIONS - ONLY FOR SETTING "WHAT TO DO"
// ===========================================================================
// ===========================================================================

void CConMod2View::Handler_AddConvert_E_Template(void)
{
        WhatToDo = ADD_CONVERT_E_TEMPLATE;
}

void CConMod2View::Handler_AddConduct_E_Template(void)
{
        WhatToDo = ADD_CONDUCT_E_TEMPLATE;
}

void CConMod2View::Handler_AddEnergize_M_Template(void)
{
        WhatToDo = ADD_ENERGIZE_M_TEMPLATE;
}

void CConMod2View::Handler_AddDistribute_E_Template(void)
{
        WhatToDo = ADD_DISTRIBUTE_E_TEMPLATE;
}

void CConMod2View::Handler_AddDeEn_M_Template(void)
{
        WhatToDo = ADD_DEEN_M_TEMPLATE;
}

void CConMod2View::Handler_AddActuateE_Template(void)
{
        WhatToDo = ADD_ACTUATEE_TEMPLATE;
}

// ===========================================================================
// ===========================================================================
// REASONING TOOLBAR EVENT HANDLER FUNCTIONS - ONLY FOR SETTING "WHAT TO DO"
// ===========================================================================
// ===========================================================================

void CConMod2View::Handler_Qualitative(void)
{
        ComposeQualitativeMessage();
}

void CConMod2View::Handler_Quantitative(void)
{
        ComposeQuantitativeMessage();
}
void CConMod2View::Handler_Causal(void)
{
        ComposeCausalMessage();
}


// ===========================================================================
// ===========================================================================
// EDIT TOOLBAR EVENT HANDLER FUNCTIONS
// ===========================================================================
// ============================================================================

void CConMod2View::Handler_EditCut()
{
        if (pSelectedElement == NULL)
                return;
        if (WhatToDo == ESCAPE)
        {
                CConMod2Doc* pDoc = GetDocument();
                DetachEdgesFromElement(pSelectedElement);
                DeleteElement(pSelectedElement);
                pSelectedElement = NULL;         // Resets the pointer to NULL
        }

        //OnDraw(this->GetDC());
};
// ============================================================================
// ============================================================================
// MOUSE EVENT HANDLER FUNCTIONS - CALLS THE APPROPRIATE INSTANCE-ADDING FNC.
// ============================================================================
// ============================================================================

void CConMod2View::OnLButtonDown(UINT nFlags, CPoint point)
{
        CConMod2Doc* pDoc = GetDocument();

        MouseLDownPoint = point;
        LButtonIsDown = TRUE;

        switch (WhatToDo)
        {
        case ESCAPE:
                if (ElementIsEdge(pSelectedElement))
                {
                        // Remember head topology, in case you have to revert back to
this state.
                        // This condition may arise if the new topology after move /
connect is
                        // unacceptable by the grammar rules, in which case the OnDraw
function
                        // reverts the topology to the older "REMEMBERED" one.

                        RememberHeadPoint = pSelectedElement->HeadPoint;
                        pRememberHeadElement = pSelectedElement->pHeadElem;
                        RememberTailPoint = pSelectedElement->TailPoint;
                        pRememberTailElement = pSelectedElement->pTailElem;
                }
                break;

        case ADD_FUNCTION:
                AddFunction();
                break;

        case ADD_ENV:
                AddEnv();
                break;
        case ADD_CONVERT_E_TEMPLATE:
                AddConvert_E_Template();
                break;
        case ADD_CONDUCT_E_TEMPLATE:
                AddConduct_E_Template();
                break;

        case ADD_ENERGIZE_M_TEMPLATE:
                AddEnergize_M_Template();
                break;

        case ADD_DISTRIBUTE_E_TEMPLATE:
                AddDistribute_E_Template();
                break;

        case ADD_DEEN_M_TEMPLATE:
                AddDeEn_M_Template();
                break;
        case ADD_ACTUATEE_TEMPLATE:
                AddActuateE_Template();
                break;
        }
}

void CConMod2View::OnLButtonUp(UINT nFlags, CPoint point)
{
        MouseLUpPoint = point;
        CConMod2Doc* pDoc = GetDocument();

        switch (WhatToDo)
        {
        case ESCAPE:
                MoveConnect();
                break;

        case ADD_MATERIAL:
                AddMaterial();
                break;

        case ADD_ENERGY:
                AddEnergy();
                break;

        case ADD_SIGNAL:
                AddSignal();
                break;
        }

        LButtonIsDown = FALSE;
}

void CConMod2View::OnRButtonDown(UINT nFlags, CPoint point)
{
        MouseRDownPoint = point;
        RButtonIsDown = TRUE;

        CConMod2Doc* pDoc = GetDocument();

        // IF PreselectionList IS EMPTY, RIGHT CLICK WILL SET WhatToDo = ESCAPE
        // OTHERWISE, IF THE LIST IS FULL, IT SHOULD SCROLL THROUGH THAT LIST
        if (pDoc->PreselectionList.IsEmpty())
```

```
                WhatToDo = ESCAPE;
        else
                ScrollThroughPreselection();
}

void CConMod2View::OnRButtonUp(UINT /* nFlags */, CPoint point)
{
        MouseRUpPoint = point;
        RButtonIsDown = FALSE;
}


void CConMod2View::OnMouseMove(UINT nFlags, CPoint point)
{
        MouseMovePoint = point;

        // FOR ALL WHATTODO's, IF BOTH BUTTONS ARE UP, MOUSE MOVEMENT WILL PRESELECT
ELEMENTS

        if ((!LButtonIsDown) && (!RButtonIsDown))
                Preselect(&point);

        switch (WhatToDo)
        {
        case ESCAPE:
                if (LButtonIsDown && pSelectedElement != NULL)
                        MoveConnectDynamic();
                break;

        case ADD_MATERIAL:
                if ((LButtonIsDown) && (!RButtonIsDown))
                        AddEdge_Dynamic();
                break;

        case ADD_ENERGY:
                if ((LButtonIsDown) && (!RButtonIsDown))
                        AddEdge_Dynamic();
                break;

        case ADD_SIGNAL:
                if ((LButtonIsDown) && (!RButtonIsDown))
                        AddEdge_Dynamic();
                break;
        }
}
void CConMod2View::OnLButtonDblClk(UINT nFlags, CPoint point)
{
        CConMod2Doc* pDoc = this->GetDocument();

        if (pSelectedElement == NULL)
                ComposeQualitativeMessage();
        else if (ElementIsEnv(pSelectedElement))
                pDoc->CEnvList.GetAt(EnvIndexInEnvList)->DoModal();
        else if (ElementIsMaterial(pSelectedElement))
                pDoc->CMaterialList.GetAt(MaterialIndexInMaterialList)->DoModal();
        else if (ElementIsEnergy(pSelectedElement))
                pDoc->CEnergyList.GetAt(EnergyIndexInEnergyList)->DoModal();
        else if (ElementIsSignal(pSelectedElement))
        {
                CSignal * selSignal = pDoc->CSignalList.GetAt(SignalIndexInSignalList);
                selSignal->DoModal();
                CString* ChangedSignal = new CString;
                *ChangedSignal = selSignal->GivenName;
                AfxMessageBox(_T("Current Signal Is " + *ChangedSignal));
                for (POSITION pos1 = pDoc->ActuateE_Template_List.GetHeadPosition();
pos1 != NULL; )
                {
                        if (pDoc->ActuateE_Template_List.GetAt(pos1)->pSignal_InS ==
pSelectedElement)
                        {
                                DetermineState(*ChangedSignal, pDoc-
>ActuateE_Template_List.GetAt(pos1));
                                CausalReasoning(pDoc-
>ActuateE_Template_List.GetAt(pos1));
                        }
                        pDoc->ActuateE_Template_List.GetNext(pos1);
                }

        }
void CConMod2View::CausalReasoning(ActuateE_Template* actuatedfunc)
{
        CConMod2Doc* pDoc = this->GetDocument();
        int i = 0;
        do
        {
                for (POSITION pos2 = pDoc->CFunctionList.GetHeadPosition(); pos2 !=
NULL; pDoc->CFunctionList.GetNext(pos2))
                {
                        CFunction * currFunc = pDoc->CFunctionList.GetAt(pos2);
                        for (POSITION pos3 = pDoc->CEdgeList.GetHeadPosition(); pos3 !=
NULL; pDoc->CEdgeList.GetNext(pos3))
                        {
                                CEdge * currEdge = pDoc->CEdgeList.GetAt(pos3);
                                if (actuatedfunc->EisActuated == true)
                                {
                                        if ((currEdge->IsHidden == false) && (currEdge-
>pHeadElem == currFunc))
                                        {
                                                currFunc->IsHidden = false;
                                        }
                                        if ((currFunc->IsHidden == false) && (currEdge-
>pTailElem == currFunc))
                                        {
                                                currEdge->IsHidden = false;
                                        }
                                }
                                else
                                {
                                        if ((currEdge->IsHidden == true) && (currEdge-
>pHeadElem == currFunc))
                                        {
                                                currFunc->IsHidden = true;
                                        }
                                        if ((currFunc->IsHidden == true) && (currEdge-
>pTailElem == currFunc))
                                        {
                                                currEdge->IsHidden = true;
                                        }
                                }
                                /*pDoc->CEdgeList.GetNext(pos3);*/
                        }
                        //pDoc->CFunctionList.GetNext(pos2);
                }
                i++;
        } while (i< pDoc->CFunctionList.GetCount());
}

void CConMod2View::OnMButtonUp(UINT nFlags, CPoint point)
{
        //if ((this->ElementIsEnergy(this->pSelectedElement)) ||
        //      (this->ElementIsMaterial(this->pSelectedElement)))
        //      this->pSelectedElement->IsResidual = !(this->pSelectedElement-
>IsResidual);
}

BOOL CConMod2View::OnEraseBkgnd(CDC* pDC)
{
        return FALSE;
}

// ============================================================================
// ============================================================================
// FUNCTIONS THAT ADD, DELETE, AND MODIFY INSTANCES IN THE MODEL
// ============================================================================
// ============================================================================

void CConMod2View::AddFunction()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        //=========================================================================
        // Auto-increment the name counter
        //=========================================================================
        Counter_F = Counter_F + 1;
        CounterString.Format(_T("%d"), Counter_F);

        if ((ElementIsEdge(pSelectedElement)) && (pSelectedElement->GrabHandle == TAIL))
        // Grabbed an edge at tail
        {
                CFunction* NewCFunction = new CFunction(NULL,
SnapToGrid(pSelectedElement->TailPoint), &CounterString);
                pSelectedElement->pTailElem = NewCFunction;
                pDoc->CElementList.AddTail(NewCFunction);
                pDoc->CNodeList.AddTail(NewCFunction);
                pDoc->CFunctionList.AddTail(NewCFunction);
        }

        if ((ElementIsEdge(pSelectedElement)) && (pSelectedElement->GrabHandle == HEAD))
        // Grabbed an edge at head
        {
                CFunction* NewCFunction = new CFunction(NULL,
SnapToGrid(pSelectedElement->HeadPoint), &CounterString);
                pSelectedElement->pHeadElem = NewCFunction;
                pDoc->CElementList.AddTail(NewCFunction);
                pDoc->CNodeList.AddTail(NewCFunction);
                pDoc->CFunctionList.AddTail(NewCFunction);
        }

        if (pSelectedElement == NULL)
        {
                CFunction* NewCFunction = new CFunction(NULL,
SnapToGrid(MouseLDownPoint), &CounterString);
                pDoc->CElementList.AddTail(NewCFunction);
                pDoc->CNodeList.AddTail(NewCFunction);
                pDoc->CFunctionList.AddTail(NewCFunction);
        }

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;           // Without this line, LButtonIsDown remains set
                                                        // to TRUE, since a
click was made on the
                                                        // screen to add the
function.  When the button
                                                        // is lifted, it is
usually in the Add Function
                                                        // Dialog, so the
graphics window does not know
                                                        // that L Button was
lifted.  Therefore,
                                                        // functions such as
Preselect misbehave.
}
void CConMod2View::AddEdge_Dynamic()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();
        CEdge* NewCEdge = new CEdge(MouseLDownPoint, MouseMovePoint);

        //NewCEdge->pTailElem = pTailElemDynamic;
        if (pTailElemDynamic != NULL)
        {
                NewCEdge->pTailElem = pTailElemDynamic;
                TailNodeSelected = true;
        }

        Preselect(&MouseMovePoint);
        NewCEdge->pHeadElem = pHeadElemDynamic;

        NewCEdge->DrawOnDC(this->GetDC());
        delete NewCEdge;
}

void CConMod2View::AddMaterial()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        Counter_M = Counter_M + 1;
        CounterString.Format(_T("%d"), Counter_M);

        CMaterial* NewCMaterial = new CMaterial(NULL, MouseLDownPoint, MouseLUpPoint,
&CounterString, this->ReasoningOption);

        NewCMaterial->pTailElem = pTailElemDynamic;
        NewCMaterial->pHeadElem = pHeadElemDynamic;
        pDoc->CElementList.AddTail(NewCMaterial);
        pDoc->CEdgeList.AddTail(NewCMaterial);
        pDoc->CMaterialList.AddTail(NewCMaterial);
        //OnDraw(this->GetDC());

        // Clear up the temporary edge creation data for the next use
        pTailElemDynamic = NULL;
        pHeadElemDynamic = NULL;
        TailNodeSelected = false;
}

void CConMod2View::AddEnergy()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        Counter_E = Counter_E + 1;
        CounterString.Format(_T("%d"), Counter_E);

        CEnergy* NewCEnergy = new CEnergy(NULL, MouseLDownPoint, MouseLUpPoint,
&CounterString, this->ReasoningOption);

        NewCEnergy->pTailElem = pTailElemDynamic;
        NewCEnergy->pHeadElem = pHeadElemDynamic;

        pDoc->CElementList.AddTail(NewCEnergy);
        pDoc->CEdgeList.AddTail(NewCEnergy);
        pDoc->CEnergyList.AddTail(NewCEnergy);
        //OnDraw(this->GetDC());

        // Clear up the temporary edge creation data for the next use
        pTailElemDynamic = NULL;
        pHeadElemDynamic = NULL;
        TailNodeSelected = false;
}

void CConMod2View::AddSignal()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        Counter_S = Counter_S + 1;
        CounterString.Format(_T("%d"), Counter_S);

        CSignal* NewCSignal = new CSignal(NULL, MouseLDownPoint, MouseLUpPoint,
&CounterString);

        NewCSignal->pTailElem = pTailElemDynamic;
        NewCSignal->pHeadElem = pHeadElemDynamic;

        pDoc->CElementList.AddTail(NewCSignal);
        pDoc->CEdgeList.AddTail(NewCSignal);
        pDoc->CSignalList.AddTail(NewCSignal);
        //OnDraw(this->GetDC());

        // Clear up the temporary edge creation data for the next use
        pTailElemDynamic = NULL;
        pHeadElemDynamic = NULL;
        TailNodeSelected = false;
}

void CConMod2View::AddEnv()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        Counter_Env = Counter_Env + 1;
        CounterString.Format(_T("%d"), Counter_Env);

        if ((ElementIsEdge(pSelectedElement)) && (pSelectedElement->GrabHandle == TAIL))
        // Grabbed an edge at tail
        {
                CEnv* NewCEnv = new CEnv(NULL, SnapToGrid(pSelectedElement->TailPoint),
&CounterString);
                pSelectedElement->pTailElem = NewCEnv;
                pDoc->CElementList.AddTail(NewCEnv);
                pDoc->CNodeList.AddTail(NewCEnv);
                pDoc->CEnvList.AddTail(NewCEnv);
        }

        if ((ElementIsEdge(pSelectedElement)) && (pSelectedElement->GrabHandle == HEAD))
        // Grabbed an edge at head
        {
                CEnv* NewCEnv = new CEnv(NULL, SnapToGrid(pSelectedElement->HeadPoint),
&CounterString);
                pSelectedElement->pHeadElem = NewCEnv;
                pDoc->CElementList.AddTail(NewCEnv);
                pDoc->CNodeList.AddTail(NewCEnv);
                pDoc->CEnvList.AddTail(NewCEnv);
        }

        if (pSelectedElement == NULL)
        {
                CEnv* NewCEnv = new CEnv(NULL, SnapToGrid(MouseLDownPoint),
&CounterString);
                pDoc->CElementList.AddTail(NewCEnv);
                pDoc->CNodeList.AddTail(NewCEnv);
                pDoc->CEnvList.AddTail(NewCEnv);
        }

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;          // Without this line, LButtonIsDown remains set
                                                        // to TRUE, since a
click was made on the
                                                        // screen to add the
function.  When the button
                                                        // is lifted, it is
usually in the Add Function
                                                        // Dialog, so the
graphics window does not know
                                                        // that L Button was
lifted.  Therefore,
                                                        // functions such as
Preselect misbehave.
}
void CConMod2View::AddConvert_E_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        CString* pCounterString_F = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE = new CString;
        CString* pCounterString_OutE_Res = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + _T(" [Conv_E]");
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE_Res->Format(_T("%d"), Counter_E);


        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                          //attachment
issues will arise
        {
                CConvert_E_Template* NewCConvert_E_Template = new
CConvert_E_Template(NULL, SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InE, pCounterString_OutE,
pCounterString_OutE_Res, this->ReasoningOption);

                pDoc->CTemplateList.AddTail(NewCConvert_E_Template);
                pDoc->CConvert_E_Template_List.AddTail(NewCConvert_E_Template);

                pDoc->CElementList.AddTail(NewCConvert_E_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewCConvert_E_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewCConvert_E_Template->pFunctionBlock);
                pDoc->CConvert_E_Function_List.AddTail(NewCConvert_E_Template-
>pFunctionBlock);       // Enables grammr checking

                pDoc->CElementList.AddTail(NewCConvert_E_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewCConvert_E_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewCConvert_E_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewCConvert_E_Template->pEnergy_OutE);
                pDoc->CEdgeList.AddTail(NewCConvert_E_Template->pEnergy_OutE);
                pDoc->CEnergyList.AddTail(NewCConvert_E_Template->pEnergy_OutE);

                pDoc->CElementList.AddTail(NewCConvert_E_Template->pEnergy_OutE_Res);
                pDoc->CEdgeList.AddTail(NewCConvert_E_Template->pEnergy_OutE_Res);
                pDoc->CEnergyList.AddTail(NewCConvert_E_Template->pEnergy_OutE_Res);
        }

        delete pCounterString_F;
        delete pCounterString_InE;
        delete pCounterString_OutE;
        delete pCounterString_OutE_Res;

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;          // Without this line, LButtonIsDown remains set
}
```

```
void CConMod2View::AddConduct_E_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        CString* pCounterString_F = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE = new CString;
        CString* pCounterString_OutE_Res = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + _T(" [Cond_E]");
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE_Res->Format(_T("%d"), Counter_E);


        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                    //attachment
issues will arise
        {
                CConduct_E_Template* NewCConduct_E_Template = new
CConduct_E_Template(NULL, SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InE, pCounterString_OutE,
pCounterString_OutE_Res, this->ReasoningOption);

                pDoc->CTemplateList.AddTail(NewCConduct_E_Template);
                pDoc->CConduct_E_Template_List.AddTail(NewCConduct_E_Template);

                pDoc->CElementList.AddTail(NewCConduct_E_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewCConduct_E_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewCConduct_E_Template->pFunctionBlock);
                pDoc->CConduct_E_Function_List.AddTail(NewCConduct_E_Template-
>pFunctionBlock);        // Enables grammr checking

                pDoc->CElementList.AddTail(NewCConduct_E_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewCConduct_E_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewCConduct_E_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewCConduct_E_Template->pEnergy_OutE);
                pDoc->CEdgeList.AddTail(NewCConduct_E_Template->pEnergy_OutE);
                pDoc->CEnergyList.AddTail(NewCConduct_E_Template->pEnergy_OutE);

                pDoc->CElementList.AddTail(NewCConduct_E_Template->pEnergy_OutE_Res);
                pDoc->CEdgeList.AddTail(NewCConduct_E_Template->pEnergy_OutE_Res);
                pDoc->CEnergyList.AddTail(NewCConduct_E_Template->pEnergy_OutE_Res);
        }

        delete pCounterString_F;
        delete pCounterString_InE;
        delete pCounterString_OutE;
        delete pCounterString_OutE_Res;

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;           // Without this line, LButtonIsDown remains set
}

void CConMod2View::AddEnergize_M_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        CString* pCounterString_F = new CString;
        CString* pCounterString_InM = new CString;
        CString* pCounterString_OutM = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + _T(" [En_Mat]");
        Counter_M++;
        pCounterString_InM->Format(_T("%d"), Counter_M);
        Counter_M++;
        pCounterString_OutM->Format(_T("%d"), Counter_M);
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE->Format(_T("%d"), Counter_E);


        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                          //attachment
issues will arise
        {
                CEnergize_M_Template* NewCEnergize_M_Template = new
CEnergize_M_Template(NULL, SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InM, pCounterString_OutM,
pCounterString_InE, pCounterString_OutE, this->ReasoningOption);

                pDoc->CTemplateList.AddTail(NewCEnergize_M_Template);
                pDoc->CEnergize_M_Template_List.AddTail(NewCEnergize_M_Template);

                pDoc->CElementList.AddTail(NewCEnergize_M_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewCEnergize_M_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewCEnergize_M_Template->pFunctionBlock);
                pDoc->CEnergize_M_Function_List.AddTail(NewCEnergize_M_Template-
>pFunctionBlock);        // Enables grammr checking

                pDoc->CElementList.AddTail(NewCEnergize_M_Template->pMaterial_InM);
                pDoc->CEdgeList.AddTail(NewCEnergize_M_Template->pMaterial_InM);
                pDoc->CMaterialList.AddTail(NewCEnergize_M_Template->pMaterial_InM);

                pDoc->CElementList.AddTail(NewCEnergize_M_Template->pMaterial_OutM);
                pDoc->CEdgeList.AddTail(NewCEnergize_M_Template->pMaterial_OutM);
                pDoc->CMaterialList.AddTail(NewCEnergize_M_Template->pMaterial_OutM);

                pDoc->CElementList.AddTail(NewCEnergize_M_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewCEnergize_M_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewCEnergize_M_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewCEnergize_M_Template->pEnergy_OutE);
                pDoc->CEdgeList.AddTail(NewCEnergize_M_Template->pEnergy_OutE);
                pDoc->CEnergyList.AddTail(NewCEnergize_M_Template->pEnergy_OutE);

        }

        delete pCounterString_F;
        delete pCounterString_InM;
        delete pCounterString_OutM;
        delete pCounterString_InE;
        delete pCounterString_OutE;

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;           // Without this line, LButtonIsDown remains set
}
void CConMod2View::AddDistribute_E_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        CString* pCounterString_F = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE1 = new CString;
        CString* pCounterString_OutE2 = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + _T(" [Dist_E]");
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE1->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE2->Format(_T("%d"), Counter_E);


        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                         //attachment
issues will arise
        {
                CDistribute_E_Template* NewCDistribute_E_Template = new
CDistribute_E_Template(NULL, SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InE, pCounterString_OutE1,
pCounterString_OutE2, this->ReasoningOption);

                pDoc->CTemplateList.AddTail(NewCDistribute_E_Template);
                pDoc->CDistribute_E_Template_List.AddTail(NewCDistribute_E_Template);

                pDoc->CElementList.AddTail(NewCDistribute_E_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewCDistribute_E_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewCDistribute_E_Template->pFunctionBlock);
                pDoc->CDistribute_E_Function_List.AddTail(NewCDistribute_E_Template-
>pFunctionBlock);        // Enables grammr checking

                pDoc->CElementList.AddTail(NewCDistribute_E_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewCDistribute_E_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewCDistribute_E_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewCDistribute_E_Template->pEnergy_OutE1);
                pDoc->CEdgeList.AddTail(NewCDistribute_E_Template->pEnergy_OutE1);
                pDoc->CEnergyList.AddTail(NewCDistribute_E_Template->pEnergy_OutE1);

                pDoc->CElementList.AddTail(NewCDistribute_E_Template->pEnergy_OutE2);
                pDoc->CEdgeList.AddTail(NewCDistribute_E_Template->pEnergy_OutE2);
                pDoc->CEnergyList.AddTail(NewCDistribute_E_Template->pEnergy_OutE2);
        }

        delete pCounterString_F;
        delete pCounterString_InE;
        delete pCounterString_OutE1;
        delete pCounterString_OutE2;

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;           // Without this line, LButtonIsDown remains set
}

void CConMod2View::AddDeEn_M_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        CString* pCounterString_F = new CString;
        CString* pCounterString_InM = new CString;
        CString* pCounterString_OutM = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + _T(" [DeEn_M]");
        Counter_M++;
        pCounterString_InM->Format(_T("%d"), Counter_M);
        Counter_M++;
        pCounterString_OutM->Format(_T("%d"), Counter_M);
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE->Format(_T("%d"), Counter_E);


        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                      //attachment
issues will arise
        {
                CDeEn_M_Template* NewCDeEn_M_Template = new CDeEn_M_Template(NULL,
SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InM, pCounterString_OutM,
pCounterString_InE, pCounterString_OutE, this->ReasoningOption);

                pDoc->CTemplateList.AddTail(NewCDeEn_M_Template);
                pDoc->CDeEn_M_Template_List.AddTail(NewCDeEn_M_Template);

                pDoc->CElementList.AddTail(NewCDeEn_M_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewCDeEn_M_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewCDeEn_M_Template->pFunctionBlock);
                pDoc->CEnergize_M_Function_List.AddTail(NewCDeEn_M_Template-
>pFunctionBlock);       // Enables grammr checking
```

```
                pDoc->CElementList.AddTail(NewCDeEn_M_Template->pMaterial_InM);
                pDoc->CEdgeList.AddTail(NewCDeEn_M_Template->pMaterial_InM);
                pDoc->CMaterialList.AddTail(NewCDeEn_M_Template->pMaterial_InM);

                pDoc->CElementList.AddTail(NewCDeEn_M_Template->pMaterial_OutM);
                pDoc->CEdgeList.AddTail(NewCDeEn_M_Template->pMaterial_OutM);
                pDoc->CMaterialList.AddTail(NewCDeEn_M_Template->pMaterial_OutM);

                pDoc->CElementList.AddTail(NewCDeEn_M_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewCDeEn_M_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewCDeEn_M_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewCDeEn_M_Template->pEnergy_OutE);
                pDoc->CEdgeList.AddTail(NewCDeEn_M_Template->pEnergy_OutE);
                pDoc->CEnergyList.AddTail(NewCDeEn_M_Template->pEnergy_OutE);

        }

        delete pCounterString_F;
        delete pCounterString_InM;
        delete pCounterString_OutM;
        delete pCounterString_InE;
        delete pCounterString_OutE;

        //OnDraw(this->GetDC());
        LButtonIsDown = FALSE;          // Without this line, LButtonIsDown remains set
}
int CConMod2View::DetermineState(CString CSinput, ActuateE_Template* actuatedfunc)
{
        TCHAR szFilters[] = _T("Text Files (*.txt)|*.txt|All Files (*.*)|*.*||");
        CFileDialog* fileDlg = new CFileDialog(TRUE, _T("txt"), _T("*.txt"),
                OFN_HIDEREADONLY | OFN_FILEMUSTEXIST, szFilters);
        if ((*fileDlg).DoModal() == IDOK)
        {
                CString pathName = (*fileDlg).GetPathName();
                CString fileName = (*fileDlg).GetFileTitle();
                CFileException Error;
                CFile* transitions = new CFile;
                if (!(*transitions).Open(pathName, CFile::modeReadWrite, &Error))
                {
                        TRACE(_T("File could not be opened %d\n"), Error.m_cause);
                }
                else
                {
                        CString* strNameValue = new CString;
                        CString* inState = new CString;
                        CString* inCS = new CString;
                        CString* outState = new CString;
                        bool StateRecognized = false;
                        bool CSRecognized = false;
                        CList<CString, CString &> FromState;
                        CList<CString, CString &> CS;
                        CList<CString, CString &> CSedited;
                        CList<CString, CString &> ToState;

                        //Opening the file

                        ULONGLONG* filelength = new ULONGLONG;
```

```
                            *filelength = (*transitions).GetLength();
                            BYTE *buffer = (BYTE *)malloc(*filelength + 1); // Add 1 extra
byte for NULL char
                            (*transitions).Read(buffer, *filelength);  // read character up
to dwLength
                            *(buffer + *filelength) = '\0';  // Make last character NULL so
that not to get garbage
                            CString* sFileContent = new CString((char*)buffer);

                            //Extracting intial state

                            AfxExtractSubString(StartingState, *sFileContent, 0, ';');
                            if ((StartingState) == _T("state1"))
                            {
                                    AfxMessageBox(_T("Start state is the actuated
state\r\n"));
                            }
                            else if ((StartingState) == _T("state0"))
                            {
                                    AfxMessageBox(_T("Start state is the deactuated
state\r\n"));
                            }
                            else
                            {
                                    AfxMessageBox(_T("Error extracting start state\r\n"));
                            }

                            //Extracting 1st, 2nd and 3rd columns to lists

                            CString* stateTransitions = new CString;
                            CString* thirdString = new CString;
                            AfxExtractSubString(*stateTransitions, *sFileContent, 1, ';');
                            int i = 0;
                            while (AfxExtractSubString(*strNameValue, *stateTransitions, i))
                            {
                                    i++;
                                    if (!AfxExtractSubString(*inState, *strNameValue, 0,
_T(',')))
                                    {
                                            AfxMessageBox(_T("Error extracting start
state\r\n"));
                                    }
                                    else
                                    {
                                            FromState.AddTail(*inState);
                                    }
                                    if (!AfxExtractSubString(*inCS, *strNameValue, 1,
_T(',')))
                                    {
                                            AfxMessageBox(_T("Error extracting control
signal\r\n"));
                                    }
                                    else
                                    {
                                            CS.AddTail(*inCS);
                                    }
                                    if (!AfxExtractSubString(*outState, *strNameValue, 2,
_T(',')))
                                    {
```

```
                                                AfxMessageBox(_T("Error extracting end 
state\r\n"));
                                        }
                                        else
                                        {
                                                ToState.AddTail(*outState);
                                        }
                                }

                                // Determination of State

                                for (int x = 0; x < FromState.GetCount(); x++)
                                        for (int y=0; y < CS.GetCount(); y++)
                                        {
                                                int FromStateIndex, CSIndex;
                                                if (StartingState == 
FromState.GetAt(FromState.FindIndex(x)))
                                                {
                                                        FromStateIndex = x;
                                                }
                                                else
                                                        continue;
                                                if (CSinput == CS.GetAt(CS.FindIndex(y)))
                                                {
                                                        CSIndex = y;
                                                }
                                                else
                                                        continue;
                                                if (FromStateIndex == CSIndex)
                                                {
                                                        (*thirdString) = 
ToState.GetAt(ToState.FindIndex(x));
                                                        if ((*thirdString) == _T("state1"))
                                                        {
                                                                AfxMessageBox(_T("The flow is 
Actuated"));

                                                                actuatedfunc->Mode = Actuate;
                                                                actuatedfunc->ModeIsActuate();
                                                                actuatedfunc->EisActuated = 
true;

                                                                actuatedfunc->pEnergy_OutE-
>IsHidden = false;

                                                                transitions->SeekToBegin();
                                                                char c[] = "state1;";
                                                                int size = sizeof(c) / 
sizeof(c[0]);

                                                                transitions->Write(c, size - 
1);

                                                                continue;
                                                        }
                                                        else if ((*thirdString) == 
_T("state0"))

                                                        {
                                                                AfxMessageBox(_T("The flow is 
DeActuated"));

                                                                actuatedfunc->Mode = DeActuate;
                                                                actuatedfunc-
>ModeIsDeActuate();
```

```
                                                            actuatedfunc->EisActuated =
false;
                                                            actuatedfunc->pEnergy_OutE-
>IsHidden = true;
                                                            transitions->SeekToBegin();
                                                            char c[] = "state0;";
                                                            int size = sizeof(c) /
sizeof(c[0]);
                                                            transitions->Write(c, size -
1);
                                                            continue;
                                                    }
                                                    else
                                                    {
                                                            AfxMessageBox(_T("Check 3rd
Column for incorrect end state"));
                                                    }
                                            }
                                            else
                                                    continue;
                                    }


                        // Reasoning on FSA grammar

                        int* StartState0Count = new int;
                        int* StartState1Count = new int;

                        *StartState0Count = 0;
                        *StartState1Count = 0;

                        int unique = 1;

                        for (int outer = 1; outer < CS.GetCount(); ++outer)
                        {
                                int is_unique = 1;
                                for (int inner = 0; is_unique && inner < outer; ++inner)
                                {
                                        if (CS.GetAt(CS.FindIndex(inner)) ==
CS.GetAt(CS.FindIndex(outer))) is_unique = 0;
                                }
                                if (is_unique) ++unique;
                        }

                        for (int i = 0; i < FromState.GetCount(); i++)
                        {
                                CString* currentString = new CString;
                                *currentString =
(FromState.GetAt(FromState.FindIndex(i)));

                                if (*currentString == _T("state0"))
                                {
                                        *StartState0Count = *StartState0Count + 1;
                                }
                                else if (*currentString == _T("state1"))
                                {
                                        *StartState1Count = *StartState1Count + 1;
                                }
                                else
                                {
                                        CString* line = new CString;
                                        (*line).Format(_T("%d"), i);
                                        AfxMessageBox(_T("Check 1st Column for incorrect 
start states"));

                                        delete line;
                                }
                                delete currentString;
                        }

                        double* uniqueCSLines = new double;
                        *uniqueCSLines = (FromState.GetCount() / 2);
                        AfxMessageBox(_T("The control signal in one of the transitions 
is  not a member of the set of valid control signals"));

                        if (unique != *StartState0Count)
                        {
                                AfxMessageBox(_T("The state transition matrix has 
incorrect number of deactuating states in the 1st column"));
                        }
                        else if (unique != *StartState1Count)
                        {
                                AfxMessageBox(_T("The state transition matrix has 
incorrect number of actuating states in the 1st column"));
                        }
                        else if (unique != *uniqueCSLines)
                        {
                                AfxMessageBox(_T("The control signal in one of the 
transitions is  not a member of the set of valid control signals"));
                        }

                        FromState.RemoveAll();
                        CS.RemoveAll();
                        ToState.RemoveAll();
                        (*transitions).Close();
                        free(buffer);

                        delete stateTransitions;
                        delete StartState0Count;
                        delete StartState1Count;
                        delete uniqueCSLines;
                        delete strNameValue;
                        delete inState;
                        delete inCS;
                        delete outState;
                        delete thirdString;
                        delete sFileContent;
                        delete filelength;
                }
                delete transitions;
        }
        delete fileDlg;
        return Mode;
}
void CConMod2View::AddActuateE_Template()
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();
        CString* pCounterString_F = new CString;
        CString* pCounterString_InE = new CString;
        CString* pCounterString_OutE = new CString;
        CString* pCounterString_InS = new CString;
        CString* pCounterString_InCarrier = new CString;
        CString* pCounterString_OutCarrier = new CString;

        Counter_F++;
        pCounterString_F->Format(_T("%d"), Counter_F);
        *pCounterString_F = *pCounterString_F + (_T(" [Actuate_E]"));

        Counter_S++;
        pCounterString_InS->Format(_T("%d"), Counter_S);

        Counter_M++;
        pCounterString_InCarrier->Format(_T("%d"), Counter_M);
        Counter_M++;
        pCounterString_OutCarrier->Format(_T("%d"), Counter_M);

        Counter_E++;
        pCounterString_InCarrier->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutCarrier->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_InE->Format(_T("%d"), Counter_E);
        Counter_E++;
        pCounterString_OutE->Format(_T("%d"), Counter_E);

        if (pSelectedElement == NULL)    // Create only in empty, white space of the
screen - otherwise more
                                                                          //attachment
issues will arise
        {
                ActuateE_Template* NewActuateE_Template = new ActuateE_Template(NULL,
SnapToGrid(MouseLDownPoint),
                        pCounterString_F, pCounterString_InE, pCounterString_OutE,
pCounterString_InS, pCounterString_InCarrier, pCounterString_OutCarrier, this-
>ReasoningOption);

                DetermineState(NewActuateE_Template->pSignal_InS->GivenName,
NewActuateE_Template);
                pDoc->CTemplateList.AddTail(NewActuateE_Template);
                pDoc->ActuateE_Template_List.AddTail(NewActuateE_Template);

                pDoc->CElementList.AddTail(NewActuateE_Template->pFunctionBlock);
                pDoc->CNodeList.AddTail(NewActuateE_Template->pFunctionBlock);
                pDoc->CFunctionList.AddTail(NewActuateE_Template->pFunctionBlock);
                pDoc->ActuateE_Function_List.AddTail(NewActuateE_Template-
>pFunctionBlock);       // Enables grammar checking

                pDoc->CElementList.AddTail(NewActuateE_Template->pSignal_InS);
                pDoc->CEdgeList.AddTail(NewActuateE_Template->pSignal_InS);
                pDoc->CSignalList.AddTail(NewActuateE_Template->pSignal_InS);

                pDoc->CElementList.AddTail(NewActuateE_Template->pEnergy_InE);
                pDoc->CEdgeList.AddTail(NewActuateE_Template->pEnergy_InE);
                pDoc->CEnergyList.AddTail(NewActuateE_Template->pEnergy_InE);

                pDoc->CElementList.AddTail(NewActuateE_Template->pEnergy_OutE);
                pDoc->CEdgeList.AddTail(NewActuateE_Template->pEnergy_OutE);
                pDoc->CEnergyList.AddTail(NewActuateE_Template->pEnergy_OutE);

                if (NewActuateE_Template->CarrierIsEnergy == true)
                {
                        pDoc->CElementList.AddTail(NewActuateE_Template-
>pEnergy_In_CarrierE);
                        pDoc->CEdgeList.AddTail(NewActuateE_Template-
>pEnergy_In_CarrierE);
                        pDoc->CEnergyList.AddTail(NewActuateE_Template-
>pEnergy_In_CarrierE);

                        pDoc->CElementList.AddTail(NewActuateE_Template-
>pEnergy_Out_CarrierE);
                        pDoc->CEdgeList.AddTail(NewActuateE_Template-
>pEnergy_Out_CarrierE);
                        pDoc->CEnergyList.AddTail(NewActuateE_Template-
>pEnergy_Out_CarrierE);

                }
                else if (NewActuateE_Template->CarrierIsMaterial == true)
                {
                        pDoc->CElementList.AddTail(NewActuateE_Template-
>pMaterial_In_CarrierM);
                        pDoc->CEdgeList.AddTail(NewActuateE_Template-
>pMaterial_In_CarrierM);
                        pDoc->CMaterialList.AddTail(NewActuateE_Template-
>pMaterial_In_CarrierM);

                        pDoc->CElementList.AddTail(NewActuateE_Template-
>pMaterial_Out_CarrierM);
                        pDoc->CEdgeList.AddTail(NewActuateE_Template-
>pMaterial_Out_CarrierM);
                        pDoc->CMaterialList.AddTail(NewActuateE_Template-
>pMaterial_Out_CarrierM);

                }
                else
                {
                        return;
                }
        }
        delete pCounterString_F;
        delete pCounterString_InE;
        delete pCounterString_OutE;
        delete pCounterString_InS;
        delete pCounterString_InCarrier;
        delete pCounterString_OutCarrier;

        LButtonIsDown = FALSE;
}

// ============================================================================
// ============================================================================
// FUNCTIONS TO SELECT AND UNSELECT OBJECTS FROM THE MODEL
// ============================================================================
// ============================================================================
void CConMod2View::Preselect(CPoint* pMouseTip)
{
        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        // ============================================================================
        // RESET THE EXISTING CONTAINERS OF PRESELECTION DATA AT EVERY NEW CALL
        // ============================================================================
        pDoc->PreselectionList.RemoveAll();
        ScrollPosition = NULL;
        pSelectedElement = NULL;

        // ============================================================================
        // SPECIAL REQUIREMENT FOR EDGES - CLEAR OFF THE TEMPORARY HEAD AND TAIL NODES
        // ============================================================================
        if (!TailNodeSelected)
                pTailElemDynamic = NULL;
        pHeadElemDynamic = NULL;

        // ============================================================================
        // LOOK FOR PROXIMITY BETWEEN MOUSE TIP AND ALL CELEMENT INSTANCES.
        // FOR ALL PROXIMAL CELEMENT INSTANCES, HIGHLIGHT, ASSIGN GRABHANDLE, AND ADD TO
        // PRESELECTIONLIST (IF NOT ALREADY THERE).
        //              THREE TESTS FOR GRABHANDLE ARE NECESSARY TO PERFORM THIS ACTION.
        // IF NOT PROXIMAL, THEN UNHIGHLIHGT, RESET GRABHANDLE, AND REMOVE FROM
        // PRESELECTIONLIST (IF NOT ALREADY REMOVED).
        // ============================================================================

        if (!pDoc->CElementList.IsEmpty())
        {
                for (POSITION pos = pDoc->CElementList.GetHeadPosition(); pos != NULL; )
                {
                        if (distance(*pMouseTip, pDoc->CElementList.GetAt(pos)-
>GeometricCenter) <= SELECTION_RADIUS)
                        {
                                pDoc->CElementList.GetAt(pos)->GrabHandle = CENTER;
        // Applies to both nodes and edges
                                Highlight(pDoc->CElementList.GetAt(pos));
                                if (!pDoc->PreselectionList.Find(pDoc-
>CElementList.GetAt(pos)))
                                        pDoc->PreselectionList.AddTail(pDoc-
>CElementList.GetAt(pos));
                        }
                        else if (distance(*pMouseTip, pDoc->CElementList.GetAt(pos)-
>HeadPoint) <= SELECTION_RADIUS)
                        {
                                pDoc->CElementList.GetAt(pos)->GrabHandle = HEAD;//
Applies to edges
                                Highlight(pDoc->CElementList.GetAt(pos));
                                if (!pDoc->PreselectionList.Find(pDoc-
>CElementList.GetAt(pos)))
                                        pDoc->PreselectionList.AddTail(pDoc-
>CElementList.GetAt(pos));
                        }
                        else if (distance(*pMouseTip, pDoc->CElementList.GetAt(pos)-
>TailPoint) <= SELECTION_RADIUS)
                        {
                                pDoc->CElementList.GetAt(pos)->GrabHandle = TAIL;//
Applies to edges
                                Highlight(pDoc->CElementList.GetAt(pos));
                                if (!pDoc->PreselectionList.Find(pDoc-
>CElementList.GetAt(pos)))
                                        pDoc->PreselectionList.AddTail(pDoc-
>CElementList.GetAt(pos));
                        }
                        else
                        {
                                UnHighlight(pDoc->CElementList.GetAt(pos));
                                pDoc->CElementList.GetAt(pos)->GrabHandle = NULL;
                                if (pDoc->PreselectionList.Find(pDoc-
>CElementList.GetAt(pos)))
                                        pDoc->PreselectionList.RemoveAt(pDoc-
>PreselectionList.Find(pDoc->CElementList.GetAt(pos)));  // (pDoc-
>CElementList.GetAt(pos));
                        }

                        pDoc->CElementList.GetNext(pos);
                }
        }

        // ============================================================================
        // GET READY FOR SCROLLING:
        // IF PreselectionList HAS THINGS IN IT, SELECT THE FIRST ITEM AND SET
        // ScrollPosition AS THE HEAD POSITION WITHIN THAT LIST.
        // OTHERWISE, THE EXISTING NULL VALUES SET AT THE BEGINNING OF THIS FUNCTION
        // CALL WILL PREVAIL.
        // ============================================================================

        if (!pDoc->PreselectionList.IsEmpty())
        {
                SelectElement(pDoc->PreselectionList.GetHead()); // stroes
pSelectedElement
                ScrollPosition = pDoc->PreselectionList.GetHeadPosition();

                // SCPECIAL CASE - IF ADDING AN EDGE, STORE ITS temporary TAIL and HEAD
                if ((WhatToDo == ADD_ENERGY) || (WhatToDo == ADD_MATERIAL) || (WhatToDo
== ADD_SIGNAL))
                {
                        if (LButtonIsDown)
                                pHeadElemDynamic = pSelectedElement;
                        else
                                pTailElemDynamic = pSelectedElement;

                        if (pHeadElemDynamic == pTailElemDynamic)
                                pHeadElemDynamic = NULL; // Prevents self-cycling edges
                }
        }

        // Finally, redraw the screen
        //OnDraw(this->GetDC());
}

void CConMod2View::Highlight(CElement* pElement)
{
        pElement->IsHighlighted = true;
        pElement->IsSelected = false;
}

void CConMod2View::UnHighlight(CElement* pElement)
```

```
{
        pElement->IsHighlighted = false;
        pElement->IsSelected = false;
}

void CConMod2View::SelectElement(CElement* pElement)
{
        pSelectedElement = pElement;
        pElement->IsSelected = true;
        pElement->IsHighlighted = false;
}

void CConMod2View::ScrollThroughPreselection()
{
        CConMod2Doc* pDoc = GetDocument();

        // Reset the current selection to PRESELECTION_PEN_ colors
        Highlight(pDoc->PreselectionList.GetAt(ScrollPosition));

        // If the tail of PreselectionList has arrived, start over at the head
        if (ScrollPosition == pDoc->PreselectionList.GetTailPosition())
                ScrollPosition = pDoc->PreselectionList.GetHeadPosition();
        else
                pDoc->PreselectionList.GetNext(ScrollPosition);

        // Select the element at this incremented ScrollPosition
        SelectElement(pDoc->PreselectionList.GetAt(ScrollPosition));

        //OnDraw(this->GetDC());

        // ===========================================================================
        // SCPCIAL CASE - IF ADDING EDGE, STORE ITS TAIL NODE and HEAD NODE.
        // AN IDENTICAL IF STATEMENT IS ALSO USED IN Preselect, TO ENABLE THE
        // SAME FEATURES IF TEH USER SELECTED THE FIRST SELECTED ELEMENT WITHOUT
        // SCROLLING.
        // ===========================================================================
        if (WhatToDo == ADD_ENERGY || WhatToDo == ADD_MATERIAL || WhatToDo ==
ADD_SIGNAL)
        {
                if (LButtonIsDown)
                        pHeadElemDynamic = pSelectedElement;
                else
                        pTailElemDynamic = pSelectedElement;

                if (pHeadElemDynamic == pTailElemDynamic)
                        pHeadElemDynamic = NULL; // Prevents self-cycling edges
        }
}

bool CConMod2View::ElementIsNode(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CNodeList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CNodeList.GetAt(pos) == pElement)
                {
                        NodeIndexInNodeList = pos;
                        return true;
                }
                pDoc->CNodeList.GetNext(pos);
        }
        return false;
}
bool CConMod2View::ElementIsFunction(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CFunctionList.GetAt(pos) == pElement)
                {
                        FunctionIndexInFunctionList = pos;
                        return true;
                }

                pDoc->CFunctionList.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsEnv(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CEnvList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CEnvList.GetAt(pos) == pElement)
                {
                        EnvIndexInEnvList = pos;
                        return true;
                }

                pDoc->CEnvList.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsEdge(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CEdgeList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CEdgeList.GetAt(pos) == pElement)
                {
                        EdgeIndexInEdgeList = pos;
                        return true;
                }

                pDoc->CEdgeList.GetNext(pos);
        }

        return false;
}
```

```
bool CConMod2View::ElementIsMaterial(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CMaterialList.GetAt(pos) == pElement)
                {
                        MaterialIndexInMaterialList = pos;
                        return true;
                }

                pDoc->CMaterialList.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsEnergy(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CEnergyList.GetAt(pos) == pElement)
                {
                        EnergyIndexInEnergyList = pos;
                        return true;
                }

                pDoc->CEnergyList.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsSignal(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CSignalList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CSignalList.GetAt(pos) == pElement)
                {
                        SignalIndexInSignalList = pos;
                        return true;
                }

                pDoc->CSignalList.GetNext(pos);
        }

        return false;
}



bool CConMod2View::ElementIsConvert_E_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();
        for (POSITION pos = pDoc->CConvert_E_Template_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CConvert_E_Template_List.GetAt(pos) == pElement)
                {
                        Convert_E_Template_IndexInConvert_E_Template_List = pos;
                        return true;
                }

                pDoc->CConvert_E_Template_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsConduct_E_Function(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CConduct_E_Function_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CConduct_E_Function_List.GetAt(pos) == pElement)
                {
                        Conduct_E_Function_IndexInConduct_E_Function_List = pos;
                        return true;
                }

                pDoc->CConduct_E_Function_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsConduct_E_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CConduct_E_Template_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CConduct_E_Template_List.GetAt(pos) == pElement)
                {
                        Conduct_E_Template_IndexInConduct_E_Template_List = pos;
                        return true;
                }

                pDoc->CConduct_E_Template_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsEnergize_M_Function(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();
        for (POSITION pos = pDoc->CEnergize_M_Function_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CEnergize_M_Function_List.GetAt(pos) == pElement)
                {
                        Energize_M_Function_IndexInEnergize_M_Function_List = pos;
                        return true;
                }

                pDoc->CEnergize_M_Function_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsEnergize_M_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CEnergize_M_Template_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CEnergize_M_Template_List.GetAt(pos) == pElement)
                {
                        Energize_M_Template_IndexInEnergize_M_Template_List = pos;
                        return true;
                }

                pDoc->CEnergize_M_Template_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsDistribute_E_Function(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CDistribute_E_Function_List.GetHeadPosition(); pos !=
NULL; )
        {
                if (pDoc->CDistribute_E_Function_List.GetAt(pos) == pElement)
                {
                        Distribute_E_Function_IndexInDistribute_E_Function_List = pos;
                        return true;
                }

                pDoc->CDistribute_E_Function_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsDistribute_E_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CDistribute_E_Template_List.GetHeadPosition(); pos !=
NULL; )
```

```
        {
                if (pDoc->CDistribute_E_Template_List.GetAt(pos) == pElement)
                {
                        Distribute_E_Template_IndexInDistribute_E_Template_List = pos;
                        return true;
                }

                pDoc->CDistribute_E_Template_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsDeEn_M_Function(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CDeEn_M_Function_List.GetHeadPosition(); pos != NULL;
)
        {
                if (pDoc->CDeEn_M_Function_List.GetAt(pos) == pElement)
                {
                        DeEn_M_Function_IndexInDeEn_M_Function_List = pos;
                        return true;
                }

                pDoc->CDeEn_M_Function_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsDeEn_M_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CDeEn_M_Template_List.GetHeadPosition(); pos != NULL;
)
        {
                if (pDoc->CDeEn_M_Template_List.GetAt(pos) == pElement)
                {
                        DeEn_M_Template_IndexInDeEn_M_Template_List = pos;
                        return true;
                }

                pDoc->CDeEn_M_Template_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsActuateE_Function(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->ActuateE_Function_List.GetHeadPosition(); pos != NULL;
)
        {
                if (pDoc->ActuateE_Function_List.GetAt(pos) == pElement)
                {
                        ActuateE_Function_IndexInDeActuateE_Function_List = pos;
                        return true;
                }

                pDoc->ActuateE_Function_List.GetNext(pos);
        }

        return false;
}

bool CConMod2View::ElementIsActuateE_Template(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->ActuateE_Template_List.GetHeadPosition(); pos != NULL;
)
        {
                if (pDoc->ActuateE_Template_List.GetAt(pos) == pElement)
                {
                        ActuateE_Template_IndexInActuateE_Template_List = pos;
                        return true;
                }

                pDoc->ActuateE_Template_List.GetNext(pos);
        }

        return false;
}
// ============================================================================
// ============================================================================
// FUNCTIONS TO EDIT OBJECTS WITHIN THE MODEL
// ============================================================================
// ============================================================================

void CConMod2View::MoveConnectDynamic()  // Called by OnMouseMove
{
        CConMod2Doc* pDoc = GetDocument();
        GrammarCheckRequired = false;           // This callis very important - without
it,
                                                                                  // the
grammr checks for topological error will take effect DURING
                                                                                  // the
move / connect operation BEFORE LIFTING UP THE MOUSE L
                                                                                  //
BUTTON and throw errors for topology that the user has not
                                                                                  //
committed to (by lifting mouse L button)

                                                                                  //
============================================================================
                                                                                  // THIS
IS A BASIC CHECK THAT AN ELEMENT IS SELECTED FOR MOVE OR CONNECT.
                                                                                  //
PRACTICALLY, THIS CHECK IS REDUNDANT, SINCE THE ONLY CALLING FUNCTION
                                                                                  // OF
THIS FUNCTION, OnMouseMove, MAKES SURE THAT AN ELEMENT IS INDEED SELECTED.
                                                                                  //
============================================================================
        if (!LButtonIsDown || pSelectedElement == NULL)
                return;

        // ===========================================================================
        // IF AN EDGE IS ANCHORED ON ANY ONE SIDE, PREVENT MOVING IT BY ITS CENTER
        // ===========================================================================
        if ((ElementIsEdge(pSelectedElement))
                &&
                ((pSelectedElement->pHeadElem != NULL)
                        ||
                        (pSelectedElement->pTailElem != NULL))
                &&
                (pSelectedElement->GrabHandle == CENTER))
                return;

        Invalidate();

        // ===========================================================================
        // MOVE NODES AND DOUBLY-DANLGING EDGES BY THE CENTER GRABHANDLE
        // ===========================================================================

        if (pSelectedElement->GrabHandle == CENTER)              // Works for nodes and
edges with both ends dangling
        {
                // First, compute the orientation and length of the arrow using its
existing
                // center, tail and head points.  This check will workk even for the
nodes, although
                // that would not mean anything real.  So, it is unnecessary to check
that the element
                // is an edge.
                long HalfDeltaX = pSelectedElement->HeadPoint.x - pSelectedElement-
>GeometricCenter.x;
                long HalfDeltaY = pSelectedElement->HeadPoint.y - pSelectedElement-
>GeometricCenter.y;

                // Then, move the center point.  This moves nodes directly.  For edges,
the ends
                // need to be recalculated, as done next.
                pSelectedElement->GeometricCenter = MouseMovePoint;

                // Then re-compute the new head and tail poitns based on the new center
point.
                pSelectedElement->HeadPoint.x = pSelectedElement->GeometricCenter.x +
HalfDeltaX;
                pSelectedElement->HeadPoint.y = pSelectedElement->GeometricCenter.y +
HalfDeltaY;
                pSelectedElement->TailPoint.x = pSelectedElement->GeometricCenter.x -
HalfDeltaX;
                pSelectedElement->TailPoint.y = pSelectedElement->GeometricCenter.y -
HalfDeltaY;
        }

        // ===========================================================================
        // MOVE AND/OR CONNECT THE head POINT OF AN EDGE
        // ===========================================================================

        if (ElementIsEdge(pSelectedElement) && (pSelectedElement->GrabHandle == HEAD))
        {
                pSelectedElement->pHeadElem = NULL;

                for (POSITION pos = pDoc->CElementList.GetHeadPosition(); pos != NULL; )
                {
                        if (distance(MouseMovePoint, pDoc->CElementList.GetAt(pos)-
>GeometricCenter) <= SELECTION_RADIUS)
                        {
                                Highlight(pDoc->CElementList.GetAt(pos));
                                pSelectedElement->pHeadElem = pDoc-
>CElementList.GetAt(pos);
                        }
                        else if ((distance(MouseMovePoint, pDoc-
>CElementList.GetAt(pos)->TailPoint) <= SELECTION_RADIUS) &&
                                (ElementIsEdge(pDoc->CElementList.GetAt(pos))) &&
                                (pDoc->CElementList.GetAt(pos)->pHeadElem != NULL) &&
                                (pDoc->CElementList.GetAt(pos)->pTailElem == NULL))
                        {
                                Highlight(pDoc->CElementList.GetAt(pos));
                                pSelectedElement->pHeadElem = pDoc-
>CElementList.GetAt(pos)->pHeadElem;
                                pElementToBeDeleted = pDoc->CElementList.GetAt(pos);
                        }
                        else
                        {
                                UnHighlight(pDoc->CElementList.GetAt(pos));
                                pSelectedElement->HeadPoint = MouseMovePoint;
                                if (pDoc->CElementList.GetAt(pos) ==
pElementToBeDeleted)
                                        pElementToBeDeleted = NULL;
                        }

                        pDoc->CElementList.GetNext(pos);
                }
        }

        // ===========================================================================
        // MOVE AND/OR CONNECT THE tail POINT OF AN EDGE
        // ===========================================================================

        if (ElementIsEdge(pSelectedElement) && (pSelectedElement->GrabHandle == TAIL))
        {
                pSelectedElement->pTailElem = NULL;

                for (POSITION pos = pDoc->CElementList.GetHeadPosition(); pos != NULL; )
                {
                        if (distance(MouseMovePoint, pDoc->CElementList.GetAt(pos)-
>GeometricCenter) <= SELECTION_RADIUS)
                        {
                                Highlight(pDoc->CElementList.GetAt(pos));
                                pSelectedElement->pTailElem = pDoc-
>CElementList.GetAt(pos);
                        }
                        else if ((distance(MouseMovePoint, pDoc-
>CElementList.GetAt(pos)->HeadPoint) <= SELECTION_RADIUS) &&
                                (ElementIsEdge(pDoc->CElementList.GetAt(pos))) &&
                                (pDoc->CElementList.GetAt(pos)->pTailElem != NULL) &&
                                (pDoc->CElementList.GetAt(pos)->pHeadElem == NULL))
                        {
                                Highlight(pDoc->CElementList.GetAt(pos));
```

```
                                        pElementToBeDeleted = pDoc->CElementList.GetAt(pos);
                                }
                                else
                                {
                                        UnHighlight(pDoc->CElementList.GetAt(pos));
                                        pSelectedElement->TailPoint = MouseMovePoint;
                                        if (pDoc->CElementList.GetAt(pos) ==
pElementToBeDeleted)
                                                pElementToBeDeleted = NULL;
                                }

                                pDoc->CElementList.GetNext(pos);
                        }
        }

        // OnDraw(this->GetDC());        // Do NOT call OnDraw here - it will fire the
        // grammar checks before the move/connect is complete
}

void CConMod2View::MoveConnect()         // Called by OnLButtonUp, when moving edges
(ESCAPE)
{
        if (pSelectedElement == NULL)
                return;

        CConMod2Doc* pDoc = GetDocument();
        Invalidate();

        // SNAP THE NODES TO THE GRID AFTER MOVE IS OVER, WHEN L-BUTTON IS LIFTED
        if (ElementIsNode(pSelectedElement))
                pSelectedElement->GeometricCenter = SnapToGrid(pSelectedElement-
>GeometricCenter);

        if (pElementToBeDeleted != NULL)
        {
                DeleteElement(pElementToBeDeleted);
                pElementToBeDeleted = NULL;
        }

        GrammarCheckRequired = true;

        //OnDraw(this->GetDC());
}

void CConMod2View::DetachEdgesFromElement(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();

        for (POSITION pos = pDoc->CEdgeList.GetHeadPosition(); pos != NULL; )
        {
                if (pElement == pDoc->CEdgeList.GetAt(pos)->pHeadElem)
                        pDoc->CEdgeList.GetAt(pos)->pHeadElem = NULL;
                if (pElement == pDoc->CEdgeList.GetAt(pos)->pTailElem)
                        pDoc->CEdgeList.GetAt(pos)->pTailElem = NULL;

                pDoc->CEdgeList.GetNext(pos);
        }
}
// ============================================================================
// ============================================================================
// FUNCTIONS TO EDIT OBJECTS WITHIN THE MODEL
// ============================================================================
// ============================================================================
void CConMod2View::DeleteElement(CElement* pElement)
{
        CConMod2Doc* pDoc = GetDocument();
        DetachEdgesFromElement(pElement);

        delete pElement;// Deletes the actual instance of the element
                                        // pointed by pElement

        POSITION pos = pDoc->CElementList.Find(pElement);
        pDoc->CElementList.RemoveAt(pos);       // Removes the pointer entry from
CElementList

        if (ElementIsNode(pElement))
                pDoc->CNodeList.RemoveAt(NodeIndexInNodeList);   // Removes the pointer
entry from CNodeList

        if (ElementIsFunction(pElement))
                pDoc->CFunctionList.RemoveAt(FunctionIndexInFunctionList);       //
Removes the pointer entry from CFunctionList

        if (ElementIsEnv(pElement))
                pDoc->CEnvList.RemoveAt(EnvIndexInEnvList);      // Removes the pointer
entry from CFunctionList

        if (ElementIsEdge(pElement))
                pDoc->CEdgeList.RemoveAt(EdgeIndexInEdgeList);   // Removes the pointer
entry from CEdgeList

        if (ElementIsMaterial(pElement))
                pDoc->CMaterialList.RemoveAt(MaterialIndexInMaterialList);       //
Removes the pointer entry from CEdgeList

        if (ElementIsEnergy(pElement))
                pDoc->CEnergyList.RemoveAt(EnergyIndexInEnergyList);     // Removes the
pointer entry from CEdgeList

        if (ElementIsSignal(pElement))
                pDoc->CSignalList.RemoveAt(SignalIndexInSignalList);     // Removes the
pointer entry from CEdgeList

}


// ============================================================================
// ============================================================================
// FUNCTIONS FOR DERIVATIONAL TOPOLOGICAL CONSERVATION CHECKS
// ============================================================================
// ============================================================================

void CConMod2View::Set_OrphanFlowMsg()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_OrphanFlow = "";
        CString* pEdgeNames = new CString;
        *pEdgeNames = _T("");

        for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CMaterialList.GetAt(pos)->ParentList.IsEmpty() &&
!ElementIsEnv(pDoc->CMaterialList.GetAt(pos)->pTailElem))
                        *pEdgeNames = *pEdgeNames + _T(", ") + pDoc-
>CMaterialList.GetAt(pos)->GivenName;

                pDoc->CMaterialList.GetNext(pos);
        }

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CEnergyList.GetAt(pos)->ParentList.IsEmpty() &&
!ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pTailElem) && !(pDoc->CEnergyList.GetAt(pos)-
>ThisFlowIsIncomingBaggage))
                        *pEdgeNames = *pEdgeNames + _T(", ") + pDoc-
>CEnergyList.GetAt(pos)->GivenName;

                pDoc->CEnergyList.GetNext(pos);
        }

        if (*pEdgeNames != "")
                Msg_OrphanFlow = _T("\nOrphan Flow Detected: ") + *pEdgeNames + _T(".");

        delete pEdgeNames;
}

void CConMod2View::Set_BarrenFlowMsg()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_BarrenFlow = "";

        CString* pEdgeNames = new CString;
        *pEdgeNames = _T("");

        for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CMaterialList.GetAt(pos)->ChildList.IsEmpty() &&
!ElementIsEnv(pDoc->CMaterialList.GetAt(pos)->pHeadElem))
                        *pEdgeNames = *pEdgeNames + _T(", ") + pDoc-
>CMaterialList.GetAt(pos)->GivenName;

                pDoc->CMaterialList.GetNext(pos);
        }

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->CEnergyList.GetAt(pos)->ChildList.IsEmpty() &&
!ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pHeadElem) && !(pDoc->CEnergyList.GetAt(pos)-
>ThisFlowIsOutgoingBaggage))
                        *pEdgeNames = *pEdgeNames + _T(", ") + pDoc-
>CEnergyList.GetAt(pos)->GivenName;

                pDoc->CEnergyList.GetNext(pos);
        }
        if (*pEdgeNames != "")
                Msg_BarrenFlow = Msg_BarrenFlow + _T("\nBarren Flow Detected: ") +
*pEdgeNames + _T(".");

        delete pEdgeNames;
}

void CConMod2View::Set_OneInManyOutMsg_M()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_OneInManyOut_M = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                CString* pInputEdgeName = new CString;
                *pInputEdgeName = _T("");
                CString *pOutputEdgeNames = new CString;
                *pOutputEdgeNames = _T("");


        //============================================================================
                // Inference of MATERIAL conservation - One In Many Out

        //============================================================================
                for (POSITION pos1 = pDoc->CMaterialList.GetHeadPosition(); pos1 !=
NULL; )
                {
                        if (pDoc->CMaterialList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                pDoc->CMaterialList_IN_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos1));

                        pDoc->CMaterialList.GetNext(pos1);
                }

                if (pDoc->CMaterialList_IN_TEMP.GetCount() == 1)
                {
                        for (POSITION pos2 = pDoc->CMaterialList.GetHeadPosition(); pos2
!= NULL; )
                        {
                                if (pDoc->CMaterialList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                        pDoc->CMaterialList_OUT_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos2));

                                pDoc->CMaterialList.GetNext(pos2);
                        }

                        if (pDoc->CMaterialList_OUT_TEMP.GetCount() >= 1)
                        {
                                *pInputEdgeName = pDoc->CMaterialList_IN_TEMP.GetHead()-
>GivenName;

                                for (POSITION pos3 = pDoc-
>CMaterialList_OUT_TEMP.GetHeadPosition(); pos3 != NULL; )
                                {
                                        pDoc->CMaterialList_IN_TEMP.GetHead()-
>ChildList.AddTail(pDoc->CMaterialList_OUT_TEMP.GetAt(pos3));
                                        pDoc->CMaterialList_OUT_TEMP.GetAt(pos3)-
>ParentList.AddTail(pDoc->CMaterialList_IN_TEMP.GetHead());
                                        *pOutputEdgeNames = *pOutputEdgeNames + _T(", ")
+ pDoc->CMaterialList_OUT_TEMP.GetAt(pos3)->GivenName;

                                        pDoc->CMaterialList_OUT_TEMP.GetNext(pos3);
                                }

                                Msg_OneInManyOut_M = Msg_OneInManyOut_M + _T("\nInferred
Derivations: {") + *pInputEdgeName + _T("} --> {") + *pOutputEdgeNames + _T("}.");
                        }
                }

                delete pInputEdgeName;
                delete pOutputEdgeNames;
                EmptyAllTempLists();     // For every function block

                pDoc->CFunctionList.GetNext(pos);
        }
}

void CConMod2View::Set_OneInManyOutMsg_E()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_OneInManyOut_E = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                CString* pInputEdgeName = new CString;
                *pInputEdgeName = _T("");
                CString *pOutputEdgeNames = new CString;
                *pOutputEdgeNames = _T("");


        //============================================================================
                // Inference of MATERIAL conservation - One In Many Out

        //============================================================================
                for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                {
                        if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                pDoc->CEnergyList_IN_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos1));

                        pDoc->CEnergyList.GetNext(pos1);
                }

                if (pDoc->CEnergyList_IN_TEMP.GetCount() == 1)
                {
                        for (POSITION pos2 = pDoc->CEnergyList.GetHeadPosition(); pos2
!= NULL; )
                        {
                                if (pDoc->CEnergyList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
```

```
                                        pDoc->CEnergyList_OUT_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos2));

                                pDoc->CEnergyList.GetNext(pos2);
                        }

                        if (pDoc->CEnergyList_OUT_TEMP.GetCount() >= 1)
                        {
                                *pInputEdgeName = pDoc->CEnergyList_IN_TEMP.GetHead()-
>GivenName;

                                for (POSITION pos3 = pDoc-
>CEnergyList_OUT_TEMP.GetHeadPosition(); pos3 != NULL; )
                                {
                                        pDoc->CEnergyList_IN_TEMP.GetHead()-
>ChildList.AddTail(pDoc->CEnergyList_OUT_TEMP.GetAt(pos3));
                                        pDoc->CEnergyList_OUT_TEMP.GetAt(pos3)-
>ParentList.AddTail(pDoc->CEnergyList_IN_TEMP.GetHead());
                                        *pOutputEdgeNames = *pOutputEdgeNames + _T(", ")
+ pDoc->CEnergyList_OUT_TEMP.GetAt(pos3)->GivenName;

                                        pDoc->CEnergyList_OUT_TEMP.GetNext(pos3);
                                }

                                Msg_OneInManyOut_E = Msg_OneInManyOut_E + _T("\nInferred
Derivations: {") + *pInputEdgeName + _T("} --> {") + *pOutputEdgeNames + _T("}.");
                        }
                }

                delete pInputEdgeName;
                delete pOutputEdgeNames;
                EmptyAllTempLists();    // For every function block

                pDoc->CFunctionList.GetNext(pos);
        }
}
void CConMod2View::Set_ManyInOneOutMsg_M()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_ManyInOneOut_M = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                CString *pInputEdgeNames = new CString;
                *pInputEdgeNames = _T("");
                CString *pOutputEdgeName = new CString;
                *pOutputEdgeName = _T("");


        //=============================================================================
                // Inference of MATERIAL conservation - Many In One Out

        //=============================================================================
                for (POSITION pos1 = pDoc->CMaterialList.GetHeadPosition(); pos1 !=
NULL; )
                {
                        if (pDoc->CMaterialList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
```

```
                                        pDoc->CMaterialList_IN_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos1));

                                pDoc->CMaterialList.GetNext(pos1);
                        }

                        if (pDoc->CMaterialList_IN_TEMP.GetCount() > 1)
                        {
                                for (POSITION pos2 = pDoc->CMaterialList.GetHeadPosition(); pos2
!= NULL; )
                                {
                                        if (pDoc->CMaterialList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                                pDoc->CMaterialList_OUT_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos2));

                                        pDoc->CMaterialList.GetNext(pos2);
                                }

                                if (pDoc->CMaterialList_OUT_TEMP.GetCount() == 1)
                                {
                                        *pOutputEdgeName = pDoc-
>CMaterialList_OUT_TEMP.GetHead()->GivenName;

                                        for (POSITION pos3 = pDoc-
>CMaterialList_IN_TEMP.GetHeadPosition(); pos3 != NULL; )
                                        {
                                                pDoc->CMaterialList_OUT_TEMP.GetHead()-
>ParentList.AddTail(pDoc->CMaterialList_IN_TEMP.GetAt(pos3));
                                                pDoc->CMaterialList_IN_TEMP.GetAt(pos3)-
>ChildList.AddTail(pDoc->CMaterialList_OUT_TEMP.GetHead());
                                                *pInputEdgeNames = *pInputEdgeNames + _T(", ") +
pDoc->CMaterialList_IN_TEMP.GetAt(pos3)->GivenName;

                                                pDoc->CMaterialList_IN_TEMP.GetNext(pos3);
                                        }

                                        Msg_ManyInOneOut_M = Msg_ManyInOneOut_M + _T("\nInferred
Derivations: {") + *pInputEdgeNames + _T("} --> {") + *pOutputEdgeName + _T("}.");
                                }
                        }
                        delete pInputEdgeNames;
                        delete pOutputEdgeName;
                        EmptyAllTempLists();    // For every function block
                        pDoc->CFunctionList.GetNext(pos);
                }
}

void CConMod2View::Set_ManyInOneOutMsg_E()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_ManyInOneOut_E = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                CString *pInputEdgeNames = new CString;
                *pInputEdgeNames = _T("");
                CString *pOutputEdgeName = new CString;
```

```
                 *pOutputEdgeName = _T("");


         //=============================================================================
                 // Inference of MATERIAL conservation - Many In One Out

         //=============================================================================
                 for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                 {
                         if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                 pDoc->CEnergyList_IN_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos1));

                         pDoc->CEnergyList.GetNext(pos1);
                 }

                 if (pDoc->CEnergyList_IN_TEMP.GetCount() > 1)
                 {
                         for (POSITION pos2 = pDoc->CEnergyList.GetHeadPosition(); pos2
!= NULL; )
                         {
                                 if (pDoc->CEnergyList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                         pDoc->CEnergyList_OUT_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos2));

                                 pDoc->CEnergyList.GetNext(pos2);
                         }

                         if (pDoc->CEnergyList_OUT_TEMP.GetCount() == 1)
                         {
                                 *pOutputEdgeName = pDoc->CEnergyList_OUT_TEMP.GetHead()-
>GivenName;

                                 for (POSITION pos3 = pDoc-
>CEnergyList_IN_TEMP.GetHeadPosition(); pos3 != NULL; )
                                 {
                                         pDoc->CEnergyList_OUT_TEMP.GetHead()-
>ParentList.AddTail(pDoc->CEnergyList_IN_TEMP.GetAt(pos3));
                                         pDoc->CEnergyList_IN_TEMP.GetAt(pos3)-
>ChildList.AddTail(pDoc->CEnergyList_OUT_TEMP.GetHead());
                                         *pInputEdgeNames = *pInputEdgeNames + _T(", ") +
pDoc->CEnergyList_IN_TEMP.GetAt(pos3)->GivenName;

                                         pDoc->CEnergyList_IN_TEMP.GetNext(pos3);
                                 }

                                 Msg_ManyInOneOut_M = Msg_ManyInOneOut_M + _T("\nInferred
Derivations: {") + *pInputEdgeNames + _T("} --> {") + *pOutputEdgeName + _T("}.");
                         }
                 }
                 delete pInputEdgeNames;
                 delete pOutputEdgeName;
                 EmptyAllTempLists();     // For every function block
                 pDoc->CFunctionList.GetNext(pos);
         }
}
```

```
void CConMod2View::Set_ManyInManyOutMsg()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_ManyInManyOut = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                CString *pInputEdgeNames = new CString;
                *pInputEdgeNames = _T("");
                CString *pOutputEdgeNames = new CString;
                *pOutputEdgeNames = _T("");


        //=============================================================================
                // Inference of Impossible Conclusion - Many In Many Out

        //=============================================================================
                for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                {
                        if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                pDoc->CEnergyList_IN_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos1));

                        pDoc->CEnergyList.GetNext(pos1);
                }

                if (pDoc->CEnergyList_IN_TEMP.GetCount() > 1)    // Only then you
investigate further, otherwise don't waste time
                {
                        for (POSITION pos2 = pDoc->CEnergyList.GetHeadPosition(); pos2
!= NULL; )
                        {
                                if (pDoc->CEnergyList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                        pDoc->CEnergyList_OUT_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos2));

                                pDoc->CEnergyList.GetNext(pos2);
                        }

                        if (pDoc->CEnergyList_OUT_TEMP.GetCount() > 1)   // Now both
sides have too many flows to conclude
                        {
                                for (POSITION pos3 = pDoc-
>CEnergyList_IN_TEMP.GetHeadPosition(); pos3 != NULL; )
                                {
                                        for (POSITION pos4 = pDoc-
>CEnergyList_OUT_TEMP.GetHeadPosition(); pos4 != NULL; )
                                        {
                                                pDoc->CEnergyList_OUT_TEMP.GetAt(pos4)-
>ParentList.AddTail(pDoc->CEnergyList_IN_TEMP.GetAt(pos3));
                                                pDoc->CEnergyList_IN_TEMP.GetAt(pos3)-
>ChildList.AddTail(pDoc->CEnergyList_OUT_TEMP.GetAt(pos4));
                                                pDoc-
>CEnergyList_OUT_TEMP.GetNext(pos4);
```

```
                                          }
                                          *pInputEdgeNames = *pInputEdgeNames + _T(", ") +
pDoc->CEnergyList_IN_TEMP.GetAt(pos3)->GivenName;

                                          pDoc->CEnergyList_IN_TEMP.GetNext(pos3);
                                  }/**/

                                  for (POSITION pos5 = pDoc-
>CEnergyList_OUT_TEMP.GetHeadPosition(); pos5 != NULL; )
                                  {
                                          *pOutputEdgeNames = *pOutputEdgeNames + _T(", ")
+ pDoc->CEnergyList_OUT_TEMP.GetAt(pos5)->GivenName;
                                          pDoc->CEnergyList_OUT_TEMP.GetNext(pos5);
                                  }

                                  Msg_ManyInManyOut = Msg_ManyInManyOut + _T("\nInferred
Derivations: {") + *pInputEdgeNames + _T("} --> {") + *pOutputEdgeNames + _T("}.");
                          }
                  }

                  *pInputEdgeNames = _T("");
                  *pOutputEdgeNames = _T("");
                  EmptyAllTempLists();    // For every function block

                  for (POSITION pos1 = pDoc->CMaterialList.GetHeadPosition(); pos1 !=
NULL; )
                  {
                          if (pDoc->CMaterialList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                  pDoc->CMaterialList_IN_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos1));

                          pDoc->CMaterialList.GetNext(pos1);
                  }

                  if (pDoc->CMaterialList_IN_TEMP.GetCount() > 1)  // Only then you
investigate further, otherwise don't waste time
                  {
                          for (POSITION pos2 = pDoc->CMaterialList.GetHeadPosition(); pos2
!= NULL; )
                          {
                                  if (pDoc->CMaterialList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                          pDoc->CMaterialList_OUT_TEMP.AddTail(pDoc-
>CMaterialList.GetAt(pos2));

                                  pDoc->CMaterialList.GetNext(pos2);
                          }

                          if (pDoc->CMaterialList_OUT_TEMP.GetCount() > 1) // Now both
sides have too many flows to conclude
                          {
                                  for (POSITION pos3 = pDoc-
>CMaterialList_IN_TEMP.GetHeadPosition(); pos3 != NULL; )
                                  {
                                          for (POSITION pos4 = pDoc-
>CMaterialList_OUT_TEMP.GetHeadPosition(); pos4 != NULL; )
                                          {
                                                pDoc-
>CMaterialList_OUT_TEMP.GetAt(pos4)->ParentList.AddTail(pDoc-
>CMaterialList_IN_TEMP.GetAt(pos3));
                                                pDoc-
>CMaterialList_IN_TEMP.GetAt(pos3)->ChildList.AddTail(pDoc-
>CMaterialList_OUT_TEMP.GetAt(pos4));
                                                pDoc-
>CMaterialList_OUT_TEMP.GetNext(pos4);
                                        }
                                        *pInputEdgeNames = *pInputEdgeNames + _T(", ") +
pDoc->CMaterialList_IN_TEMP.GetAt(pos3)->GivenName;

                                        pDoc->CMaterialList_IN_TEMP.GetNext(pos3);
                                }/**/

                                for (POSITION pos5 = pDoc-
>CMaterialList_OUT_TEMP.GetHeadPosition(); pos5 != NULL; )
                                {
                                        *pOutputEdgeNames = *pOutputEdgeNames + _T(", ")
+ pDoc->CMaterialList_OUT_TEMP.GetAt(pos5)->GivenName;
                                        pDoc->CMaterialList_OUT_TEMP.GetNext(pos5);
                                }

                                Msg_ManyInManyOut = Msg_ManyInManyOut + _T("\nInferred
Derivations: {") + (*pInputEdgeNames) + _T("} --> {") + *pOutputEdgeNames + _T("}.");
                        }
                }

                delete pInputEdgeNames;
                delete pOutputEdgeNames;
                EmptyAllTempLists();    // For every function block
                pDoc->CFunctionList.GetNext(pos);
        }
}
void CConMod2View::Set_MissingResidualEnergyMsg()
{
        if (ReasoningOption == QUALITATIVE_CONSERVATION)
                return;

        CConMod2Doc* pDoc = GetDocument();

        Msg_MissingResidualEnergy = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                {
                        if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                pDoc->CEnergyList_IN_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos1));

                        pDoc->CEnergyList.GetNext(pos1);
                }

                if (pDoc->CEnergyList_IN_TEMP.GetCount() >= 1)
                {
```

```
                         for (POSITION pos2 = pDoc->CEnergyList.GetHeadPosition(); pos2 
!= NULL; )
                         {
                                 if (pDoc->CEnergyList.GetAt(pos2)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                         pDoc->CEnergyList_OUT_TEMP.AddTail(pDoc-
>CEnergyList.GetAt(pos2));

                                 pDoc->CEnergyList.GetNext(pos2);
                         }

                         if (pDoc->CEnergyList_OUT_TEMP.GetCount() >= 1)
                         {
                                 // Testing if there is at least one residual energy flow 
at the output
                                 bool ResidualEnergyFound = false;

                                 for (POSITION pos3 = pDoc-
>CEnergyList_OUT_TEMP.GetHeadPosition(); pos3 != NULL; )
                                 {
                                         if (pDoc->CEnergyList_OUT_TEMP.GetAt(pos3)-
>IsResidual)
                                                 ResidualEnergyFound = true;
                                         pDoc->CEnergyList_OUT_TEMP.GetNext(pos3);
                                 }

                                 if (!ResidualEnergyFound)
                                 {
                                         Msg_MissingResidualEnergy = 
Msg_MissingResidualEnergy + 
                                                 _T("\n::Warning:: Energy Loss Not Shown 
in Function: ") + pDoc->CFunctionList.GetAt(pos)->GivenName + _T(".");
                                 }
                         }
                 }

                 EmptyAllTempLists();     // For every function block
                 pDoc->CFunctionList.GetNext(pos);
         }
}

void CConMod2View::Set_MaterialChangeWithoutEnergyMsg()
{
        CConMod2Doc* pDoc = GetDocument();

        Msg_MaterialChangeWithoutEnergy = "";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                bool ThisFuncHasInputM;
                bool ThisFuncHasOutputM;
                bool ThisFuncHasInputEBaggage;
                bool ThisFuncHasOutputEBaggage;

                ThisFuncHasInputM = false;
                ThisFuncHasOutputM = false;
                ThisFuncHasInputEBaggage = false;
                ThisFuncHasOutputEBaggage = false;
                for (POSITION pos1 = pDoc->CMaterialList.GetHeadPosition(); pos1 !=
NULL; )
                {
                        if (pDoc->CMaterialList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                        {
                                ThisFuncHasInputM = true;

                                for (POSITION pos2 = pDoc-
>CEnergyList.GetHeadPosition(); pos2 != NULL; )
                                {
                                        if ((pDoc->CEnergyList.GetAt(pos2)->pTailElem ==
pDoc->CMaterialList.GetAt(pos1)) &&
                                                (pDoc->CEnergyList.GetAt(pos2)-
>pHeadElem == pDoc->CFunctionList.GetAt(pos)))
                                        {
                                                ThisFuncHasInputEBaggage = true;
                                                //return;
                                        }
                                        pDoc->CEnergyList.GetNext(pos2);
                                }
                        }

                        if (pDoc->CMaterialList.GetAt(pos1)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                        {
                                ThisFuncHasOutputM = true;

                                for (POSITION pos2 = pDoc-
>CEnergyList.GetHeadPosition(); pos2 != NULL; )
                                {
                                        if ((pDoc->CEnergyList.GetAt(pos2)->pHeadElem ==
pDoc->CMaterialList.GetAt(pos1)) &&
                                                (pDoc->CEnergyList.GetAt(pos2)-
>pTailElem == pDoc->CFunctionList.GetAt(pos)))
                                        {
                                                ThisFuncHasOutputEBaggage = true;
                                                //return;
                                        }
                                        pDoc->CEnergyList.GetNext(pos2);
                                }
                        }

                        pDoc->CMaterialList.GetNext(pos1);
                }

                if ((ThisFuncHasInputM) && (ThisFuncHasOutputM) &&
!(ThisFuncHasInputEBaggage) && !(ThisFuncHasOutputEBaggage))
                        Msg_MaterialChangeWithoutEnergy =
Msg_MaterialChangeWithoutEnergy +
                        _T("\nEnergy must be exchanged to/from Material to transform
Material (") + pDoc->CFunctionList.GetAt(pos)->GivenName + _T(").");

                EmptyAllTempLists();     // For every function block
                pDoc->CFunctionList.GetNext(pos);
        }
}/**/

void CConMod2View::EmptyAllTempLists()
```

```
{
        CConMod2Doc* pDoc = GetDocument();

        pDoc->CMaterialList_IN_TEMP.RemoveAll();
        pDoc->CMaterialList_OUT_TEMP.RemoveAll();
        pDoc->CEnergyList_IN_TEMP.RemoveAll();
        pDoc->CEnergyList_OUT_TEMP.RemoveAll();
        pDoc->CSignalList_IN_TEMP.RemoveAll();
        pDoc->CSignalList_OUT_TEMP.RemoveAll();
}

void CConMod2View::ComposeQualitativeMessage()
{
        CConMod2Doc* pDoc = this->GetDocument();

        //==============================
        // Derivation Check
        //==============================

        Msg_OneInManyOut_M = "";
        Msg_OneInManyOut_E = "";
        Msg_ManyInOneOut_M = "";
        Msg_ManyInOneOut_E = "";
        Msg_ManyInManyOut = "";
        Msg_MissingResidualEnergy = "";
        Msg_MaterialChangeWithoutEnergy = "";
        Msg_OrphanFlow = "";
        Msg_BarrenFlow = "";

        CString* pMsg_DerivationChecks = new CString;
        *pMsg_DerivationChecks = _T("***** QUALITATIVE CONSERVATION REPORT *****\n");

        // First, clear all existing parent-child relations that are
        // leftover from a previous call to this function.
        // The relations will be recomputed during the next inferences anyways.
        for (POSITION pos = pDoc->CMaterialList.GetHeadPosition(); pos != NULL; )
        {
                pDoc->CMaterialList.GetAt(pos)->ParentList.RemoveAll();
                pDoc->CMaterialList.GetAt(pos)->ChildList.RemoveAll();
                pDoc->CMaterialList.GetNext(pos);
        }

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                pDoc->CEnergyList.GetAt(pos)->ParentList.RemoveAll();
                pDoc->CEnergyList.GetAt(pos)->ChildList.RemoveAll();
                pDoc->CEnergyList.GetNext(pos);
        }

        // Must finish drawing inferences before deciding barren and orphan flows,
        // because it is during these inferences that parent and children are
        // computed.  WIthout these inferences, all flows will return as both
        // orphan abd barren.
        Set_OneInManyOutMsg_M();
        Set_OneInManyOutMsg_E();
        Set_ManyInOneOutMsg_M();
        Set_ManyInOneOutMsg_E();
        Set_ManyInManyOutMsg();
        Set_MissingResidualEnergyMsg();
        Set_MaterialChangeWithoutEnergyMsg();

        // Now call orphan and barren flow messages
        Set_OrphanFlowMsg();
        Set_BarrenFlowMsg();

        // Now compose all the messages generated by the above checks and display
        *pMsg_DerivationChecks = *pMsg_DerivationChecks +
                Msg_OneInManyOut_M +
                Msg_OneInManyOut_E +
                Msg_ManyInOneOut_M +
                Msg_ManyInOneOut_E +
                Msg_ManyInManyOut +
                Msg_MaterialChangeWithoutEnergy +
                Msg_OrphanFlow +
                Msg_BarrenFlow;

        int* m = new int;
        *m = MessageBox(*pMsg_DerivationChecks, _T("Qualitative Conservation Report"),
MB_ICONWARNING | MB_OK);
        delete m;

        delete pMsg_DerivationChecks;            // Resets to empty string


        //==============================
                                                                                //
Irreversivbility Check

        //==============================

        CString* pMsg_IrrevChecks = new CString;
        *pMsg_IrrevChecks = _T("***** QUALITATIVE IRREVERSIBILITY REPORT *****\n");

        if (ReasoningOption >= QUALITATIVE_IRREVERSIBILITY)
        {
                *pMsg_IrrevChecks = *pMsg_IrrevChecks + Msg_MissingResidualEnergy;
                int* n = new int;
                *n = MessageBox(*pMsg_IrrevChecks, _T("Qualitative Irreversibility
Report"), MB_ICONWARNING | MB_OK);
                delete n;
        }

        delete pMsg_IrrevChecks;
}

void CConMod2View::VerifyPositivePowerOfFlows()
{
        CConMod2Doc* pDoc = this->GetDocument();

        ContinueReasoning = true;

        CString* pNegativeEnergyReportString = new CString;
        *pNegativeEnergyReportString = "***** NEGATIVE POWER REPORT *****\nThe following
flows have negative power. \n";

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                pDoc->CEnergyList.GetAt(pos)->Power = pDoc->CEnergyList.GetAt(pos)-
>UI_ForceTerm *
                        pDoc->CEnergyList.GetAt(pos)->UI_RateTerm;

                if (pDoc->CEnergyList.GetAt(pos)->Power < 0)
                {
                        ContinueReasoning = false;
                        CString* pPowerString = new CString;
                        pPowerString->Format(_T("%4.1f"), pDoc->CEnergyList.GetAt(pos)-
>Power);

                        *pNegativeEnergyReportString = *pNegativeEnergyReportString +
_T("\nFlow: ") + pDoc->CEnergyList.GetAt(pos)->GivenName +
                                _T("\t\tPower = ") + *pPowerString + _T(" W");

                        delete pPowerString;
                }

                pDoc->CEnergyList.GetNext(pos);
        }

        if (ContinueReasoning == false)
        {
                int n = MessageBox(*pNegativeEnergyReportString, _T("Negative Power
Report"), MB_ICONWARNING | MB_OK);
                AfxMessageBox(_T("Quantitative reasoning (Energy Balance, Efficiency,
Confluence) cannot continue with flows with negative power."));
        }

        delete pNegativeEnergyReportString;
}

void CConMod2View::VerifyEnergyBalanceOfFunctions()
{
        if (ContinueReasoning == false)
                return;

        CConMod2Doc* pDoc = this->GetDocument();

        CString* pEnergyBalanceReportString = new CString;
        *pEnergyBalanceReportString = "***** ENERGY BALANCE REPORT *****\n";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                double* pTotalInputPower = new double;
                double* pTotalOutputPower = new double;

                *pTotalInputPower = 0.0;
                *pTotalOutputPower = 0.0;

                for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                {
                        pDoc->CEnergyList.GetAt(pos1)->Power = pDoc-
>CEnergyList.GetAt(pos1)->UI_ForceTerm *
                                pDoc->CEnergyList.GetAt(pos1)->UI_RateTerm;

                        if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
```

```
                                *pTotalInputPower = *pTotalInputPower + (pDoc-
>CEnergyList.GetAt(pos1)->Power);

                        if (pDoc->CEnergyList.GetAt(pos1)->pTailElem == pDoc-
>CFunctionList.GetAt(pos))
                                *pTotalOutputPower = *pTotalOutputPower + (pDoc-
>CEnergyList.GetAt(pos1)->Power);

                        pDoc->CEnergyList.GetNext(pos1);
                }

                if (*pTotalInputPower == *pTotalOutputPower)
                        *pEnergyBalanceReportString = *pEnergyBalanceReportString +
_T("\nFunction: ") + pDoc->CFunctionList.GetAt(pos)->GivenName + _T("\tBalanced.");

                else
                {
                        ContinueReasoning = false;

                        CString* pInputPString = new CString;
                        CString* pOutputPString = new CString;

                        pInputPString->Format(_T("%4.1f"), *pTotalInputPower);
                        pOutputPString->Format(_T("%4.1f"), *pTotalOutputPower);

                        *pEnergyBalanceReportString = *pEnergyBalanceReportString +
_T("\nFunction: ") +
                                pDoc->CFunctionList.GetAt(pos)->GivenName + _T("\tInput
= ") + *pInputPString +
                                _T(" W\tOutput = ") + *pOutputPString + _T(" W.");

                        delete pInputPString;
                        delete pOutputPString;
                }

                delete pTotalInputPower;
                delete pTotalOutputPower;

                pDoc->CFunctionList.GetNext(pos);
        }

        int n = MessageBox(*pEnergyBalanceReportString, _T("Energy Balance Violation
Report"), MB_ICONWARNING | MB_OK);

        if (ContinueReasoning == false)
                AfxMessageBox(_T("Quantitative reasoning (Efficiency, Confluence) cannot
continue without energy balance in each function."));

        delete pEnergyBalanceReportString;
}

void CConMod2View::ComputeEfficiency()
{
        if (ContinueReasoning == false)
                return;

        CConMod2Doc* pDoc = this->GetDocument();

        //=====================================
        // Compute function-wise efficiency
        //=====================================

        CString* pEfficiencyMessage = new CString;
        *pEfficiencyMessage = "***** INDIVIDUAL FUNCTION EFFICIENCY REPORT *****\n"
                "\nFunction\tInput\tUsable\tLoss\tEfficiency"
                "\n==========================================";

        for (POSITION pos = pDoc->CFunctionList.GetHeadPosition(); pos != NULL; )
        {
                pDoc->CFunctionList.GetAt(pos)->Efficiency = 0.0;        // Reset the
efficiency at the beginning of

                                                 // each run of this algorithm

                double* pTotalInputPower = new double;
                double* pTotalUsableOutputPower = new double;

                *pTotalInputPower = 0.0;
                *pTotalUsableOutputPower = 0.0;

                for (POSITION pos1 = pDoc->CEnergyList.GetHeadPosition(); pos1 != NULL;
)
                {
                        pDoc->CEnergyList.GetAt(pos1)->Power = pDoc-
>CEnergyList.GetAt(pos1)->UI_ForceTerm *
                                pDoc->CEnergyList.GetAt(pos1)->UI_RateTerm;

                        if (pDoc->CEnergyList.GetAt(pos1)->pHeadElem == pDoc-
>CFunctionList.GetAt(pos))
                                *pTotalInputPower = *pTotalInputPower + (pDoc-
>CEnergyList.GetAt(pos1)->Power);

                        if ((pDoc->CEnergyList.GetAt(pos1)->pTailElem == pDoc-
>CFunctionList.GetAt(pos)) &&
                                (pDoc->CEnergyList.GetAt(pos1)->IsResidual == false))
                                *pTotalUsableOutputPower = *pTotalUsableOutputPower +
(pDoc->CEnergyList.GetAt(pos1)->Power);

                        pDoc->CEnergyList.GetNext(pos1);
                }

                if ((*pTotalInputPower != 0.0) && (*pTotalUsableOutputPower != 0))
                        pDoc->CFunctionList.GetAt(pos)->Efficiency =
(*pTotalUsableOutputPower / *pTotalInputPower);

                CString* pInputEString = new CString;
                CString* pUsableOutputEString = new CString;
                CString* pLossEString = new CString;
                CString* pEffyString = new CString;

                pInputEString->Format(_T("%5.1f"), *pTotalInputPower);
                pUsableOutputEString->Format(_T("%5.1f"), *pTotalUsableOutputPower);
                pLossEString->Format(_T("%5.1f"), (*pTotalInputPower -
*pTotalUsableOutputPower));
                pEffyString->Format(_T("%5.3f"), pDoc->CFunctionList.GetAt(pos)-
>Efficiency);

                *pEfficiencyMessage = *pEfficiencyMessage +
                        "\n" + pDoc->CFunctionList.GetAt(pos)->GivenName +
                        "\t" + *pInputEString +
                        "\t" + *pUsableOutputEString +
                        "\t" + *pLossEString +
                        "\t" + *pEffyString;

                delete pTotalInputPower;
                delete pTotalUsableOutputPower;
                delete pInputEString;
                delete pUsableOutputEString;
                delete pLossEString;
                delete pEffyString;

                pDoc->CFunctionList.GetNext(pos);
        }

        //======================================
        // Compute efficiency for the whole model
        //======================================

        double* pModelInputPower = new double;
        double* pModelLossPower = new double;
        double* pModelEfficiency = new double;

        *pModelInputPower = 0;
        *pModelLossPower = 0;
        *pModelEfficiency = 0;

        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                if ((ElementIsFunction(pDoc->CEnergyList.GetAt(pos)->pHeadElem)) &&
                        (pDoc->CEnergyList.GetAt(pos)->pTailElem != NULL) &&
                        ((ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pTailElem)) ||
(ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pTailElem->pTailElem))))
                        *pModelInputPower = *pModelInputPower + pDoc-
>CEnergyList.GetAt(pos)->Power;

                if ((ElementIsFunction(pDoc->CEnergyList.GetAt(pos)->pTailElem)) &&
                        (pDoc->CEnergyList.GetAt(pos)->pHeadElem != NULL) &&
                        ((ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pHeadElem)) ||
(ElementIsEnv(pDoc->CEnergyList.GetAt(pos)->pHeadElem->pHeadElem))) &&
                        (pDoc->CEnergyList.GetAt(pos)->IsResidual == true))
                        *pModelLossPower = *pModelLossPower + pDoc-
>CEnergyList.GetAt(pos)->Power;

                pDoc->CEnergyList.GetNext(pos);
        }

        if ((*pModelInputPower != 0) /*&& (*pModelLossPower != 0)*/)      // Have to set
more traps
                *pModelEfficiency = (*pModelInputPower - *pModelLossPower) /
*pModelInputPower;

        CString* pModelEffyString = new CString;
        pModelEffyString->Format(_T("%5.3f"), *pModelEfficiency);

        *pEfficiencyMessage = *pEfficiencyMessage + "\n\nOVERAL MODEL EFFICIENCY: " +
*pModelEffyString;
```

```
        delete pModelInputPower;
        delete pModelLossPower;
        delete pModelEfficiency;
        delete pModelEffyString;

        int n = MessageBox(*pEfficiencyMessage, _T("Efficiency Report"), MB_ICONWARNING
| MB_OK);

        delete pEfficiencyMessage;
}

void CConMod2View::ComposeQuantitativeMessage()
{
        //================================
        // Commented out for rolling back to Layer One (Chapter 6)
        //================================

        if ((ReasoningOption == QUALITATIVE_CONSERVATION) || (ReasoningOption ==
QUALITATIVE_IRREVERSIBILITY))
        {
                AfxMessageBox(_T("***** QUANTITATIVE REASONING NOT AVAILABLE *****\n\nTo
turn on, choose \"Quantitative -> Efficiency\" from Reasoning Menu."));
                return;
        }

        if (ReasoningOption == QUANTITATIVE_EFFICIENCY)
        {
                VerifyPositivePowerOfFlows();
                VerifyEnergyBalanceOfFunctions();
                ComputeEfficiency();     // Resets every function's effy to zero, then
recomputes
                                                                // from present state
of model
        }

        if (ReasoningOption == QUANTITATIVE_POWERREQUIRED)
        {
                AfxMessageBox(_T("Under Construction."));
        }
}

void CConMod2View::ComposeCausalMessage()
{
        AfxMessageBox(_T("Reasoning on Causal Behaviour"));
        CConMod2Doc* pDoc = this->GetDocument();
        for (POSITION pos = pDoc->CEnergyList.GetHeadPosition(); pos != NULL; )
        {
                if (pDoc->ActuateE_Template_List.GetAt(pos)->EisActuated == false)
                {
                        for (POSITION pos2 = pDoc->CFunctionList.GetHeadPosition(); pos2
!= NULL; )
                        {
                                if (pDoc->ActuateE_Template_List.GetAt(pos)-
>pEnergy_OutE->pTailElem == pDoc->CFunctionList.GetAt(pos2))
                                {
                                        pDoc->CFunctionList.GetAt(pos2)->ElementIsHidden
= true;
                                }
                                if (pDoc->CFunctionList.GetAt(pos2)->ElementIsHidden ==
true)
                                {
                                        for (POSITION pos3 = pDoc-
>CEnergyList.GetHeadPosition(); pos3 != NULL; )
                                        {
                                                if (pDoc->CEnergyList.GetAt(pos3) ==
pDoc->CFunctionList.GetAt(pos2)->pTailElem)
                                                {
                                                        pDoc-
>CFunctionList.GetAt(pos2)->ElementIsHidden = true;
                                                }
                                                pDoc->CEnergyList.GetNext(pos3);
                                        }
                                }
                                pDoc->CFunctionList.GetNext(pos2);
                        }
                }
                pDoc->ActuateE_Template_List.GetNext(pos);
        }
}
#include "stdafx.h"
#include "ConMod2.h"
#include "Convert_E_Template.h"
#include "afxdialogex.h"

IMPLEMENT_DYNAMIC(CConvert_E_Template, CDialog)

CConvert_E_Template::CConvert_E_Template(CWnd* pParent /*= NULL*/, CPoint InsertionPoint
/*= (500,500)*/,
        CString* pCounterString_F /*= NULL*/, CString* pCounterString_InE /*= NULL*/,
        CString* pCounterString_OutE /*= NULL*/, CString* pCounterString_OutE_Res /*=
NULL*/,
        int ReasOpt)
        : CDialog(CConvert_E_Template::IDD, pParent)
        , ReasoningOption(ReasOpt)
{
        pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);

        CPoint TailOfInE(InsertionPoint.x - TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
        CPoint HeadOfOutE(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
        CPoint HeadOfOutE_Res(InsertionPoint.x, InsertionPoint.y +
TEMPLATE_FLOW_LENGTH);

        pEnergy_InE = new CEnergy(NULL, TailOfInE, InsertionPoint, pCounterString_InE,
this->ReasoningOption);
        pEnergy_OutE = new CEnergy(NULL, InsertionPoint, HeadOfOutE,
pCounterString_OutE, this->ReasoningOption);
        pEnergy_OutE_Res = new CEnergy(NULL, InsertionPoint, HeadOfOutE_Res,
pCounterString_OutE_Res, this->ReasoningOption);

        pEnergy_InE->pHeadElem = pFunctionBlock;
        pEnergy_OutE->pTailElem = pFunctionBlock;
        pEnergy_OutE_Res->pTailElem = pFunctionBlock;
        pEnergy_OutE_Res->UI_IsResidual = true;
}
CConvert_E_Template::~CConvert_E_Template()
{
}
```

```
void CConvert_E_Template::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CConvert_E_Template, CDialog)
END_MESSAGE_MAP()


// Convert_E_Template message handlers
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "DeEn_M_Template.h"
#include "afxdialogex.h"

IMPLEMENT_DYNAMIC(CDeEn_M_Template, CDialog)

CDeEn_M_Template::CDeEn_M_Template(CWnd* pParent/* = NULL*/,
	CPoint InsertionPoint /*= (500,500)*/,
	CString* pCounterString_F /*= NULL*/,
	CString* pCounterString_InM /*= NULL*/,
	CString* pCounterString_OutM /*= NULL*/,
	CString* pCounterString_InE /*= NULL*/,
	CString* pCounterString_OutE /*= NULL*/,
	int ReasOpt)
	: CDialog(CDeEn_M_Template::IDD, pParent)
	, ReasoningOption(ReasOpt)
{
	pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);

	CPoint TailOfInM(InsertionPoint.x - 1.5*TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
	CPoint HeadOfOutM(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
	CPoint HeadOfOutE(InsertionPoint.x, InsertionPoint.y - TEMPLATE_FLOW_LENGTH);

	pMaterial_InM = new CMaterial(NULL, TailOfInM, InsertionPoint,
pCounterString_InM, this->ReasoningOption);
	pMaterial_OutM = new CMaterial(NULL, InsertionPoint, HeadOfOutM,
pCounterString_OutM, this->ReasoningOption);
	pEnergy_InE = new CEnergy(NULL, InsertionPoint /*Dummy*/, InsertionPoint,
pCounterString_InE, this->ReasoningOption);
	pEnergy_OutE = new CEnergy(NULL, InsertionPoint, HeadOfOutE,
pCounterString_OutE, this->ReasoningOption);

	pMaterial_InM->pHeadElem = pFunctionBlock;
	pMaterial_OutM->pTailElem = pFunctionBlock;
	pEnergy_InE->pHeadElem = pFunctionBlock;
	pEnergy_InE->pTailElem = pMaterial_InM;
	pEnergy_OutE->pTailElem = pFunctionBlock;
}

CDeEn_M_Template::~CDeEn_M_Template()
{
}

void CDeEn_M_Template::DoDataExchange(CDataExchange* pDX)
```

```
{
        CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CDeEn_M_Template, CDialog)
END_MESSAGE_MAP()


// DeEn_M_Template message handlers
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Distribute_E_Template.h"
#include "afxdialogex.h"

IMPLEMENT_DYNAMIC(CDistribute_E_Template, CDialog)

CDistribute_E_Template::CDistribute_E_Template(CWnd* pParent /*= NULL*/,
        CPoint InsertionPoint /*= (500,500)*/,
        CString* pCounterString_F /*= NULL*/,
        CString* pCounterString_InE /*= NULL*/,
        CString* pCounterString_OutE1 /*= NULL*/,
        CString* pCounterString_OutE2 /*= NULL*/,
        int ReasOpt)
        : CDialog(CDistribute_E_Template::IDD, pParent)
        , ReasoningOption(ReasOpt)
{
        pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);

        CPoint TailOfInE(InsertionPoint.x - TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
        CPoint HeadOfOutE1(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y -
TEMPLATE_FLOW_LENGTH);
        CPoint HeadOfOutE2(InsertionPoint.x + TEMPLATE_FLOW_LENGTH, InsertionPoint.y +
TEMPLATE_FLOW_LENGTH);

        pEnergy_InE = new CEnergy(NULL, TailOfInE, InsertionPoint, pCounterString_InE,
this->ReasoningOption);
        pEnergy_OutE1 = new CEnergy(NULL, InsertionPoint, HeadOfOutE1,
pCounterString_OutE1, this->ReasoningOption);
        pEnergy_OutE2 = new CEnergy(NULL, InsertionPoint, HeadOfOutE2,
pCounterString_OutE2, this->ReasoningOption);

        pEnergy_InE->pHeadElem = pFunctionBlock;
        pEnergy_OutE1->pTailElem = pFunctionBlock;
        pEnergy_OutE2->pTailElem = pFunctionBlock;
}

CDistribute_E_Template::~CDistribute_E_Template()
{
}
void CDistribute_E_Template::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CDistribute_E_Template, CDialog)
END_MESSAGE_MAP()
```

```
// Distribute_E_Template message handlers
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Edge.h"
#include "math.h"

CEdge::CEdge(void)
{
}

CEdge::CEdge(CPoint TailClick, CPoint HeadClick)
{
        TailPoint = TailClick;
        HeadPoint = HeadClick;
        GeometricCenter = *InterpolatePoints(TailPoint, HeadPoint, 0.5);
        StemThickness = THIN;
        StemLineFont = PS_SOLID;

        HeadSize = EDGE_HEAD_SIZE;
        HalfHeadAngle = EDGE_HEAD_HALF_ANGLE;

        ComputeAnchorPoints();
        pHeadElem = NULL;
        pTailElem = NULL;
        ThisFlowIsIncomingBaggage = false;
        ThisFlowIsOutgoingBaggage = false;

        FontSize = GENERIC_FONT_SIZE;
}

CEdge::~CEdge(void)
{
}

void CEdge::AttachEdgeToNearestAnchor()
{
        FontSize = GENERIC_FONT_SIZE;
        HeadSize = EDGE_HEAD_SIZE;

        if (pTailElem != NULL)
        {
                // Initialize with any one anchor point - zeroth chosen arbitrarily
                long double d = distance(HeadPoint, pTailElem->Anchors[0]);
                TailPoint = pTailElem->Anchors[0];

                for (int n = 1; n <= 15; n++)    // Hard coded - 16 anchor points on
both nodes and edges
                {
                        if (distance(pTailElem->Anchors[n], HeadPoint) < d)
                        {
                                d = distance(HeadPoint, pTailElem->Anchors[n]);
                                TailPoint = pTailElem->Anchors[n];
                        }
                }
        }
        if (pHeadElem != NULL)
        {
                // Initialize with any one anchor point - zeroth chosen arbitrarily
                long double d = distance(TailPoint, pHeadElem->Anchors[0]);
                this->HeadPoint = pHeadElem->Anchors[0];

                for (int n = 1; n <= 15; n++)    // Hard coded - 16 anchor points on
both nodes and edges
                {
                        if (distance(pHeadElem->Anchors[n], TailPoint) < d)
                        {
                                d = distance(TailPoint, pHeadElem->Anchors[n]);
                                HeadPoint = pHeadElem->Anchors[n];
                        }
                }
        }

        if (ThisFlowIsOutgoingBaggage)   // i.e., this flow's is a baggage flow and its
pHeadElem (carrier)
                                                                        // is another
flow that is exiting the same function as this one
        {
                for (int n = 0; n <= 15; n++)
                {
                        if (this->pTailElem->AnchorsForBaggageFlows[n] == this-
>pHeadElem->TailPoint)
                                this->TailPoint = this->pTailElem-
>AnchorsForBaggageFlows[n + 2];
                        // The following two conditional statements are special cases
where
                        // the 14+2 = 16 and 15+2 = 17 -th elements do not exist in the
array.
                        if (this->pTailElem->AnchorsForBaggageFlows[14] == this-
>pHeadElem->TailPoint)
                                this->TailPoint = this->pTailElem-
>AnchorsForBaggageFlows[0];
                        if (this->pTailElem->AnchorsForBaggageFlows[15] == this-
>pHeadElem->TailPoint)
                                this->TailPoint = this->pTailElem-
>AnchorsForBaggageFlows[1];
                        if (this->pTailElem->AnchorsForBaggageFlows[0] == this-
>pHeadElem->TailPoint)
                                this->TailPoint = this->pTailElem-
>AnchorsForBaggageFlows[2];

                        this->HeadPoint = this->pHeadElem->Anchors[5];
                }
                // The following two lines improves readability of baggage flows
                FontSize = BAGGAGE_FONT_SIZE;
                HeadSize = EDGE_HEAD_SIZE / 2;
        }

        if (ThisFlowIsIncomingBaggage)
        {
                for (int n = 0; n <= 15; n++)
                {
                        if (this->pHeadElem->AnchorsForBaggageFlows[n] == this-
>pTailElem->HeadPoint)
```

```
                                    this->HeadPoint = this->pHeadElem-
>AnchorsForBaggageFlows[n + 2];
                            // The following two conditional statements are special cases
where
                            // the 14+2 = 16 and 15+2 = 17 -th elements do not exist in the
array.
                            if (this->pHeadElem->AnchorsForBaggageFlows[14] == this-
>pTailElem->HeadPoint)
                                    this->HeadPoint = this->pHeadElem-
>AnchorsForBaggageFlows[0];
                            if (this->pHeadElem->AnchorsForBaggageFlows[15] == this-
>pTailElem->HeadPoint)
                                    this->HeadPoint = this->pHeadElem-
>AnchorsForBaggageFlows[1];
                            if (this->pHeadElem->AnchorsForBaggageFlows[0] == this-
>pTailElem->HeadPoint)
                                    this->HeadPoint = this->pHeadElem-
>AnchorsForBaggageFlows[2];

                            this->TailPoint = this->pTailElem->Anchors[10];
                    }
                    // The following two lines improves readability of baggage flows
                    FontSize = BAGGAGE_FONT_SIZE;
                    HeadSize = EDGE_HEAD_SIZE / 2;
        }
}

void CEdge::DrawOnDC(CDC* pDC)
{
//      if (this->IsHidden == false)
        {

        AttachEdgeToNearestAnchor();

        CElement::DrawOnDC(pDC); // Call the drawing function of the parent class - sets
pen color

                                                                  //
============================================================================
                                                                  // Draw the STEM of the
arrow using PenStem (No brush required)
                                                                  //
============================================================================
                CPen PenStem;
                PenStem.CreatePen(StemLineFont, StemThickness, RGB(PenR, PenG, PenB));
                CPen* pOldPen = pDC->SelectObject(&PenStem);
                CPoint* ArrowTerminalPoints = new CPoint[2];
                ArrowTerminalPoints[0] = TailPoint;
                ArrowTerminalPoints[1] = HeadPoint;
                pDC->Polyline(ArrowTerminalPoints, 2);
                delete[] ArrowTerminalPoints;

                ComputeAnchorPoints();          // Computes the eight anchor points
along the stem of
                                                                          // the edge
whenever the edge is edited, moved, or whatever.

                ResetGeometricCenter();         // Makes sure that the GeometricCenter
is reset between the
```

```
                                                                          // Tail and
Head points, when an arrow is moved by grabbing
                                                                          // Those
terminal points

                                                                          /*for (int
AnchorInx = 1; AnchorInx <= 16; AnchorInx++)
                                                                          pDC-
>Ellipse(Anchors[AnchorInx - 1].x - 1, Anchors[AnchorInx - 1].y - 1,

        Anchors[AnchorInx - 1].x + 1, Anchors[AnchorInx - 1].y + 1);*/

                                                                          //
============================================================================
                                                                          // Draw the
HEAD of the arrow using PenHead and BrushHead
                                                                          //
============================================================================

                CPen PenHead;
                PenHead.CreatePen(PS_SOLID, THIN, RGB(PenR, PenG, PenB));

                if (this->pHeadElem == NULL)
                {
                        HeadBrushR = DANGLING_BRUSH_R;
                        HeadBrushG = DANGLING_BRUSH_G;
                        HeadBrushB = DANGLING_BRUSH_B;
                }
                else
                {
                        HeadBrushR = GENERIC_BRUSH_R;
                        HeadBrushG = GENERIC_BRUSH_G;
                        HeadBrushB = GENERIC_BRUSH_B;
                }
                if (this->IsHidden == true)
                {
                        HeadBrushR = HIDDEN_PEN_R;
                        HeadBrushG = HIDDEN_PEN_G;
                        HeadBrushB = HIDDEN_PEN_B;
                }

                CBrush BrushHead(RGB(HeadBrushR, HeadBrushG, HeadBrushB));
                CBrush* pOldBrush = pDC->SelectObject(&BrushHead);

                pOldPen = pDC->SelectObject(&PenHead);

                double alpha = atan(fabs(double(HeadPoint.y) - double(TailPoint.y)) /
fabs(double(HeadPoint.x) - double(TailPoint.x)));

                int X_Factor, Y_Factor;

                if (HeadPoint.x >= TailPoint.x)
                        X_Factor = 1;
                else X_Factor = (-1);

                if (TailPoint.y >= HeadPoint.y)
                        Y_Factor = 1;
                else Y_Factor = (-1);
```

```
		HeadLeftVertex.x = HeadPoint.x - HeadSize * cos(alpha - HalfHeadAngle) *
X_Factor;
		HeadLeftVertex.y = HeadPoint.y + HeadSize * sin(alpha - HalfHeadAngle) *
Y_Factor;

		HeadRightVertex.x = HeadPoint.x - HeadSize * cos(alpha + HalfHeadAngle)
* X_Factor;
		HeadRightVertex.y = HeadPoint.y + HeadSize * sin(alpha + HalfHeadAngle)
* Y_Factor;

		HeadVertexArray[0] = HeadPoint;
		HeadVertexArray[1] = HeadLeftVertex;
		HeadVertexArray[2] = HeadRightVertex;

		pDC->Polygon(HeadVertexArray, 3);

		//
============================================================================
		// Draw the TAIL of the arrow using PenTail and BrushTail
		//
============================================================================

		CPen PenTail;
		PenTail.CreatePen(PS_SOLID, THIN, RGB(PenR, PenG, PenB));

		if (this->pTailElem == NULL)
		{
			TailBrushR = DANGLING_BRUSH_R;
			TailBrushG = DANGLING_BRUSH_G;
			TailBrushB = DANGLING_BRUSH_B;
		}
		else
		{
			TailBrushR = GENERIC_BRUSH_R;
			TailBrushG = GENERIC_BRUSH_G;
			TailBrushB = GENERIC_BRUSH_B;
		}

		if(this->IsHidden==true)
		{
			TailBrushR = HIDDEN_PEN_R;
			TailBrushG = HIDDEN_PEN_G;
			TailBrushB = HIDDEN_PEN_B;
		}

		CBrush BrushTail(RGB(TailBrushR, TailBrushG, TailBrushB));
		pOldBrush = pDC->SelectObject(&BrushTail);
		pOldPen = pDC->SelectObject(&PenTail);

		pDC->Ellipse(TailPoint.x - 4, TailPoint.y - 4, TailPoint.x + 4,
TailPoint.y + 4);

		//
============================================================================
		// Put back the old objects, although I do not understand how this
impacts anything.
		//
============================================================================
                pDC->SelectObject(pOldPen);
                pDC->SelectObject(pOldBrush);
        }
}

void CEdge::ComputeAnchorPoints()
{
        // First eight poitns - between tail and center
        for (int AnchorInx = 1; AnchorInx <= 16; AnchorInx++)
        {
                Anchors[AnchorInx - 1] = *InterpolatePoints(TailPoint, HeadPoint, (0.5 /
9 * AnchorInx));
        }

        // Second eight popints - between center and head
        for (int AnchorInx = 1; AnchorInx <= 8; AnchorInx++)
        {
                Anchors[AnchorInx + 7] = *InterpolatePoints(TailPoint, HeadPoint, (0.5 +
0.5 / 9 * AnchorInx));
        }
}

void CEdge::ResetGeometricCenter()
{
        GeometricCenter = *this->InterpolatePoints(this->HeadPoint, this->TailPoint,
0.5);
}
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Element.h"

CElement::CElement(void)
{
        IsHighlighted = false;
        IsSelected = false;
        IsResidual = false;
        IsHidden = false;

        PenR = GENERIC_PEN_R;
        PenG = GENERIC_PEN_G;
        PenB = GENERIC_PEN_B;

        GrabHandle = 0; // NONE
}

CElement::~CElement(void)
{
}

void CElement::DrawOnDC(CDC* pDC)
{
        // ===========================================================================
        // Decide the color, based on HIGHLIGHT, SELECTED, or GENERIC status
        // ===========================================================================

        if (this->IsHighlighted) // This ORDER of checks is very important.  If
        {                                                        // changed, this will
change the highlight and
```

```
                this->PenR = PRESELECTION_PEN_R; // unhighlight behavior of energies
                this->PenG = PRESELECTION_PEN_G;
                this->PenB = PRESELECTION_PEN_B;
        }

        else if (this->IsSelected)
        {
                this->PenR = SELECTION_PEN_R;
                this->PenG = SELECTION_PEN_G;
                this->PenB = SELECTION_PEN_B;
        }

        else if (this->IsResidual)
        {
                this->PenR = RESIDUAL_PEN_R;
                this->PenG = RESIDUAL_PEN_G;
                this->PenB = RESIDUAL_PEN_B;
        }
        else if (this->IsHidden)
        {
                this->PenR = HIDDEN_PEN_R;
                this->PenG = HIDDEN_PEN_G;
                this->PenB = HIDDEN_PEN_B;
        }

        else
        {
                this->PenR = GENERIC_PEN_R;
                this->PenG = GENERIC_PEN_G;
                this->PenB = GENERIC_PEN_B;
        }
}
```

```
// Energize_M_Template.cpp : implementation file
//

#include "stdafx.h"
#include "ConMod2.h"
#include "Energize_M_Template.h"
#include "afxdialogex.h"


// Energize_M_Template dialog

IMPLEMENT_DYNAMIC(CEnergize_M_Template, CDialog)

CEnergize_M_Template::CEnergize_M_Template(CWnd* pParent/* = NULL*/,
        CPoint InsertionPoint /*= (500,500)*/,
        CString* pCounterString_F /*= NULL*/,
        CString* pCounterString_InM /*= NULL*/,
        CString* pCounterString_OutM /*= NULL*/,
        CString* pCounterString_InE /*= NULL*/,
        CString* pCounterString_OutE /*= NULL*/,
        int ReasOpt)
        : CDialog(CEnergize_M_Template::IDD, pParent)
        , ReasoningOption(ReasOpt)
{
        pFunctionBlock = new CFunction(NULL, InsertionPoint, pCounterString_F);
```

```
	CPoint TailOfInM(InsertionPoint.x - TEMPLATE_FLOW_LENGTH, InsertionPoint.y);
	CPoint HeadOfOutM(InsertionPoint.x + 1.5*TEMPLATE_FLOW_LENGTH,
InsertionPoint.y);
	CPoint TailOfInE(InsertionPoint.x, InsertionPoint.y - TEMPLATE_FLOW_LENGTH);

	pMaterial_InM = new CMaterial(NULL, TailOfInM, InsertionPoint,
pCounterString_InM, this->ReasoningOption);
	pMaterial_OutM = new CMaterial(NULL, InsertionPoint, HeadOfOutM,
pCounterString_OutM, this->ReasoningOption);
	pEnergy_InE = new CEnergy(NULL, TailOfInE, InsertionPoint, pCounterString_InE,
this->ReasoningOption);
	pEnergy_OutE = new CEnergy(NULL, InsertionPoint, HeadOfOutM /*Dummy*/,
pCounterString_OutE, this->ReasoningOption);

	pMaterial_InM->pHeadElem = pFunctionBlock;
	pMaterial_OutM->pTailElem = pFunctionBlock;
	pEnergy_InE->pHeadElem = pFunctionBlock;
	pEnergy_OutE->pTailElem = pFunctionBlock;
	pEnergy_OutE->pHeadElem = pMaterial_OutM;
	if (pEnergy_InE->IsHidden == true)
	{
		pEnergy_OutE->IsHidden = true;
		pMaterial_OutM->IsHidden = true;
	}

}

CEnergize_M_Template::~CEnergize_M_Template()
{
}

void CEnergize_M_Template::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
}


BEGIN_MESSAGE_MAP(CEnergize_M_Template, CDialog)
END_MESSAGE_MAP()


// Energize_M_Template message handlers
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Env.h"
#include "afxdialogex.h"
#include "geometry.h"

// CEnv dialog
IMPLEMENT_DYNAMIC(CEnv, CDialog)

CEnv::CEnv(CWnd* pParent, CPoint InsertionPoint, CString* pCounterString)
	: CDialog(CEnv::IDD, pParent)
	, GivenName(_T("Env") + *pCounterString)
{
	GeometricCenter = InsertionPoint;
	ComputeBlockCoordinates();
```

```
        DoModal();              // Launches modal dialog
}

CEnv::~CEnv()
{
}

void CEnv::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        DDX_Text(pDX, IDC_ENV_NAME, GivenName);
}

void CEnv::ComputeBlockCoordinates()
{
        Anchors[0] = CPoint((GeometricCenter.x + ENV_SIZE), (GeometricCenter.y));
        Anchors[1] = CPoint((GeometricCenter.x), (GeometricCenter.y + ENV_SIZE *
0.866));
        Anchors[2] = CPoint((GeometricCenter.x - ENV_SIZE), (GeometricCenter.y));
        Anchors[3] = CPoint((GeometricCenter.x), (GeometricCenter.y - ENV_SIZE *
0.866));
        Anchors[4] = CPoint((GeometricCenter.x + 0.5*ENV_SIZE), (GeometricCenter.y +
ENV_SIZE * 0.866));
        Anchors[5] = CPoint((GeometricCenter.x - 0.5*ENV_SIZE), (GeometricCenter.y +
ENV_SIZE * 0.866));
        Anchors[6] = CPoint((GeometricCenter.x - 0.5*ENV_SIZE), (GeometricCenter.y -
ENV_SIZE * 0.866));
        Anchors[7] = CPoint((GeometricCenter.x + 0.5*ENV_SIZE), (GeometricCenter.y -
ENV_SIZE * 0.866));

        Anchors[8] = CPoint((GeometricCenter.x + 0.75*ENV_SIZE), (GeometricCenter.y +
ENV_SIZE * 0.433));
        Anchors[9] = CPoint((GeometricCenter.x - 0.75*ENV_SIZE), (GeometricCenter.y +
ENV_SIZE * 0.433));
        Anchors[10] = CPoint((GeometricCenter.x - 0.75*ENV_SIZE), (GeometricCenter.y -
ENV_SIZE * 0.433));
        Anchors[11] = CPoint((GeometricCenter.x + 0.75*ENV_SIZE), (GeometricCenter.y -
ENV_SIZE * 0.433));
        Anchors[12] = Anchors[0];
        Anchors[13] = Anchors[0];
        Anchors[14] = Anchors[0];
        Anchors[15] = Anchors[0];

        AnchorsForBaggageFlows[0] = Anchors[0];
        AnchorsForBaggageFlows[1] = Anchors[11];
        AnchorsForBaggageFlows[2] = Anchors[7];
        AnchorsForBaggageFlows[3] = Anchors[3];
        AnchorsForBaggageFlows[4] = Anchors[6];
        AnchorsForBaggageFlows[5] = Anchors[10];
        AnchorsForBaggageFlows[6] = Anchors[2];
        AnchorsForBaggageFlows[7] = Anchors[9];
        AnchorsForBaggageFlows[8] = Anchors[5];
        AnchorsForBaggageFlows[9] = Anchors[1];
        AnchorsForBaggageFlows[10] = Anchors[4];
        AnchorsForBaggageFlows[11] = Anchors[8];
        AnchorsForBaggageFlows[12] = Anchors[0];
        AnchorsForBaggageFlows[13] = Anchors[0];
        AnchorsForBaggageFlows[14] = Anchors[0];
        AnchorsForBaggageFlows[15] = Anchors[0];
```

```
}

void CEnv::DrawOnDC(CDC* pDC)
{
	CElement::DrawOnDC(pDC); // Call the drawing function of the parent class - sets 
pen color

	if (this->NoInputAttached && this->NoOutputAttached)
	{
		BrushR = DANGLING_BRUSH_R;
		BrushG = DANGLING_BRUSH_G;
		BrushB = DANGLING_BRUSH_B;
	}
	else
	{
		BrushR = ENV_BRUSH_R;
		BrushG = ENV_BRUSH_G;
		BrushB = ENV_BRUSH_B;
	}

	CPen Pen;
	Pen.CreatePen(PS_SOLID, 2, RGB(PenR, PenG, PenB));
	CPen* pOldPen = pDC->SelectObject(&Pen);
	CBrush Brush(RGB(BrushR, BrushG, BrushB));
	CBrush* pOldBrush = pDC->SelectObject(&Brush);

	ComputeBlockCoordinates();

	CPoint AnchorsForHexagon[6];
	AnchorsForHexagon[0] = Anchors[0];
	AnchorsForHexagon[1] = Anchors[4];
	AnchorsForHexagon[2] = Anchors[5];
	AnchorsForHexagon[3] = Anchors[2];
	AnchorsForHexagon[4] = Anchors[6];
	AnchorsForHexagon[5] = Anchors[7];
	pDC->Polygon(AnchorsForHexagon, 6);

	//pDC->Ellipse(GeometricCenter.x - 2, GeometricCenter.y - 2,
	//GeometricCenter.x + 2, GeometricCenter.y + 2);
	/*
	for (int AnchorInx = 1; AnchorInx <= 16; AnchorInx++)
	pDC->Ellipse(Anchors[AnchorInx - 1].x - 1, Anchors[AnchorInx - 1].y - 1,
	Anchors[AnchorInx - 1].x + 1, Anchors[AnchorInx - 1].y + 1);*/

	// Initializes a CFont object with the specified characteristics.
	CFont font;
	VERIFY(font.CreateFont(
		16,                        // nHeight
		0,                         // nWidth
		0,                         // nEscapement
		0,                         // nOrientation
		FW_NORMAL,                 // nWeight
		FALSE,                     // bItalic
		FALSE,                     // bUnderline
		0,                         // cStrikeOut
		ANSI_CHARSET,              // nCharSet
		OUT_DEFAULT_PRECIS,        // nOutPrecision
		CLIP_DEFAULT_PRECIS,       // nClipPrecision
		DEFAULT_QUALITY,           // nQuality
```

```
                 DEFAULT_PITCH | FF_SWISS,  // nPitchAndFamily
                 _T("Arial")));                 // lpszFacename

        CFont* def_font = pDC->SelectObject(&font);
        pDC->SetTextAlign(TA_CENTER | TA_BASELINE);
        pDC->TextOut(GeometricCenter.x, GeometricCenter.y, GivenName);
        pDC->SelectObject(def_font);

        font.DeleteObject();
        pDC->SelectObject(pOldPen);
        pDC->SelectObject(pOldBrush);
}

BEGIN_MESSAGE_MAP(CEnv, CDialog)
END_MESSAGE_MAP()


// CEnv message handlers


void CEnv::OnEnChangeEdit1()
{
        // TODO:  If this is a RICHEDIT control, the control will not
        // send this notification unless you override the __super::OnInitDialog()
        // function and call CRichEditCtrl().SetEventMask()
        // with the ENM_CHANGE flag ORed into the mask.

        // TODO:  Add your control notification handler code here
}
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Function.h"
#include "afxdialogex.h"


// CFunction dialog

IMPLEMENT_DYNAMIC(CFunction, CDialog)

CFunction::CFunction(CWnd* pParent, CPoint InsertionPoint, CString* pCounterString)
        : CDialog(CFunction::IDD, pParent)
        , GivenName(_T("F") + *pCounterString)           // Populates the string in the
GivenName field - default
{
        GeometricCenter = InsertionPoint;
        ComputeBlockCoordinates();
        Efficiency = 0;

        DoModal();               // Launches modal dialog
}

CFunction::~CFunction()
{
}

void CFunction::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        DDX_Text(pDX, IDC_FUNCTION_NAME, GivenName); // Connects variable with dialog
element
}

BEGIN_MESSAGE_MAP(CFunction, CDialog)

END_MESSAGE_MAP()

void CFunction::ComputeBlockCoordinates()
{
        left = GeometricCenter.x - BLOCK_LENGTH / 2;
        right = GeometricCenter.x + BLOCK_LENGTH / 2;
        top = GeometricCenter.y - BLOCK_HEIGHT / 2;
        bottom = GeometricCenter.y + BLOCK_HEIGHT / 2;

        Anchors[0] = CPoint(right, (top + bottom) / 2);  // E
        Anchors[4] = CPoint(right, top);                         // NE
        Anchors[1] = CPoint((right + left) / 2, top);    // N
        Anchors[5] = CPoint(left, top);                          // NW

        Anchors[2] = CPoint(left, (top + bottom) / 2);   // W
        Anchors[6] = CPoint(left, bottom);                               // SW
        Anchors[3] = CPoint((left + right) / 2, bottom); // S
        Anchors[7] = CPoint(right, bottom);                      // SE

        Anchors[8] = CPoint(right, top + BLOCK_HEIGHT / 4);      // ENE
        Anchors[9] = CPoint(right - BLOCK_LENGTH / 4, top);      // NNE
        Anchors[10] = CPoint(left + BLOCK_LENGTH / 4, top);              // NNW
        Anchors[11] = CPoint(left, top + BLOCK_HEIGHT / 4);              // WNW

        Anchors[12] = CPoint(left, bottom - BLOCK_HEIGHT / 4);   // WSW
        Anchors[13] = CPoint(left + BLOCK_LENGTH / 4, bottom);   // SSW
        Anchors[14] = CPoint(right - BLOCK_LENGTH / 4, bottom);// SSE
        Anchors[15] = CPoint(right, bottom - BLOCK_HEIGHT / 4);  // ESE


                                         // The following lines reorders the anchors to a
different list,

                                         // AnchorsForBaggageFlows.  This list is
scrolled through when a

                                         // baggage flow (incoming or outgoing) needs to
be attached to

                                         // the function with only two nodes apart from
where the main

                                         // flow is attached.

        AnchorsForBaggageFlows[0] = Anchors[0];
        AnchorsForBaggageFlows[1] = Anchors[8];
        AnchorsForBaggageFlows[2] = Anchors[4];
        AnchorsForBaggageFlows[3] = Anchors[9];
        AnchorsForBaggageFlows[4] = Anchors[1];
        AnchorsForBaggageFlows[5] = Anchors[10];
        AnchorsForBaggageFlows[6] = Anchors[5];
        AnchorsForBaggageFlows[7] = Anchors[11];
        AnchorsForBaggageFlows[8] = Anchors[2];
        AnchorsForBaggageFlows[9] = Anchors[12];
        AnchorsForBaggageFlows[10] = Anchors[6];
        AnchorsForBaggageFlows[11] = Anchors[13];
        AnchorsForBaggageFlows[12] = Anchors[3];
        AnchorsForBaggageFlows[13] = Anchors[14];
        AnchorsForBaggageFlows[14] = Anchors[7];
        AnchorsForBaggageFlows[15] = Anchors[15];
}

void CFunction::DrawOnDC(CDC* pDC)
{

                CElement::DrawOnDC(pDC); // Call the drawing function of the parent class
- sets pen color

                if (this->NoInputAttached && this->NoOutputAttached)
                {
                        BrushR = DANGLING_BRUSH_R;
                        BrushG = DANGLING_BRUSH_G;
                        BrushB = DANGLING_BRUSH_B;
                }
                else
                {
                        BrushR = FUNCTION_BRUSH_R;
                        BrushG = FUNCTION_BRUSH_G;
                        BrushB = FUNCTION_BRUSH_B;
                }

                if (this->IsHidden)
                {
                        BrushR = HIDDEN_PEN_R;
                        BrushG = HIDDEN_PEN_G;
                        BrushB = HIDDEN_PEN_B;
                }


                CPen Pen;
                Pen.CreatePen(PS_SOLID, 2, RGB(PenR, PenG, PenB));
                CPen* pOldPen = pDC->SelectObject(&Pen);
                CBrush Brush(RGB(BrushR, BrushG, BrushB));
                CBrush* pOldBrush = pDC->SelectObject(&Brush);

                ComputeBlockCoordinates();
                CRect VerbRect(left, top, right, bottom);
                pDC->Rectangle(VerbRect);

                //pDC->Ellipse(GeometricCenter.x - 2, GeometricCenter.y - 2,
                //GeometricCenter.x + 2, GeometricCenter.y + 2);
                /*
                for (int AnchorInx = 1; AnchorInx <= 16; AnchorInx++)
                pDC->Ellipse(Anchors[AnchorInx - 1].x - 1, Anchors[AnchorInx - 1].y - 1,
                Anchors[AnchorInx - 1].x + 1, Anchors[AnchorInx - 1].y + 1);*/

                // Initializes a CFont object with the specified characteristics.
                CFont font;
                VERIFY(font.CreateFont(
                        16,                         // nHeight
                        0,                          // nWidth
                        0,                          // nEscapement
```

```
                          0,                          // nOrientation
                          FW_NORMAL,                  // nWeight
                          FALSE,                      // bItalic
                          FALSE,                      // bUnderline
                          0,                          // cStrikeOut
                          ANSI_CHARSET,               // nCharSet
                          OUT_DEFAULT_PRECIS,         // nOutPrecision
                          CLIP_DEFAULT_PRECIS,        // nClipPrecision
                          DEFAULT_QUALITY,            // nQuality
                          DEFAULT_PITCH | FF_SWISS,   // nPitchAndFamily
                          _T("Arial")));                  // lpszFacename

                  CFont* def_font = pDC->SelectObject(&font);
                  pDC->SetTextAlign(TA_CENTER | TA_BASELINE);
                  //if (this->IsHidden == false)
                  {
                          pDC->TextOut(GeometricCenter.x, GeometricCenter.y, GivenName);
                          pDC->SelectObject(def_font);
                  }

                  font.DeleteObject();
                  pDC->SelectObject(pOldPen);
                  pDC->SelectObject(pOldBrush);
}
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "Geometry.h"

#define GRID_SIZE 20
CGeometry::CGeometry(void)
{
}

CGeometry::~CGeometry(void)
{
}

int CGeometry::RoundToInteger(long Coordinate, int GridSize)
{
        int GridCountLower = int(Coordinate) / GridSize;
        if ((Coordinate - GridCountLower * GridSize) <= (GridSize / 2))
                return (GridCountLower * GridSize);
        else return (GridCountLower * GridSize + GridSize);
}

CPoint CGeometry::SnapToGrid(CPoint p)
{
        return CPoint(RoundToInteger(p.x, GRID_SIZE), RoundToInteger(p.y, GRID_SIZE));
}

long CGeometry::distance(CPoint p1, CPoint p2)
{
        return sqrt(pow((p1.x - p2.x), 2.0) + pow((p1.y - p2.y), 2.0));
}

CPoint* CGeometry::InterpolatePoints(CPoint p1, CPoint p2, double ratio)
{
        long x_new = ((p2.x - p1.x) * ratio) + p1.x;
        long y_new = ((p2.y - p1.y) * ratio) + p1.y;
        CPoint NewPoint(x_new, y_new);
        return &NewPoint;
}/**/
```

```
// MainFrm.cpp : implementation of the CMainFrame class
//

#include "stdafx.h"
#include "ConMod2.h"

#include "MainFrm.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CMainFrame

IMPLEMENT_DYNAMIC(CMainFrame, CMDIFrameWndEx)

const int  iMaxUserToolbars = 10;
const UINT uiFirstUserToolBarId = AFX_IDW_CONTROLBAR_FIRST + 40;
const UINT uiLastUserToolBarId = uiFirstUserToolBarId + iMaxUserToolbars - 1;

BEGIN_MESSAGE_MAP(CMainFrame, CMDIFrameWndEx)
        ON_WM_CREATE()
        ON_COMMAND(ID_WINDOW_MANAGER, &CMainFrame::OnWindowManager)
        ON_COMMAND(ID_VIEW_CUSTOMIZE, &CMainFrame::OnViewCustomize)
        ON_REGISTERED_MESSAGE(AFX_WM_CREATETOOLBAR, &CMainFrame::OnToolbarCreateNew)
        ON_COMMAND_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7,
&CMainFrame::OnApplicationLook)
        ON_UPDATE_COMMAND_UI_RANGE(ID_VIEW_APPLOOK_WIN_2000, ID_VIEW_APPLOOK_WINDOWS_7,
&CMainFrame::OnUpdateApplicationLook)
        ON_WM_SETTINGCHANGE()
END_MESSAGE_MAP()

static UINT indicators[] =
{
        ID_SEPARATOR,           // status line indicator
        ID_INDICATOR_CAPS,
        ID_INDICATOR_NUM,
        ID_INDICATOR_SCRL,
};

// CMainFrame construction/destruction

CMainFrame::CMainFrame()
{
        // TODO: add member initialization code here
        theApp.m_nAppLook = theApp.GetInt(_T("ApplicationLook"),
ID_VIEW_APPLOOK_VS_2008);
}

CMainFrame::~CMainFrame()
{
}

int CMainFrame::OnCreate(LPCREATESTRUCT lpCreateStruct)
```

```
{
        if (CMDIFrameWndEx::OnCreate(lpCreateStruct) == -1)
                return -1;

        BOOL bNameValid;

        CMDITabInfo mdiTabParams;
        mdiTabParams.m_style = CMFCTabCtrl::STYLE_3D_ONENOTE; // other styles 
available...
        mdiTabParams.m_bActiveTabCloseButton = TRUE;      // set to FALSE to place close 
button at right of tab area
        mdiTabParams.m_bTabIcons = FALSE;    // set to TRUE to enable document icons on 
MDI taba
        mdiTabParams.m_bAutoColor = TRUE;    // set to FALSE to disable auto-coloring of 
MDI tabs
        mdiTabParams.m_bDocumentMenu = TRUE; // enable the document menu at the right 
edge of the tab area
        EnableMDITabbedGroups(TRUE, mdiTabParams);

        if (!m_wndMenuBar.Create(this))
        {
                TRACE0("Failed to create menubar\n");
                return -1;      // fail to create
        }

        m_wndMenuBar.SetPaneStyle(m_wndMenuBar.GetPaneStyle() | CBRS_SIZE_DYNAMIC | 
CBRS_TOOLTIPS | CBRS_FLYBY);

        // prevent the menu bar from taking the focus on activation
        CMFCPopupMenu::SetForceMenuFocus(FALSE);

        if (!m_wndToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP 
| CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
                !m_wndToolBar.LoadToolBar(theApp.m_bHiColorIcons ? IDR_MAINFRAME_256 : 
IDR_MAINFRAME))
        {
                TRACE0("Failed to create toolbar\n");
                return -1;      // fail to create
        }

        // Custom ConMod2 toolbar controls:  PRIMITIVES TOOLBAR

        if (!m_primitivesToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | 
CBRS_LEFT | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
                !m_primitivesToolBar.LoadToolBar(IDR_PRIMITIVES, 0, 0, TRUE, 0, 0, 0))
        {
                TRACE0("Failed to create the PRIMITIVES toolbar\n");
                return -1;      // fail to create
        }

        //m_primitivesToolBar.EnableDocking(CBRS_ALIGN_ANY);
        //EnableDocking(CBRS_ALIGN_ANY);
        //DockPane(&m_primitivesToolBar);

        // Custom ConMod2 toolbar controls:  REASONING TOOLBAR
        if (!m_reasoningToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | 
CBRS_RIGHT | CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
                !m_reasoningToolBar.LoadToolBar(IDR_REASONING, 0, 0, TRUE, 0, 0, 0))
        {
```

```
		TRACE0("Failed to create the REASONING toolbar\n");
		return -1;      // fail to create
	}

	//m_reasoningToolBar.EnableDocking(CBRS_ALIGN_ANY);
	//EnableDocking(CBRS_ALIGN_ANY);
	//DockPane(&m_reasoningToolBar);

	// Custom ConMod2 toolbar controls:  FEATURES TOOLBAR

	//==================
	// Commented out for rolling back to Layer 1 (Chapter 6)
	//==================

	if (!m_featuresToolBar.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE |
CBRS_BOTTOM	| CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) ||
		!m_featuresToolBar.LoadToolBar(IDR_FEATURES, 0, 0, TRUE, 0, 0, 0))
	{
		TRACE0("Failed to create FEATURES toolbar\n");
		return -1;      // fail to create
	}

	//m_featuresToolBar.EnableDocking(CBRS_ALIGN_ANY);
	//EnableDocking(CBRS_ALIGN_ANY);
	//DockPane(&m_featuresToolBar);

	CString strToolBarName;
	bNameValid = strToolBarName.LoadString(IDS_TOOLBAR_STANDARD);
	ASSERT(bNameValid);
	m_wndToolBar.SetWindowText(strToolBarName);

	CString strToolBarName_PRIMITIVES;
	bNameValid = strToolBarName_PRIMITIVES.LoadString(IDS_TOOLBAR_PRIMITIVES);
	ASSERT(bNameValid);
	m_primitivesToolBar.SetWindowText(strToolBarName_PRIMITIVES);

	CString strToolBarName_REASONING;
	bNameValid = strToolBarName_REASONING.LoadString(IDS_TOOLBAR_REASONING);
	ASSERT(bNameValid);
	m_reasoningToolBar.SetWindowText(strToolBarName_REASONING);

	CString strToolBarName_FEATURES;
	bNameValid = strToolBarName_FEATURES.LoadString(IDS_TOOLBAR_FEATURES);
	ASSERT(bNameValid);
	m_primitivesToolBar.SetWindowText(strToolBarName_FEATURES);

	CString strCustomize;
	bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
	ASSERT(bNameValid);
	m_wndToolBar.EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE, strCustomize);

	// Allow user-defined toolbars operations:
	InitUserToolbars(NULL, uiFirstUserToolBarId, uiLastUserToolBarId);

	if (!m_wndStatusBar.Create(this))
	{
		TRACE0("Failed to create status bar\n");
		return -1;      // fail to create
	}
	m_wndStatusBar.SetIndicators(indicators, sizeof(indicators)/sizeof(UINT));
	// TODO: Delete these five lines if you don't want the toolbar and menubar to be
dockable
	m_wndMenuBar.EnableDocking(CBRS_ALIGN_ANY);
	m_wndToolBar.EnableDocking(CBRS_ALIGN_ANY);
	m_primitivesToolBar.EnableDocking(CBRS_ALIGN_ANY);
	m_reasoningToolBar.EnableDocking(CBRS_ALIGN_ANY);
	m_featuresToolBar.EnableDocking(CBRS_ALIGN_ANY);

	EnableDocking(CBRS_ALIGN_ANY);

	DockPane(&m_wndMenuBar);
	DockPane(&m_wndToolBar);
	DockPane(&m_primitivesToolBar);
	DockPane(&m_reasoningToolBar);
	DockPane(&m_featuresToolBar);

	// enable Visual Studio 2005 style docking window behavior
	CDockingManager::SetDockingMode(DT_SMART);
	// enable Visual Studio 2005 style docking window auto-hide behavior
	EnableAutoHidePanes(CBRS_ALIGN_ANY);

	// Load menu item image (not placed on any standard toolbars):
	CMFCToolBar::AddToolBarForImageCollection(IDR_MENU_IMAGES,
theApp.m_bHiColorIcons ? IDB_MENU_IMAGES_24 : 0);

	// create docking windows
	if (!CreateDockingWindows())
	{
		TRACE0("Failed to create docking windows\n");
		return -1;
	}

	m_wndFileView.EnableDocking(CBRS_ALIGN_ANY);
	m_wndClassView.EnableDocking(CBRS_ALIGN_ANY);
	DockPane(&m_wndFileView);
	CDockablePane* pTabbedBar = NULL;
	m_wndClassView.AttachToTabWnd(&m_wndFileView, DM_SHOW, TRUE, &pTabbedBar);
	m_wndOutput.EnableDocking(CBRS_ALIGN_ANY);
	DockPane(&m_wndOutput);
	m_wndProperties.EnableDocking(CBRS_ALIGN_ANY);
	DockPane(&m_wndProperties);

	// set the visual manager and style based on persisted value
	OnApplicationLook(theApp.m_nAppLook);

	// Enable enhanced windows management dialog
	EnableWindowsDialog(ID_WINDOW_MANAGER, ID_WINDOW_MANAGER, TRUE);

	// Enable toolbar and docking window menu replacement
	EnablePaneMenu(TRUE, ID_VIEW_CUSTOMIZE, strCustomize, ID_VIEW_TOOLBAR);

	// enable quick (Alt+drag) toolbar customization
	CMFCToolBar::EnableQuickCustomization();

	if (CMFCToolBar::GetUserImages() == NULL)
	{
		// load user-defined toolbar images
		if (m_UserImages.Load(_T(".\\UserImages.bmp")))
```

```
                {
                        CMFCToolBar::SetUserImages(&m_UserImages);
                }
        }

        // enable menu personalization (most-recently used commands)
        // TODO: define your own basic commands, ensuring that each pulldown menu has at
least one basic command.
        CList<UINT, UINT> lstBasicCommands;

        lstBasicCommands.AddTail(ID_FILE_NEW);
        lstBasicCommands.AddTail(ID_FILE_OPEN);
        lstBasicCommands.AddTail(ID_FILE_SAVE);
        lstBasicCommands.AddTail(ID_FILE_PRINT);
        lstBasicCommands.AddTail(ID_APP_EXIT);
        lstBasicCommands.AddTail(ID_EDIT_CUT);
        lstBasicCommands.AddTail(ID_EDIT_PASTE);
        lstBasicCommands.AddTail(ID_EDIT_UNDO);
        lstBasicCommands.AddTail(ID_APP_ABOUT);
        lstBasicCommands.AddTail(ID_VIEW_STATUS_BAR);
        lstBasicCommands.AddTail(ID_VIEW_TOOLBAR);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2003);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_VS_2005);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_BLUE);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_SILVER);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_BLACK);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_OFF_2007_AQUA);
        lstBasicCommands.AddTail(ID_VIEW_APPLOOK_WINDOWS_7);
        lstBasicCommands.AddTail(ID_SORTING_SORTALPHABETIC);
        lstBasicCommands.AddTail(ID_SORTING_SORTBYTYPE);
        lstBasicCommands.AddTail(ID_SORTING_SORTBYACCESS);
        lstBasicCommands.AddTail(ID_SORTING_GROUPBYTYPE);
        CMFCToolBar::SetBasicCommands(lstBasicCommands);

        // Switch the order of document name and application name on the window title
bar. This
        // improves the usability of the taskbar because the document name is visible
with the thumbnail.
        ModifyStyle(0, FWS_PREFIXTITLE);

        return 0;
}

BOOL CMainFrame::PreCreateWindow(CREATESTRUCT& cs)
{
        if( !CMDIFrameWndEx::PreCreateWindow(cs) )
                return FALSE;
        // TODO: Modify the Window class or styles here by modifying
        //  the CREATESTRUCT cs

        return TRUE;
}

BOOL CMainFrame::CreateDockingWindows()
{
        BOOL bNameValid;

        // Create class view
        CString strClassView;
```

```
	bNameValid = strClassView.LoadString(IDS_CLASS_VIEW);
	ASSERT(bNameValid);
	if (!m_wndClassView.Create(strClassView, this, CRect(0, 0, 200, 200), TRUE, 
ID_VIEW_CLASSVIEW, WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | CBRS_LEFT 
| CBRS_FLOAT_MULTI))
	{
		TRACE0("Failed to create Class View window\n");
		return FALSE; // failed to create
	}

	// Create file view
	CString strFileView;
	bNameValid = strFileView.LoadString(IDS_FILE_VIEW);
	ASSERT(bNameValid);
	if (!m_wndFileView.Create(strFileView, this, CRect(0, 0, 200, 200), TRUE, 
ID_VIEW_FILEVIEW, WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | CBRS_LEFT| 
CBRS_FLOAT_MULTI))
	{
		TRACE0("Failed to create File View window\n");
		return FALSE; // failed to create
	}

	// Create output window
	CString strOutputWnd;
	bNameValid = strOutputWnd.LoadString(IDS_OUTPUT_WND);
	ASSERT(bNameValid);
	if (!m_wndOutput.Create(strOutputWnd, this, CRect(0, 0, 100, 100), TRUE, 
ID_VIEW_OUTPUTWND, WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | 
CBRS_BOTTOM | CBRS_FLOAT_MULTI))
	{
		TRACE0("Failed to create Output window\n");
		return FALSE; // failed to create
	}

	// Create properties window
	CString strPropertiesWnd;
	bNameValid = strPropertiesWnd.LoadString(IDS_PROPERTIES_WND);
	ASSERT(bNameValid);
	if (!m_wndProperties.Create(strPropertiesWnd, this, CRect(0, 0, 200, 200), TRUE, 
ID_VIEW_PROPERTIESWND, WS_CHILD | WS_VISIBLE | WS_CLIPSIBLINGS | WS_CLIPCHILDREN | 
CBRS_RIGHT | CBRS_FLOAT_MULTI))
	{
		TRACE0("Failed to create Properties window\n");
		return FALSE; // failed to create
	}

	SetDockingWindowIcons(theApp.m_bHiColorIcons);
	return TRUE;
}

void CMainFrame::SetDockingWindowIcons(BOOL bHiColorIcons)
{
	HICON hFileViewIcon = (HICON) ::LoadImage(::AfxGetResourceHandle(), 
MAKEINTRESOURCE(bHiColorIcons ? IDI_FILE_VIEW_HC : IDI_FILE_VIEW), IMAGE_ICON, 
::GetSystemMetrics(SM_CXSMICON), ::GetSystemMetrics(SM_CYSMICON), 0);
	m_wndFileView.SetIcon(hFileViewIcon, FALSE);
```

```
        HICON hClassViewIcon = (HICON) ::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(bHiColorIcons ? IDI_CLASS_VIEW_HC : IDI_CLASS_VIEW), IMAGE_ICON,
::GetSystemMetrics(SM_CXSMICON), ::GetSystemMetrics(SM_CYSMICON), 0);
        m_wndClassView.SetIcon(hClassViewIcon, FALSE);

        HICON hOutputBarIcon = (HICON) ::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(bHiColorIcons ? IDI_OUTPUT_WND_HC : IDI_OUTPUT_WND), IMAGE_ICON,
::GetSystemMetrics(SM_CXSMICON), ::GetSystemMetrics(SM_CYSMICON), 0);
        m_wndOutput.SetIcon(hOutputBarIcon, FALSE);

        HICON hPropertiesBarIcon = (HICON) ::LoadImage(::AfxGetResourceHandle(),
MAKEINTRESOURCE(bHiColorIcons ? IDI_PROPERTIES_WND_HC : IDI_PROPERTIES_WND), IMAGE_ICON,
::GetSystemMetrics(SM_CXSMICON), ::GetSystemMetrics(SM_CYSMICON), 0);
        m_wndProperties.SetIcon(hPropertiesBarIcon, FALSE);

        UpdateMDITabbedBarsIcons();
}

// CMainFrame diagnostics

#ifdef _DEBUG
void CMainFrame::AssertValid() const
{
        CMDIFrameWndEx::AssertValid();
}

void CMainFrame::Dump(CDumpContext& dc) const
{
        CMDIFrameWndEx::Dump(dc);
}
#endif //_DEBUG
// CMainFrame message handlers

void CMainFrame::OnWindowManager()
{
        ShowWindowsDialog();
}

void CMainFrame::OnViewCustomize()
{
        CMFCToolBarsCustomizeDialog* pDlgCust = new CMFCToolBarsCustomizeDialog(this,
TRUE /* scan menus */);
        pDlgCust->EnableUserDefinedToolbars();
        pDlgCust->Create();
}

LRESULT CMainFrame::OnToolbarCreateNew(WPARAM wp,LPARAM lp)
{
        LRESULT lres = CMDIFrameWndEx::OnToolbarCreateNew(wp,lp);
        if (lres == 0)
        {
                return 0;
        }

        CMFCToolBar* pUserToolbar = (CMFCToolBar*)lres;
        ASSERT_VALID(pUserToolbar);

        BOOL bNameValid;
        CString strCustomize;
        bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
        ASSERT(bNameValid);

        pUserToolbar->EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE, strCustomize);
        return lres;
}

void CMainFrame::OnApplicationLook(UINT id)
{
        CWaitCursor wait;

        theApp.m_nAppLook = id;

        switch (theApp.m_nAppLook)
        {
        case ID_VIEW_APPLOOK_WIN_2000:
                CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManager));
                break;

        case ID_VIEW_APPLOOK_OFF_XP:

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerOfficeXP));
                break;

        case ID_VIEW_APPLOOK_WIN_XP:
                CMFCVisualManagerWindows::m_b3DTabsXPTheme = TRUE;

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows));
                break;

        case ID_VIEW_APPLOOK_OFF_2003:

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerOffice2003))
;
                CDockingManager::SetDockingMode(DT_SMART);
                break;

        case ID_VIEW_APPLOOK_VS_2005:

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerVS2005));
                CDockingManager::SetDockingMode(DT_SMART);
                break;

        case ID_VIEW_APPLOOK_VS_2008:

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerVS2008));
                CDockingManager::SetDockingMode(DT_SMART);
                break;

        case ID_VIEW_APPLOOK_WINDOWS_7:

        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerWindows7));
                CDockingManager::SetDockingMode(DT_SMART);
                break;

        default:
                switch (theApp.m_nAppLook)
                {
                case ID_VIEW_APPLOOK_OFF_2007_BLUE:
```

```
        CMFCVisualManagerOffice2007::SetStyle(CMFCVisualManagerOffice2007::Office2007_Lu
naBlue);
                        break;

                case ID_VIEW_APPLOOK_OFF_2007_BLACK:

        CMFCVisualManagerOffice2007::SetStyle(CMFCVisualManagerOffice2007::Office2007_Ob
sidianBlack);
                        break;

                case ID_VIEW_APPLOOK_OFF_2007_SILVER:

        CMFCVisualManagerOffice2007::SetStyle(CMFCVisualManagerOffice2007::Office2007_Si
lver);
                        break;

                case ID_VIEW_APPLOOK_OFF_2007_AQUA:

        CMFCVisualManagerOffice2007::SetStyle(CMFCVisualManagerOffice2007::Office2007_Aq
ua);
                        break;
                }


        CMFCVisualManager::SetDefaultManager(RUNTIME_CLASS(CMFCVisualManagerOffice2007))
;
                CDockingManager::SetDockingMode(DT_SMART);
        }

        m_wndOutput.UpdateFonts();
        RedrawWindow(NULL, NULL, RDW_ALLCHILDREN | RDW_INVALIDATE | RDW_UPDATENOW |
RDW_FRAME | RDW_ERASE);

        theApp.WriteInt(_T("ApplicationLook"), theApp.m_nAppLook);
}

void CMainFrame::OnUpdateApplicationLook(CCmdUI* pCmdUI)
{
        pCmdUI->SetRadio(theApp.m_nAppLook == pCmdUI->m_nID);
}


BOOL CMainFrame::LoadFrame(UINT nIDResource, DWORD dwDefaultStyle, CWnd* pParentWnd,
CCreateContext* pContext)
{
        // base class does the real work

        if (!CMDIFrameWndEx::LoadFrame(nIDResource, dwDefaultStyle, pParentWnd,
pContext))
        {
                return FALSE;
        }


        // enable customization button for all user toolbars
        BOOL bNameValid;
        CString strCustomize;
        bNameValid = strCustomize.LoadString(IDS_TOOLBAR_CUSTOMIZE);
        ASSERT(bNameValid);

        for (int i = 0; i < iMaxUserToolbars; i ++)
        {
                CMFCToolBar* pUserToolbar = GetUserToolBarByIndex(i);
                if (pUserToolbar != NULL)
                {
                        pUserToolbar->EnableCustomizeButton(TRUE, ID_VIEW_CUSTOMIZE,
strCustomize);
                }
        }

        return TRUE;
}


void CMainFrame::OnSettingChange(UINT uFlags, LPCTSTR lpszSection)
{
        CMDIFrameWndEx::OnSettingChange(uFlags, lpszSection);
        m_wndOutput.UpdateFonts();
}
```

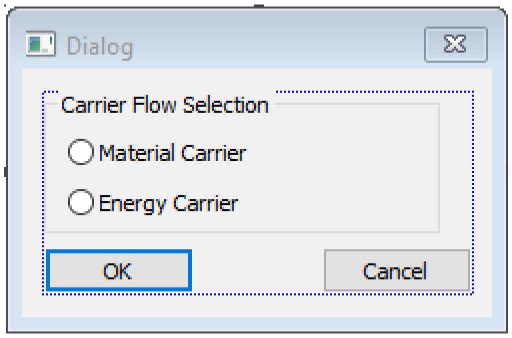
```
#include "stdafx.h"
#include "ConMod2.h"
#include "Material.h"
IMPLEMENT_DYNAMIC(CMaterial, CDialog)

CMaterial::CMaterial(CWnd* pParent,
        CPoint TailClick,
        CPoint HeadClick,
        CString* pCounterString,
        int ReasOpt)
        : CDialog(CMaterial::IDD, pParent)
        , GivenName(_T("M") + *pCounterString)
        , UI_IsResidual(false)
        , ReasoningOption(ReasOpt)
{
        TailPoint = TailClick;
        HeadPoint = HeadClick;
        GeometricCenter = *InterpolatePoints(TailPoint, HeadPoint, 0.5);
        StemThickness = MEDIUM;  // This sets the thickness of Material arrows

        HeadSize = EDGE_HEAD_SIZE;
        HalfHeadAngle = EDGE_HEAD_HALF_ANGLE;

        ComputeAnchorPoints();
        pHeadElem = NULL;
        pTailElem = NULL;

        DoModal();              // Launches modal dialog
}

BOOL CMaterial::OnInitDialog()
{
        CDialog::OnInitDialog();

        //==============================
        // Grey Out Dialog Controls
        //==============================
        if (this->ReasoningOption == QUALITATIVE_CONSERVATION)
                GetDlgItem(IDC_RESIDUAL_MATERIAL)->EnableWindow(false);   //Greys out
control

        if (this->ReasoningOption >= QUANTITATIVE_EFFICIENCY)
        {
                pMaterialTaxonomy = new CTreeCtrl;

                pMaterialTaxonomy->Create(WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP
|
                        TVS_HASLINES | TVS_HASBUTTONS | TVS_LINESATROOT |
                        /*TVS_SINGLEEXPAND | */TVS_SHOWSELALWAYS | TVS_TRACKSELECT,
                        CRect(11, 60, 248, 150), this, 0x1221);

                // Full List of all energy types (leaf nodes and intermediate nodes)
                HTREEITEM hMaterial, // Primary
                        hSolid, hLiquid, hGaseous; //Secondary under hMaterial

                                                                                // PRIMARY
LEVEL
                hMaterial = pMaterialTaxonomy->InsertItem(_T("M"), TVI_ROOT);

                // SECONDARY LEVEL (UNDER hMaterial)
                hSolid = pMaterialTaxonomy->InsertItem(_T("S"), hMaterial);
                hLiquid = pMaterialTaxonomy->InsertItem(_T("L"), hMaterial);
                hGaseous = pMaterialTaxonomy->InsertItem(_T("G"), hMaterial);

                pMaterialTaxonomy->SelectItem(hMaterialType);
                pMaterialTaxonomy->Expand(hMaterial, TVE_EXPAND);
        }

        return TRUE; // return TRUE unless you set the focus to a control
}

CMaterial::~CMaterial()
{
        //delete pMaterialTaxonomy;
}

void CMaterial::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        DDX_Text(pDX, IDC_MATERIAL_NAME, GivenName);             // Connects variable
IDC_MATERIAL_NAME to member GivenName

        DDX_Check(pDX, IDC_RESIDUAL_MATERIAL, UI_IsResidual);
}

void CMaterial::OnOK()
{
        CDialog::OnOK();

        if (this->ReasoningOption >= QUANTITATIVE_EFFICIENCY)
        {
                hMaterialType = pMaterialTaxonomy->GetSelectedItem();
                MaterialTypeName = pMaterialTaxonomy->GetItemText(hMaterialType);
                delete pMaterialTaxonomy;
        }
        else
                MaterialTypeName = "M";
}

void CMaterial::DrawOnDC(CDC* pDC)
{
                IsResidual = UI_IsResidual;

                CEdge::DrawOnDC(pDC);           // Execute the entire drawing code of
the parent class CEdge

                                                                          //
============================================================================
                                                                          // Write the
name of CMaterial using a Font object
                                                                          //
============================================================================

                                                                          // Initializes
a CFont object with the specified characteristics.
                CFont font;
                VERIFY(font.CreateFont(
                        FontSize,                     // nHeight
                        0,                      // nWidth
                        0,                      // nEscapement
                        0,                      // nOrientation
                        FW_NORMAL,              // nWeight
                        FALSE,                  // bItalic
                        FALSE,                  // bUnderline
                        0,                      // cStrikeOut
                        ANSI_CHARSET,           // nCharSet
                        OUT_DEFAULT_PRECIS,     // nOutPrecision
                        CLIP_DEFAULT_PRECIS,    // nClipPrecision
                        DEFAULT_QUALITY,        // nQuality
                        DEFAULT_PITCH | FF_SWISS,  // nPitchAndFamily
                        _T("Arial")));              // lpszFacename


//              if (this->IsHidden == false)
                {

                        CFont* def_font = pDC->SelectObject(&font);
                        pDC->SetTextAlign(TA_CENTER | TA_BASELINE);
                        pDC->TextOut(GeometricCenter.x, GeometricCenter.y, (GivenName +
_T(" [") + MaterialTypeName + _T("]")));
                        pDC->SelectObject(def_font);
                }
                //
============================================================================
                // Put back the old objects, although I do not understand how this
impacts anything.
                //
============================================================================

                font.DeleteObject();

}
```

```
BEGIN_MESSAGE_MAP(CMaterial, CDialog)
END_MESSAGE_MAP()
```

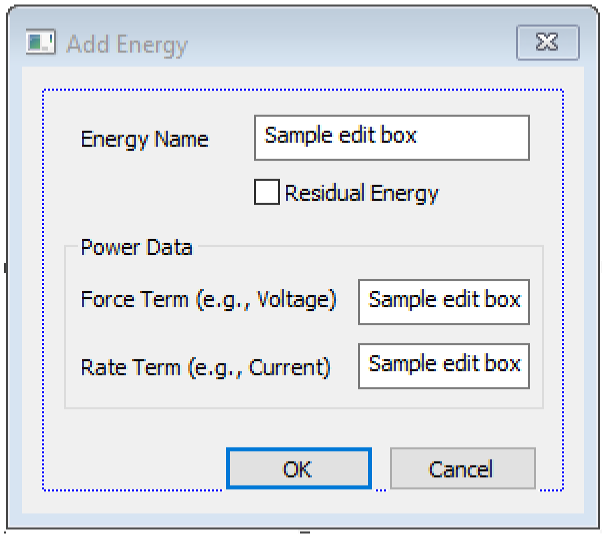
```
#include "stdafx.h"
#include "ConMod2.h"
#include "Node.h"

CNode::CNode(void)
{
}

CNode::~CNode(void)
{
}

void CNode::ComputeBlockCoordinates()
{}

/*void CNode::DrawOnDC(CDC* pDC)
{}*/
```

```
#include "stdafx.h"
#include "ConMod2.h"
#include "afxdialogex.h"
#include "Signal.h"


// CSignal dialog

IMPLEMENT_DYNAMIC(CSignal, CDialog)

CSignal::CSignal(CWnd* pParent, CPoint TailClick, CPoint HeadClick, CString*
pCounterString)
        : CDialog(CSignal::IDD, pParent)
        , GivenName(_T("S") + *pCounterString)
{
        TailPoint = TailClick;
        HeadPoint = HeadClick;
        GeometricCenter = *InterpolatePoints(TailPoint, HeadPoint, 0.5);
        StemThickness = THIN;  // This sets the thickness of signal arrows
        StemLineFont = PS_DOT;

        HeadSize = EDGE_HEAD_SIZE;
        HalfHeadAngle = EDGE_HEAD_HALF_ANGLE;

        ComputeAnchorPoints();
        pHeadElem = NULL;
        pTailElem = NULL;

        DoModal();              // Launches modal dialog
}

CSignal::~CSignal()
{
}

void CSignal::DoDataExchange(CDataExchange* pDX)
{
        CDialog::DoDataExchange(pDX);
        DDX_Text(pDX, IDC_SIGNAL_NAME, GivenName);               // Connects variable
IDC_SIGNAL_NAME to member GivenName
}

void CSignal::DrawOnDC(CDC* pDC)
{
        if (this->IsHidden == false)
        {
                CEdge::DrawOnDC(pDC);            // Execute the entire drawing code of
the parent class CEdge

                                                                          //
============================================================================
                                                                          // Write the
name of CMaterial using a Font object
                                                                          //
============================================================================

                                                                          // Initializes
a CFont object with the specified characteristics.
                CFont font;
                VERIFY(font.CreateFont(
                        FontSize,                         // nHeight
                        0,                          // nWidth
                        0,                          // nEscapement
                        0,                          // nOrientation
                        FW_NORMAL,                  // nWeight
                        FALSE,                      // bItalic
                        FALSE,                      // bUnderline
                        0,                          // cStrikeOut
                        ANSI_CHARSET,               // nCharSet
                        OUT_DEFAULT_PRECIS,         // nOutPrecision
                        CLIP_DEFAULT_PRECIS,        // nClipPrecision
                        DEFAULT_QUALITY,            // nQuality
                        DEFAULT_PITCH | FF_SWISS,   // nPitchAndFamily
                        _T("Arial")));                  // lpszFacename

                CFont* def_font = pDC->SelectObject(&font);
                pDC->SetTextAlign(TA_CENTER | TA_BASELINE);
                pDC->TextOut(GeometricCenter.x, GeometricCenter.y, GivenName);
                pDC->SelectObject(def_font);

                //
============================================================================
                // Put back the old objects, although I do not understand how this
impacts anything.
                //
============================================================================

                font.DeleteObject();
        }
}

BEGIN_MESSAGE_MAP(CSignal, CDialog)
END_MESSAGE_MAP()


// CSignal message handlers
// Template.cpp : implementation file
//

#include "stdafx.h"
#include "ConMod2.h"
#include "Template.h"


// TemplateCTemplate::CTemplate()
{

}

CTemplate::~CTemplate()
{
}
```

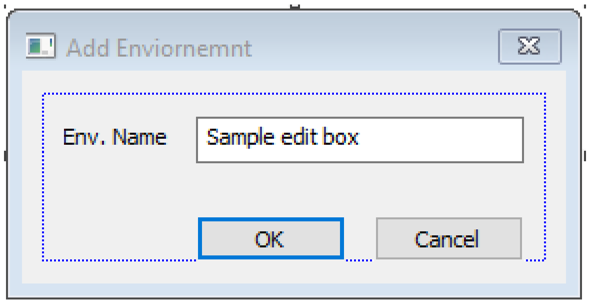
Appendix D
Resource files for ConMod 2.0



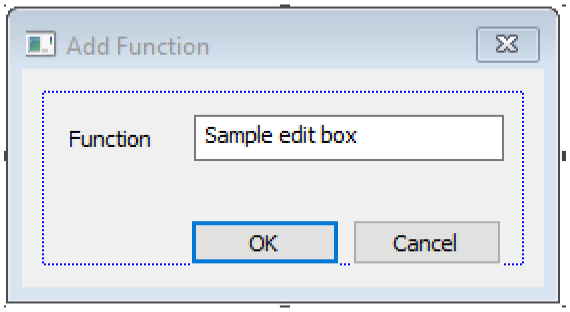
Dialog Box for Actuate_E



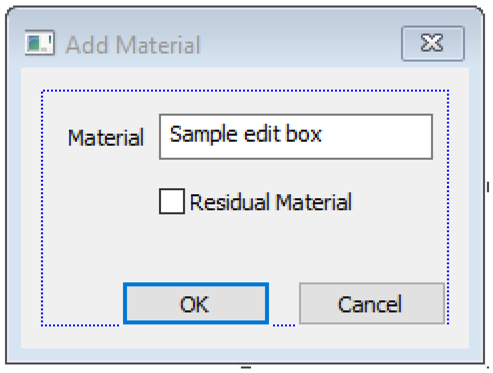
Dialog Box for an Energy Entity



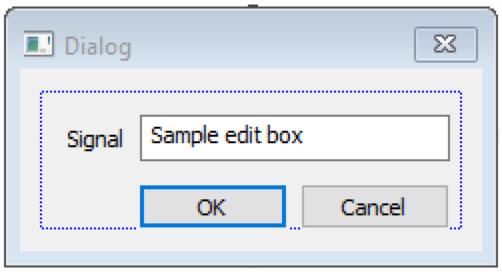
Dialog Box for an Environment Entity



Dialog Box for a Function Entity



Dialog Box for a Material Entity



Dialog Box for a Signal Entity