Thermal-Electrical Co-Simulation of Shipboard Integrated Power Systems on an All-Electric Ship

By

Matthew Andrew Pruske, B.S.M.E.

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment of the

Requirement for the Degree of

Master of Science in Engineering

The University of Texas at Austin

August, 2009

Thermal-Electrical Co-Simulation of Shipboard Integrated Power Systems on an All-Electric Ship

Approved by
Supervising Committee:

"Two little mice fell in a bucket of cream. The first mouse quickly gave up and drowned. The second mouse, wouldn't quit. He struggled so hard that eventually he churned that cream into butter and crawled out. Gentlemen, as of this moment, I am that second mouse."

-Christopher Walken, *Catch Me If You Can*

Acknowledgements

The following people offered me support and guidance during a long journey. Most importantly, they helped me maintain my sanity and peace of mind.

My parents, for supporting me throughout every endeavor I have ever undertaken.

Renee Tetrick, for listening.

Michael Pierce, for supporting me as a researcher and friend.

Dr. Kiehne, for challenging me to better my work and myself.

Abstract

Thermal-Electrical Co-Simulation of

Shipboard Integrated Power Systems of

an All-Electric Ship

By

Matthew Andrew Pruske, M.S.E.

The University of Texas at Austin, 2009

Co-Supervisor: Thomas M. Kiehne

Co-Supervisor: Carolyn C. Seepersad

The goal of the work reported herein has been to model aspects of the electrical distribution system of an all-electric ship (AES) and to couple electrical load behavior with the thermal management network aboard the ship. The development of a thermally dependent electrical network has built upon an in-house thermal management simulation environment to replace the existing steady state heat loads with dynamic, thermally dependent, electrical heat loads. Quantifying the close relationship between thermal and electrical systems is of fundamental importance in a large, integrated system like the AES.

This in-house thermal management environment, called the Dynamic Thermal Modeling and Simulation (*DTMS*) framework, provided the fundamental capabilities for modeling thermal systems and subsystems relevant to the AES. The motivation behind the initial work on DTMS was to understand the dynamics of thermal management aboard the ship. The first version, developed in 2007, captured the fundamental aspects of system-level thermal management while maintaining modularity and allowing for further development into other energy domains.

The reconfigurable nature of the *DTMS* framework allowed for the expansion into the electrical domain with the creation of an electrical distribution network in support of thermal simulations. The dynamics of the electrical distribution system of the AES were captured using reconfigurable and physics-based circuit elements that allow for thermal feedback to affect the behavior of the system. Following the creation of the electrical network, subsystems and systems were created to simulate electrical distribution. Then, again using the modularity features of *DTMS*, a thermal resistive heat flow network was created to capture the transient behavior of heat flow from the electrical network to the existing thermal management framework. This network provides the intimate link between the thermal management framework and the electrical distribution system.

Finally, the three frameworks (electrical, thermal resistive, and thermal management) were combined to quantify the impact that each system has relative to system-level operation. Simulations provide an indication of the unlimited configurations and potential design space a user of *DTMS* can explore to explore the design of an AES.

Table of Contents

List of Figures

List of Tables

1: Introduction

A variety of factors are changing the way the United States Navy approaches ship design and development. Fuel costs are volatile and the reliability of the international oil supply continues to be of concern to the United States from a national security standpoint. In addition, future implementation of new technologies in the arena of high-powered sensors and weapons will create significant challenges for shipboard energy utilization and thermal management. Finally, considering the lethality of current weapons systems, reconfigurable components have become a priority, and the focus on survivability and adaptability has increased.

To increase fuel efficiency and accommodate emerging technologies, the Navy is actively developing an Integrated Power System (IPS). In the IPS approach, power is generated, converted to electricity, and distributed to ship propulsion, sensor, weapon, and service loads in a timely and efficient while accommodating active control, optimization, and reconfigurability. The Navy calls this new generation of warship the all-electric ship (AES). This chapter will further elaborate on the motivation behind IPS, discuss the University of Texas research role for the AES, and outline the remaining chapters of this thesis.

1.1 Motivation Behind the All-Electric Ship

Efficiency and reliability are the motivation behind the thrust to convert the Navy fleet to all-electric systems. The legacy design approach employed gas turbines to power the main ship's propulsion while using auxiliary turbine-generator sets (gen-sets) to power remaining ship service needs. This configuration allocated approximately 90 % of the total, onboard power capacity toward propulsion on the DDG-51 [14]. As a result, a ship is rarely using the full capacity of the propulsion gen-sets, which in turn lowers fuel efficiency. The service power provided by the auxiliary gen-sets is not capable of powering additional high-powered sensors and weapons desired by the Navy to enhance system effectiveness. The under-utilized propulsion gen-sets and the increased demand for power led the Navy to adopt the IPS approach.

Under the IPS approach gen-sets produce power, which is converted into electrical power and distributed using a smart grid. Control of the grid allows the Navy to intelligently distribute electrical power, using Power Distribution Modules (PDM) and Power Converter Modules (PCM), to ship propulsion, high-powered loads, and service loads. Control over the grid allows for implementation of other methods of energy production and conservation (e.g., fuel cells and brake power fed to the grid) and energy storage (e.g., battery and fly wheel). This level of control has the potential to dramatically increase efficiency and flexibility of these future ships.

The proposed AES may have a grid distributing Medium Voltage Direct Current (MVDC). The desire for a MVDC grid is based on increasing power density of the next-generation AES, and the following advantages described in [13]. Using a MVDC system, the navy can decouple the prime mover from the grid, which allows for removal of reduction and speed increasing gears while relaxing restrictions on selection of a proper generator. Likewise, in a MVDC system, generators do not have to be synced in phase since the ac power is converted immediately to dc power. The new architecture allows for higher frequencies within the grid, which reduces size and weight of transformers. By delivering power through dc, the power factor is increased because of the reduction in electrical impedance. In addition, the MVDC system should reduce the acoustic signature of a ship because the power system now has a wider spectrum of operating frequencies.

1.2 Electric Ship Research and Development Consortium

With the Navy increasingly developing innovative technologies across their fleet of warships, the costs to build a ship are increasing. The Navy can no longer simply build a ship and work out the kinks later, as this is neither time nor cost effective. In response to a research need, the Office of Naval Research (ONR) formed and funded the Electric Ship Research and Development Consortium (ESRDC) in 2002 to assist in the design and optimization of the next-generation AES.

In forming the consortium, the Navy continues to realize its objective of developing an in-house modeling and simulation framework capable of assisting with the architectural design of the AES. Reconfigurability and modularity are stressed in the design of this ship, thus these features are required of any modeling and simulation framework. The end point demonstration of any software package should be the dynamic co-simulation of electrical, thermal, and mechanical systems with reliable controls and optimization capabilities.

1.3 Heat Generation

With the Navy leaning heavily towards implementing additional technologies with greater power densities, the reliability of the power grid is of great concern in maintaining survivability. Integral to the reliability and sustainability of power distribution is the performance of heat generating electronics within the electrical architecture. The semiconductor industry continues to improve the capability of switching devices, but the improved performance comes at a cost. Smaller devices and larger currents create greater heat fluxes and the performance of semiconductors deteriorates rapidly as the local temperature rises. Thus, the electronics must be cooled continuously, which will in turn require greater cooling capacity on the ship.

To meet the demands of the new IPS architecture more capable power conversion equipment will be required. A greater number of more capable Power Conversion Module (PCM) cabinets present a challenge in the thermal management arena as these cabinets will substantially increase zonal cooling demands. The PCM cabinets will be converting greater quantities of power than previous ship designs have encountered [27]. Converters may be expected to provide high power of about 30 MW from a few seconds to up to several minutes for pulsed power systems such as high power weapons and advanced sonar. Under heat loads generated by full capacity power conversion, current cabinets have heated to the point of failure and have only operated successfully under these operating conditions with the panels removed, an undesirable solution due to HVAC concerns [27]. The Navy estimates the cabinets could generate from 80 to 140 kW of heat, creating a need to design a more efficient means of removing heat from these mission critical systems.

1.4 Thermal-Electrical Co-Simulation

The goal of an electro-thermal co-simulation is to explain the intimate relationship between an electrical distribution system and the thermal management network.   The electrical system provides power to components aboard the AES including the chillers and pumps of the thermal management network. The electrical components also produce waste heat that must be removed by chilled water-cooling loops. The thermal management network is designed to prevent electrical components from heating up beyond operational temperatures, an occurrence that can lead to shut down and damage of the overlying system.

The *Dynamic Thermal Modeling and Simulation (DTMS*) framework was launched in December of 2007 with the fundamental capabilities of modeling thermal systems and subsystems relevant to the AES. This thesis presents the creation of a thermally dependent electrical power system and links this system and its subsequent heat losses with the existing thermal management capabilities of the *DTMS* framework. Prior to the creation of an electrical distribution framework in *DTMS*, heat created during the operation of an MVDC zonal distribution was simulated using steady state techniques. With the advent of the electrical distribution framework within *DTMS*, the dynamic loads of an MVDC system may now be linked to the existing thermal management framework. The goal is to understand the electrical heat dissipation and the consequences of thermal feedback to the electrical network, which introduces a design limitation to the development of future all-electric ships. Figure 1.1 shows a representation of a PCM including the four electronic bays and the bottom bay, which is reserved for thermal management (e.g. heat exchangers, cold plates). The goal of a thermal-electrical co-simulation is to link the electrical flow and heat dissipation within the cabinet to the fluid flow (i.e. thermal management) through the cabinet and quantify feedback between the two networks.

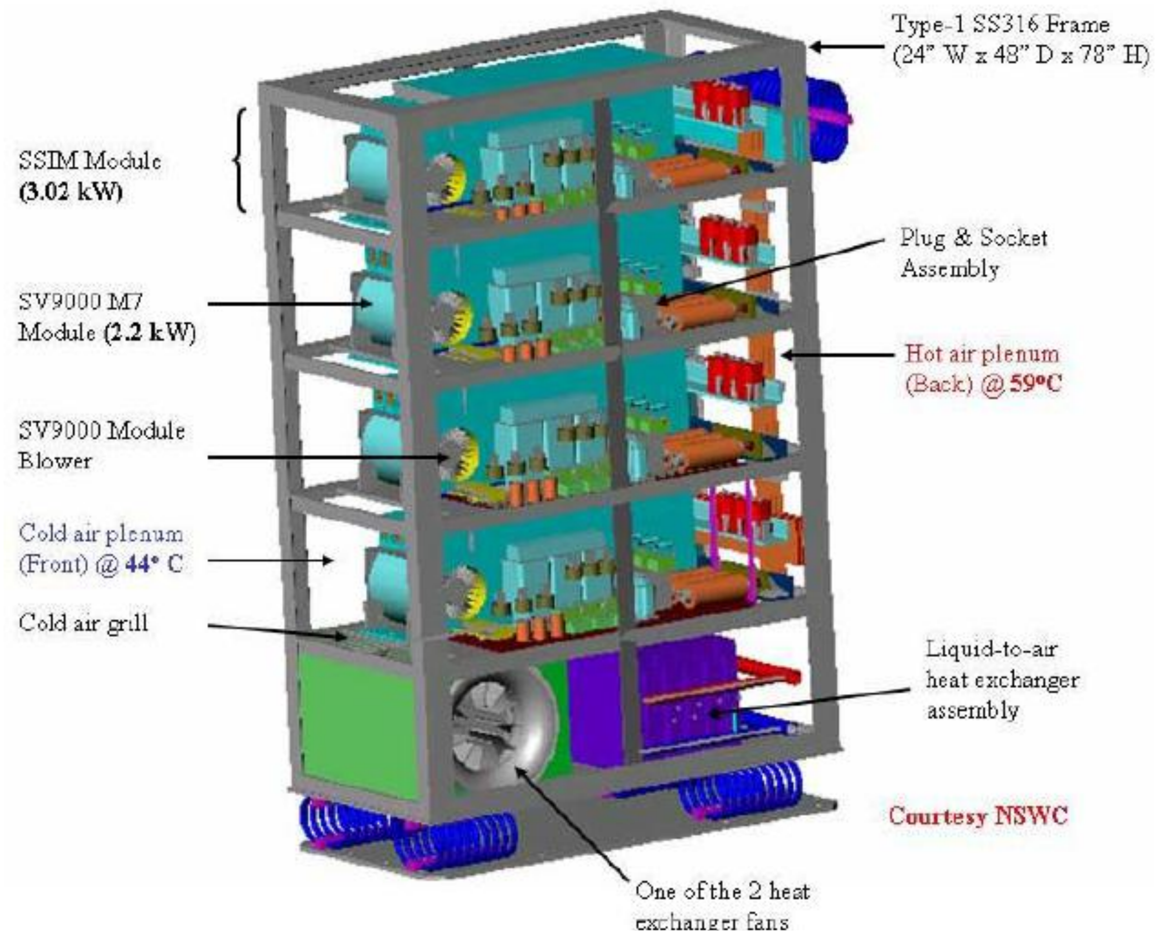


Figure 1.1: Representation of PCM [2]

Thermal-electrical co-simulation demands several steps to ensure proper feedback between the thermal management models and the electrical models. The following steps describe the procedure for implementing thermal feedback in a system-level programming environment:

1. The temperature of the electrical components updates electrical parameters.
2. The electrical system dissipates waste heat.
3. The thermal management network removes waste heat.
4. The thermal resistive network updates the temperature of the electrical components.

In a simulation one then simply iterates on steps 1 thru 4. Figure 1.2 graphically depicts the thermal-electrical co-simulation methodology employed in the *DTMS* framework.

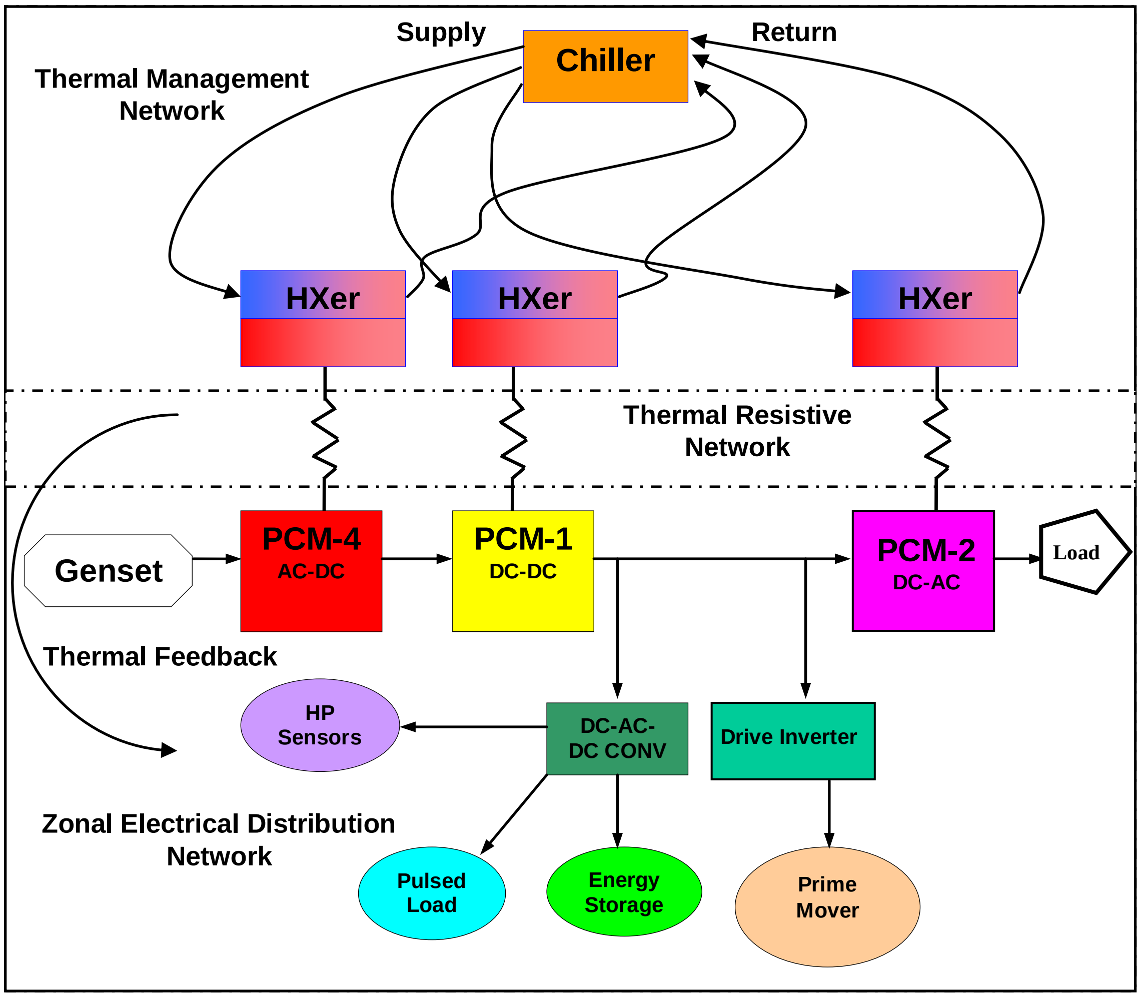


Figure 1.2: Thermal-electrical modeling in *DTMS*.

As electrical components heat up, the thermal management network must ramp up its cooling capacity. This in turn increases the amount of electrical power required from the electrical distribution system. Maintaining this balance is an optimization challenge, which is why the Navy seeks a modeling and simulation environment capable of evaluating the performance and stability of various system-wide configurations.

1.5 Thermal Modeling and Simulation at the University of Texas

This work is a continuation of previous work at the University of Texas which has contributed to the ESRDC in the thermal management arena since its inception in 2002. Previous efforts include a fuel cell model [3], a chiller model developed in *ProTrax* [9], an electromagnetic rail gun model [24], and an IPS model developed in *ProTrax* [11].

In 2006, the thermal management team embarked on a new approach to thermal modeling and simulation. Because commercial software, i.e., *ProTrax,* is expensive and often inflexible, an in-house framework was developed in C++, which is now called the *DTMS* framework [14]. This framework was developed with the capability of simulating a freshwater cooling system with its respective loads. Over the past several years, *DTMS* has been further developed to handle multiple fluids, two-phase flow, and a dynamic chiller simulation [10]. The goal of this continued development is to provide the Navy with a reconfigurable, modular, scalable and high fidelity modeling tool to aid in the design of the next-generation AES.

Previous and concurrent work within the ESRDC has focused on simulation of the electrical distribution system [16]. Other universities have addressed thermal-electrical interactions using constant electrical loads and efficiency coefficients of components within the electrical distribution network [8]. The work contained in this thesis addresses dynamic power loads and thermally dependent performance of components within the electrical distribution network coupled with the existing thermal management framework to create a complete system-level, thermal-electrical co-simulation.

1.6 Thesis Organization

With the objective of continuing to develop a physics-based, system-level modeling and simulation environment, Chapter 2 discusses the foundations of *DTMS* in relation to bond graph theory and modeling of physical systems. A brief tutorial on programming demonstrates why object-oriented programming achieves the goal of creating modular shipboard components based on bond graph theory. Chapter 2 concludes with a simple example of a simulation run using *DTMS* in order to show the possibilities of system-level configurations that can be modeled in the framework.

Because an electrical network did not exist prior to this work, Chapter 3 discusses the creation and evolution of modular electrical components, which uses the same physics-based strategies and solving techniques employed in the fluid flow network. Chapter 3 discusses the constitutive equations of each component along with the thermal impacts associated with the model as well as the resistive and switching losses associated with electrical components.

Chapter 4 discusses the creation of the building blocks of an electrical distribution system. The Power Conversion Modules (PCM) aboard the AES are assembled from modular electrical components in *DTMS*. Through the creation of reconfigurable PCM models, a *DTMS* user can construct a zone of any configuration. A developer may easily modify a PCM model should more information become available regarding its configuration and performance.

After the creation of system-level components in the electrical domain, Chapter 5 validates the framework by comparing results with previous modeling efforts. The simulation offered in this chapter shows the capability of the electrical framework in *DTMS* to simulate a multiple zone configuration onboard the AES.

In order to link the electrical heat loads to the existing thermal management framework, a thermal resistive heat flow network is necessary. Thus, Chapter 6 discusses the creation of this network in a similar fashion to the electrical and fluid flow networks. The chapter discusses the creation of new modular components such as an internal heat generation, heat flow, and energy storage components. Models that link the heat flow to the thermal management network through convection heat transfer are also discussed.

Chapter 7 demonstrates the capability of *DTMS* to model and simulate multiple zones of the AES while accounting for the intimate links between the heat flow, electrical, and thermal management networks. This chapter compares cooling strategies and discusses the importance of thermal feedback to the electrical system.

The thesis concludes in Chapter 8 with a discussion on the merits of the new thermal-electrical capabilities in *DTMS* as well as the drawbacks and limitations. Finally, Chapter 8 suggests further development of the electrical network and general improvements to the *DTMS* framework.

2. Dynamic Thermal Modeling and Simulation (*DTMS*) Framework

For several years the thermal management team at the University of Texas used and adapted commercial software to meet its thermal management simulation needs associated with the Navy's next-generation all-electric ship (AES). The most heavily used programs, *CycleTemp* and *ProTrax,* were selected at the outset to model and simulate the thermal, mechanical, and electrical behavior of system-level ship interactions. Several challenges were encountered which eventually led to a decision to move away from commercial software in favor of an in-house modeling environment.

At issue in the decision to move to an in-house modeling tool were cost, flexibility, responsiveness, utility, speed, and control over the modeling environment. The shift to internal development of the in-house modeling tool was brought on ultimately by the cost-effectiveness of the previously mentioned commercial software. This chapter serves as an introduction to the in-house *DTMS* Framework. The evolution, modularity, and reconfigurability of the framework are emphasized. A brief overview of object-oriented programming is also provided in order to provide a background for the reader of subsequent chapters. Finally, a simple simulation in *DTMS* is also provided to demonstrate its features and potential.

2.1 Evolution of *DTMS*

Because the Navy places particular emphasis on modular, reconfigurable, and customizable simulation tools, *DTMS* is founded on bond graph theory. Bond graph theory is, by its very nature, a modeling tool built upon modularity. The approach allows a user to connect efforts (pressure, voltage, force, momentum) in their respective energy domain (hydraulic, electrical, mechanical, rotational) with the respective flows (fluid, current, velocity, rotational velocity) while solving for the dependent intermediate efforts and flows given independent boundary conditions. This method allows the user to configure a network with any workable combination of efforts and flows to create a model of a physics-based system. The foundation for *DTMS* was originally a thermal-fluids, bond graph network, leaving the developer the option of expanding to other energy domains. Development has now expanded into the electrical, mechanical, and rotational energy domains to further demonstrate the capacity of *DTMS* to eventually handle the modeling and simulation of the electro-thermo-mechanical interactions of an AES. Subsequent chapters will discuss evolution of the electrical domain using bond graph techniques. For a more extensive overview of the bond graph techniques used in *DTMS*, consult Section 2.1.1 of [14].

2.2 Object-Oriented Programming and *DTMS*

DTMS has been developed in C++, an object-oriented language. Each element (efforts and flows) of the *DTMS* architecture has its own file, denoted as a *class* in C++ terminology. In *DTMS*, the base class (the foundation for all derived and specialized classes) is *DTMSModel*. This base class *encapsulates* the basic *methods* (functions) necessary to each effort and flow. These methods or functions include setting defaults, calculating states and state derivatives, and performing data output.

Within each class, *members* (parameters and functions) are separated into *private, protected,* and *public* members. Parameters and functions designated as private are only used within their respective file. Members designated as protected are accessible by derived classes while public members are accessible to all classes. This method of interface, known as *message passing,* is fundamental to the connection of efforts and flows in *DTMS*. Figure 2.1 is an example of a class with public and private members.

```
class CapacitiveFlowModel
{
protected:
    double displacement_;
    virtual double BackwardDifferenceDerivative();
public:
    virtual void setDefaults();
    virtual void calculateFlow();
};
```

Figure 2.1: Class *CapacitiveFlowModel*; example of a class with public and protected members.

The class is called *CapacitiveFlowModel* with protected members *displacement_* and *BackwardDifferenceDerivative(),*and public members *setDefaults()* and *calculateFlow()*. The member *displacement_* is a variable with numerical precision designated by the data type *double*. The function *BackwardDifferenceDerivative()* is designated a *double* because the method returns a value as data type double. The functions *setDefaults()* and *calculateFlow()* are designated *void* because these methods do not return a value.

Object oriented programming allows additional classes to be *derived* from previously defined classes for example: *ResistiveNetworkModel* is derived from *DTMSModel*, which in turn has derived subclasses of *ResistiveNetworkEffortModel*, *ResistiveNetworkFlowModel*, *CapacitiveFlowModel*, and *InertialFlowModel*. These effort and flow models *inherit* the previously mentioned protected and public methods attributed to their base class. These subclasses of efforts and flows are the modular building blocks from which the user may construct a system of any combination of energy domains. These subclasses are generic by nature to allow the developer to expand the *DTMS* framework into any energy domain. For example, the *CapacitiveFlowModel* currently has two derived subclasses of *Capacitor* and *Spring* in the electrical and mechanical energy domains, respectively. Figure 2.2 is an example of *inheritance* through a derived base class.

```
class Capacitor : public CapacitiveFlowModel
{
protected:
    double capacitance_;
    double temperature_;
    virtual void calculateTemperatureDependentCapacitance();
public:
    virtual void setCapacitance(double capacitance);
    virtual void setTemperature(double temperature);
};
```

Figure 2.2: Class *Capacitor*. Example of a derived class.

The derived class is *Capacitor*, the capacitive flow model in the electrical domain. This class inherits the backwards-differencing method from *CapacitiveFlowModel* as well as other protected and public members. The class *Capacitance* then further expands its capabilities with methods inherent to the electrical domain, such as the method *calculateTemperatureDependentCapacitance()*, which calculates the capacitance based on a reference capacitance and temperature.

In *DTMS*, a group model aptly named *ResistiveNetworkGroupModel* was created to develop specific engineering models to ensure ease of use. Classes derived from *ResistiveNetworkGroupModel* create *instances* of and *pointers* to previously defined classes constructed to simulate a specific shipboard component. An example of a group model is provided in Figure 2.3.

```
class PCM1 : public ResistiveNetworkGroupModel
{
protected:
    double efficiency_;
    Resistor* inletFlowModel_;
    std::vector<Resistor*> outletFlowModels_;
    Transformer transformer_;
    Inductor inductor_;
    Resistor resistor_;
    Capacitor capacitor_;
    ElectricalEffortModel effort_;
    virtual void calculateEfficiency();
public:
    virtual void addOutletFlowModel(Resistor * outletFlowModel);
    virtual void addInletFlowModel(Resistor * inletFlowModel);
    virtual void setPowerFactor(double powerFactor);
};
```

Figure 2.3. Class PCM1. Example of a group model with instances and pointers.

The derived class is *PCM1*, a power converter model. The model creates an instance of the classes *Transformer*, *Inductor*, *Resistor*, etc. during runtime. The class also creates pointers to an instance of *Resistor* (*inletFlowModel_*) and a vector of instances of *Resistor* (*outletFlowModels_*). These pointers point to the value of a specified object at the respective address assigned at runtime. In this example, the pointers point to the inlet and outlet current models of the converter.

2.3 *DTMS* Simulation

Currently there is no user interface by which to interact with *DTMS*. The framework consists of individual files for models, controls, and solvers. A main file must be constructed consisting of instances of the existing files to run a simulation of a physical system.

As an example, an RC circuit with dc current would physically consist of a voltage source, a resistor, a capacitor, and grounding. In *DTMS*, this system is comprised of the classes *ElectricalEffortModel*, *Resistor*, and *Capacitor*. Three instances of the *ElectricalEffortModel* are used to simulate the voltage source, the node potential between the capacitor and resistor, and ground. Figure 2.4 provides a view of the incomplete main file containing only instances of the electrical models.

```
#include "DTMSFramework.h"
using namespace DTMSFramework;

int main()
{
    //Electrical efforts
    ElectricalEffortModel source("Source");
    source.setVoltage(500);
    source.setDependent(false);

    ElectricalEffortModel midEffort("MidEffort");

    ElectricalEffortModel ground("Ground");
    ground.setVoltage(0);
    ground.setDependent(false);

    //Capacitor
    Capacitor capacitor("Capacitor");
    capacitor.setCapacitance(2);
    capacitor.setTemperature(300);

    //Resistor
    Resistor resistor("Resistor");
    resistor.setResistance(3);
    resistor.setTemperature(300);
```

Figure 2.4: Main file with instances of electronic models.

The first line, *#include "DTMSFramework.h"*, ensures that all *include* files of every header file in *DTMS* is included. The next line, *using namespace DTMSFramework;*,tells the compiler that the instances and functions used within the main file are items found in the container or "library" of *DTMSFramework*. A namespace container provides disambiguation in the case that a simulation includes two different namespaces each containing items of the same name. Within the *main file*, the instances of the electrical effort models are declared and allow the user to input the voltage and dependency using functions *setVoltage* and *setDependent*. The boundary nodes *source* and *ground* are independent in this simulation, thus their dependency is set to *false* (the default value is *true*). The *midEffort* instance is dependent and has a time-varying voltage, thus the dependency and voltage do not need to be set.

The instance of *Capacitor* allows the user to input the capacitance and initial temperature using functions *setCapacitance* and *setTemperature*. Likewise, the instance of *Resistor* allows the user to input resistance and initial temperature. An initial temperature creates a resistance or capacitance dependent on thermal feedback. Every flow model in *DTMS* is required to solve for flow (f) as a function of effort (*e)* where

$$f = f(\Delta e) \tag{2.1}$$

Therefore, both the *Capacitor* and *Resistor* internally solve for current as a function of voltage.

Once the instances of models are created, they must be connected. Each *ElectricalEffortModel* maintains the ability to add unlimited current models through the functions *addInletFlowModel* and *addOutletFlowModel.* Each flow model maintains the ability to add one inlet and one outlet effort through *setInletEffortModel* and *setOutletEffortModel.* These functions are inherent to every effort and flow model in every energy domain. Figure 2.5 is the main file of Figure 2.4 continued with the connections for the RC circuit, which are declared below declaration of the *Resistor.*

```
//Resistor
Resistor resistor("Resistor");
resistor.setResistance(3);
resistor.setTemperature(300);

//Connect the circuit
source.addOutletFlowModel(capacitor);
capacitor.setOutletEffortModel(midNode);
midNode.addOutletFlowModel(resistor);
resistor.setOutletEffortModel(ground);
```

Figure 2.5: Continued main file with connections.

The next class added to the main file is the simulation solver. *DTMS* currently employs three solvers: a linear solver, a nonlinear solver using the Newton-Raphson method, and a globally convergent solver.

The class *ResistiveNetworkSolver* is a linear solver, which populates a matrix with the efforts, flows, and conductance (or resistance) of the network and solves for conservation of flow at each of the dependent efforts. The system of equations that populate the matrix is as follows:

$$\bar{C}_D \bar{e}_D + \bar{f} + \bar{H} + \bar{C}_I \bar{e}_I = 0 \tag{2.2}$$

where *C* is the conductance vector, *e* is the effort vector, *f* is the flow vector, *H* is the head flow vector, and the subscripts *D* and *I* indicate dependent and independent values. In the electrical domain, the head represents a source flow. Using linear algebra, the dependent values of the efforts are resolved accordingly in the solver.

The class *NewtonRaphsonResistiveSolver* is a nonlinear solver which utilizes a Taylor series expansion of the conservation of flow at each dependent effort to solve for the potentials. The flows are summed at each dependent effort model and linearized using a first order approximation for the flow partial derivative with respect to the effort. The solver iterates on the following system of equations until conservation of flow is achieved.

$$\bar{e} = \bar{e}_o - \bar{\bar{J}}^{-1} \bar{f}(\bar{e}_o) \tag{2.3}$$

Here e is the potential at each node, e_o is the reference effort, f is the flow, and J is the Jacobian matrix consisting of partial derivatives of the flows.

The class *GloballyConvergentResistiveSolver* is an adapted version of the Newton Raphson method except that the solver checks to see if the solution is closer to the root. If not, the step size is reduced until convergence is achieved. Each solver contains the function *setErrorTolerance* for the user to indicate an acceptable error term at each time step.

Following declaration of the solver, an instance of the class *DTMSSimulation* is created to run the simulation. The main functions in this class are *setDefaults*, *initialize*, *calculateState* and *calculateStateDerivatives*, and a method to output the data to a .csv file. These functions are called at every time step in its derived classes to update the characteristics of each model. The declaration of *DTMSSimulation* also allows the user to input the name of the output file, the simulation time period, and the time step and write step of the simulation. In Figure 2.6 these values are RC_Test, 100, 0.1, and 0.1, respectively.

Once an instance of *DTMSSimulation* and a solver are created, the simulation executive adds the solver and all models through the functions *addSolver* and *addModel*. The solver in turn must also add each instance of the models through the function *addModel*. The last item is for the user to indicate which models must output data through the function *setWriteFlag*. Figure 2.6 completes the main file for the RC circuit by adding the instances of *DTMSSimulation*, *GloballyConvergentResistiveSolver*, the respective models for the simulation executive and solver, and declaring for which instances to output data. Finally, the command *executive.runSimulation()* actually runs the simulation.

```
    midNode.addOutletFlowModel(resistor);
    resistor.setOutletEffortModel(ground);

    //System solver
    GloballyConvergentResistiveSolver simulationSolver;
    simulationSolver.setErrorTolerance(1e-4);
    simulationSolver.addModel(source);
    simulationSolver.addModel(capacitor);
    simulationSolver.addModel(midNode);
    simulationSolver.addModel(resistor);
    simulationSolver.addModel(ground);

    //Simulation executive
    DTMSSimulation executive("RC_Test.csv",100,0.1,0.1);
    executive.addModel(source);
    executive.addModel(capacitor);
    executive.addModel(midNode);
    executive.addModel(resistor);
    executive.addModel(ground);
    executive.addSolver(simulationSolver);

    //Set write flags
    source.setWriteFlag(1);
    capacitor.setWriteFlag(1);
    midNode.setWriteFlag(1);
    resistor.setWriteFlag(1);
    ground.setWriteFlag(1);

    //Run simulation
    executive.runSimulation();

    return 0;
```

Figure 2.6: Completion of the main file with simulation executive and solver.

This brief overview of *DTMS* provides a glimpse into the inner workings and capabilities of the framework. For a more comprehensive overview of the code, consult [14] and [10]. For a more extensive tutorial on construction of a main file, consult [15].

3. Electrical Framework

The goal of the electro-thermal simulation is to link the heat losses associated with an electrical distribution network with the current thermal management framework of *DTMS* and then provide feedback to the electrical network to update electrical properties. Because the *DTMS Framework* was founded on the principle of efforts and flows, *DTMS* was easily adapted to construct an electrical framework capable of simulating the dynamics of a shipboard electrical system. This chapter provides details concerning the evolution of *DTMS* within the electrical domain. Subsequent chapters will discuss overlay of the thermal management framework within *DTMS* over the newly constructed electrical network.

3.1 Three-Phase Power and RMS Values

Three-Phase Power

The Navy brought the ESRDC together to examine architectural alternatives for a future AES. While the final architectural layout has not been finalized, the electrical distribution network will clearly contain multiple stages of power conversion between ac and dc voltage. Thus, the decision must be made whether the thermal management modeling framework will use ac or rms values for voltage and current. Using ac values raises two modeling issues: simulation of reactive power and simulation run time.

The issue of simulation of reactive power is a concern because the PCM class members contain inductive and capacitive elements. Any electrical system containing these components causes three-phase ac voltage to drop out of phase with the alternating current due to energy storage within components. To maintain an acceptable power factor (discussed later in this chapter) controls must be implemented.

The second issue surrounds simulation run time in *DTMS*. Electrical simulations require smaller time steps than a thermal network due to stability considerations in numerical integration schemes. Thus, the run time is significantly larger for an electrical simulation. Introducing a time varying source voltage only exacerbates an already slow and laborious run time.

Anticipating these potential issues and with the aim of avoiding them, rms voltage sources are implemented in the electrical framework. The prospect of using rms rather than ac values stems from the fact that three-phase power does not vary with time [22] since

$$P_{total} = V_{L1} i_{L1} + V_{L2} i_{L2} + V_{L3} i_{L3} = \frac{3}{2} V_P i_P \tag{3.1}$$

where P_{total} is the total power, V_{L1}, V_{L2}, V_{L3} are the three-phase line voltages, V_P is the peak voltage, and i_P is the peak current. Treating three phase power in this way is a helpful modeling tool for creating the lossless conversion of ac power to dc power and back again in PCM models focused on issues of thermal management.

Determining Root Mean Squared Values

The amplitude of an alternating voltage source (peak voltage) can be determined from rms values of the voltage source and vice versa [4]. The rms value of voltage, V_{rms}, is determined by integrating the peak voltage, V_p, over one period:

$$V_{rms} = \sqrt{\frac{1}{T} \int_0^T (V_P \sin(\omega t))^2 \, dt} \tag{3.2}$$

where T is the period of the wave function and ω is the frequency. Extracting peak voltage from the square root and using a trigonometric identity this equation becomes

$$V_{rms} = V_P \sqrt{\frac{1}{T} \int_0^T \frac{(1 - \cos(2\omega t))}{2} dt} \tag{3.3}$$

After integration, the expression becomes

$$V_{rms} = V_P \sqrt{\frac{1}{T} \left(\frac{t}{2} - \frac{\sin(2\omega t)}{4\omega} \right) \Bigg|^T} \tag{3.4}$$

Noting that sin ($2\omega t$) evaluated over T equals zero, this equation becomes

$$V_{rms} = \frac{V_P}{\sqrt{2}} \tag{3.5}$$

In the same fashion, the rms value of current, i_{rms}, is determined to be

$$i_{rms} = \frac{i_P}{\sqrt{2}} \tag{3.6}$$

where i_P is the peak current. Substituting Equations 3.5 and 3.6 into Equation 3.1, the total power, P_{total}, as it relates to the rms value of voltage, V_{rms}, and the rms value of current, i_{rms}, is described by the following expression:

$$P_{total} = 3 \cdot V_{rms} i_{rms} \tag{3.7}$$

Alternating Current Effort Model

A three-phase alternating current group model, *ACElectricalEffortModel* was created in *DTMS* consisting of three instances of *SineElectricalEffortModel*. The three-phase model allows the user to input the amplitude and frequency of the wave function.

Since the use of rms values rather than a three-phase ac model is sufficient for the task of an electro-thermal co-simulation, the *ACElectricalEffortModel*, consisting of three-phases, is not currently implemented into the *DTMS Framework*, though it remains a future modeling option.

3.2 Harmonics, Impedance and Power Factor

Due to the increased utilization of rectifiers, as in a pulse-width-modulation (PWM) converter, harmonics and a decreased power factor have become prominent and limiting factors in the design and implementation of power electronics. These factors can lead to power quality issues including surges in the neutral current of three-phase four wire circuits, increased thermal loads and shorter life spans of transformers and motors, and poor quality of voltage source waveforms [1]. These undesirable side effects lead to the general degradation of power systems. Since the current *DTMS Framework* uses rms values, the effects of harmonics and the power factor must be addressed with separate models. The following sections address harmonics and power factor issues and the approach used in *DTMS*.

With a linear load, the current drawn from a sinusoidal voltage source remains sinusoidal, though not necessarily in phase. Harmonics are introduced into ac systems when a nonlinear load (e.g., a rectifier or induction motor) is attached to an ac voltage source. These harmonics are due to, for example, the switching in converters where nonlinear loads draw pulsed currents from the ac source. Under the influence of a nonlinear load, the nonlinear current drawn is composed of sinusoidal waves of various harmonic frequencies. The harmonic numbers, *h*, depend on the number of rectified current pulses per cycle, *p*, and obey the following equation:

$$\begin{aligned} h &= np \pm 1 \\ n &= 1,2,3\ldots \end{aligned} \tag{3.8}$$

These harmonic frequencies are integer multiples of the fundamental frequency, the lowest frequency of a periodic waveform. In theory, the percentage of harmonic current distortion obeys a reciprocal rule where the percentage equals one over the harmonic number. Figure 3.1 shows the current spectrum for a 6-pulse, pulse-width modular, where *h* =1, 5,7,11…

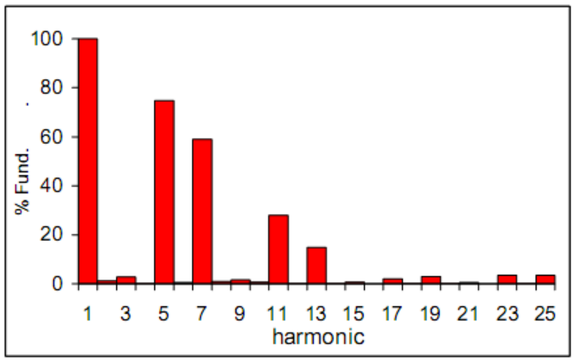


Figure 3.1. Example harmonic current spectrum for a 6-pulse PWM. The x-axis represents the harmonics and the y-axis represents the percentage of harmonic current distortion. [1]

Harmonics Equations

The total voltage, $V(t)$, of a distorted sinusoidal waveform can be determined by summing the dc voltage component along with the harmonic sinusoidal waveforms. Ignoring the dc component, this summation can be written as follows:

$$V(t) = \sum_{h=1}^{\infty} V_h(t) = \sum_{h=1}^{\infty} \sqrt{2} V_h \sin(h\omega_0 t + \phi_h) \qquad (3.9)$$

where h is the harmonic integer, ω_0 is the fundamental frequency, t is time, and ϕ_h is the phase shift. The rms value of the voltage, V_{rms}, can be determined using the following expression:

$$V_{rms} = \sqrt{\frac{1}{T} \int_0^T V^2(t)dt} \qquad (3.10)$$

where T is the period of the waveform and t is time. Evaluating this expression, the rms value of voltage is simply:

$$V_{rms} = \sqrt{V_1^2 + V_2^2 + ... + V_n^2} = \sqrt{V_{fund}^2 + V_{harm}^2} \tag{3.11}$$

and similarly for the rms current,

$$i_{rms} = \sqrt{i_1^2 + i_2^2 + ... + i_n^2} = \sqrt{i_{fund}^2 + i_{harm}^2} \tag{3.12}$$

from these it can be seen that the rms values of voltage and current are composed of both fundamental and harmonic values [1]. With increased harmonics from inductive and capacitive elements, the rms value of the current is observed to increase. Henceforth, current and voltage are assumed to be rms values unless noted otherwise.

Adverse Effects of Harmonics

Although harmonics degrade many power system components, the focus here is on a consideration of the adverse effects that pertain to power conversion equipment, in particular transformers, cabling, and capacitors.

Thermal issues in regard to transformers involve copper losses (resistive) and iron losses (eddy current and hysteresis). While these losses exist in the absence of harmonics, harmonics exacerbate these losses by increasing the rms current flowing through transformers. Copper or resistive losses occur through the electrical resistance of the conductor. The power losses, $P_{cu,loss}$, related to the copper losses are those of a resistor.

$$P_{cu,Loss} = iV = i^2 R \tag{3.13}$$

where i is the rms current, V is the rms voltage, and R is the winding resistance. Eddy currents are caused by the existence of a moving electrical field. Similar to copper losses, eddy currents create Joule heating with behavior described by the following expression.

$$P_{TECLoss} = P_{FECLoss} \sum_{h=1}^{\infty} i_h^2 h^2 \tag{3.14}$$

where $P_{TECLoss}$ is the total eddy current loss, $P_{FECLoss}$ is the eddy current loss at the fundamental frequency, and i_h is the current at harmonic, h. While eddy currents exist at the fundamental frequency, harmonics magnify the effect.

Thermal losses in cables occur from resistive losses and skin effects. As with resistors and transformers, these losses are resistive in nature, i.e., due to Joule heating.

$$P_{Loss} = i^2 R \tag{3.15}$$

Cables also suffer from skin effects where current will tend to flow through the outer surface of the conductor to circumvent the higher resistance in the center of an axial cable. In large cables, skin effects are negligible among greater harmonic frequencies. In *DTMS*, thermal losses in cables are modeled as resistive in nature.

Capacitors absorb harmonic voltages by nature. They are used in “snubbers” to reduce voltage transients during a step load and to mitigate the effect of harmonics. As a result, capacitors exhibit dielectric losses described by the following expression.

$$P_{DielectricLoss} = V^2 \omega C \tan(90^\circ - \phi) \tag{3.16}$$

where ω is the frequency, C is the capacitance, and ϕ is the circuit phase angle. This can also be represented as a resistor (as is the case in *DTMS*) where

$$P_{DielectricLoss} = \frac{V^2}{R} \tag{3.17}$$

and R=1/(ωCtan(90°-ϕ)). [1]

Power Factor

The power factor, *pf*, of an electrical network is the ratio of real power, *P*, to apparent power, *S*:

$$pf = P / S \tag{3.18}$$

In an ac network, a power factor of 1.0 corresponds with the sinusoidal voltage and current in phase with each other. This is the case with a purely resistive load. In nonlinear systems, like the PCM family, the inductive and capacitive elements introduce harmonics that distort waveforms causing the voltage and current to be out of phase with each other, thus reducing the power factor and increasing reactive power, *Q*, where

$$S^2 = P^2 + Q^2 \tag{3.19}$$

Reactive power does not have the ability to do work, and does not produce heat. Reactive power is stored in the magnetic and electrical fields of the inductive and capacitive elements and is returned to the electrical source at the end of each cycle. This phenomenon often results in large current spikes in the neutral wire of a four-wire ac cabling system and significantly degrades power conversion equipment [1].

Modeling Harmonics in DTMS

Because *DTMS* does not currently employ a sinusoidal ac effort model as the voltage source, the effects of harmonics and a reduced power factor may be modeled either as voltage drops described by Ohm's Law or by assigning a system power factor to the PCM.

The total harmonic voltage drop across a system can be found using Ohm's Law by recognizing the resistance as an impedance of individual elements across a circuit to obtain the following:

$$V_{thd} = \sum_{h=1}^{n} i_h Z_h \qquad (3.20)$$

where V_{thd} is the total harmonic distortion voltage and i_h and Z_h are respectively the current and impedance of the circuit elements at harmonic number, *h*. In *DTMS,* a resistor with no resistive heat production is used to represent this voltage drop.

Currently, the PCM models employ a power factor to account for harmonics and out of phase current distortion. Values of power factors can vary significantly depending on the contribution of nonlinear components (transistors, diodes, inductors, capacitors). Because each of the converters of the PCM class contains many nonlinear elements, [17] suggests a maximum attainable power factor of 0.96, which is the default power factor applied to the PCM models in *DTMS.*

3.3 Circuit Elements

The following sections discuss the circuit elements, which comprise the framework of the electrical domain. In keeping with the bond graph approach of efforts and flows established in *DTMS*, current is established as the flow. Voltage is established as the effort and acts at nodes to connect the flow of current. Effort models neither create nor store flow. Therefore the circuit solver adjusts efforts until flow conservation is enforced at each effort model. Thus, each flow model is required to provide an equation to solve for the current through the circuit element. Each elemental model must then use its constitutive equation to solve for flow as a function of potential.

$$i = f(V) \tag{3.21}$$

This electrical framework provides the foundation for assembling group models of circuit elements to create PCM models. While it is not necessary to model each PCM element down to individual components, it is necessary to model the fundamental behavior of the electrical power system including resistance, conductance and capacitance. Switches are also modeled by including generic diodes and transistors. This groundwork will allow for future modeling of generators, motors, pulsed power and energy storage.

Electrical components heat up during operation and in turn their properties change as function of local temperature. This chapter provides an approach for modeling properties related to temperature transience, while Chapter 6 provides the framework for removing heat from the electrical system.

ElectricalEffortModel

In *DTMS*, voltage nodes are modeled as class *ElectricalEffort,* which is derived from the base class *ResistiveNetworkEffortModel.* The model allows the user to set the voltage through the function *setVoltage()* while inheriting the function *setDependent()* from *ResistiveNetworkModel.*

ACElectricalEffortModel

Alternating current electrical voltage nodes are modeled as class *ACElectricalEffortModel* with the primary function of *setAmplitude()* to establish the peak voltage of the waveform of a pure ac voltage source.

DCElectricalEffortModel

The class *DCElectricalEffortModel* was created solely to provide a means to convert ac peak voltage to dc voltage in a rectifier model. In an ideal three-phase rectifier, the three-phase ac voltage, V_{ac}, is sinusoidal and must be converted to dc voltage, V_{dc}, through a sinusoidal Pulse Width Modulation (PWM). The average switch voltages for each ac line are as follows:

$$\begin{aligned} \left. V_{ac,1}(t) \right|_{T_s} &= d_1(t) \cdot \left. V_{dc}(t) \right|_{T_s} \\ \left. V_{ac,2}(t) \right|_{T_s} &= d_2(t) \cdot \left. V_{dc}(t) \right|_{T_s} \\ \left. V_{ac,3}(t) \right|_{T_s} &= d_3(t) \cdot \left. V_{dc}(t) \right|_{T_s} \end{aligned} \tag{3.22}$$

where $V_{ac,n}$ is the ac line voltage with n=1,2,3 for the three respective lines, T_s is the switching period, and d_n is the sinusoidal PWM duty cycle. To convert the sinusoidal ac voltage to dc voltage the duty cycle must be three-phase sinusoidal. Thus the duty cycles for the three lines of ac voltage are as follows:

$$\begin{aligned} d_1(t) &= D_0 + \tfrac{1}{2} D_M \sin(\omega t) \\ d_2(t) &= D_0 + \tfrac{1}{2} D_M \sin(\omega t - 120^\circ) \\ d_3(t) &= D_0 + \tfrac{1}{2} D_M \sin(\omega t - 240^\circ) \end{aligned} \tag{3.23}$$

where d_n is the line duty cycle, D_0 is the dc bias, D_M is the modulation index, and ω is the ac line frequency. The ac line voltages are as follows:

$$\begin{aligned} V_{ac,1}(t) &= V_M \sin(\omega t) \\ V_{ac,2}(t) &= V_M \sin(\omega t - 120^\circ) \\ V_{ac,3}(t) &= V_M \sin(\omega t - 240^\circ) \end{aligned} \tag{3.24}$$

where V_M is the peak line voltage. The average line-to-line switch voltages are as follows:

$$\begin{aligned} V_{12}(t)\big|_{T_s} &= V_{ac,1}(t)\big|_{T_s} - V_{ac,2}(t)\big|_{T_s} = \big(d_1(t) - d_2(t)\big)V_{dc}(t)\big|_{T_s} \\ V_{23}(t)\big|_{T_s} &= V_{ac,2}(t)\big|_{T_s} - V_{ac,3}(t)\big|_{T_s} = \big(d_2(t) - d_3(t)\big)V_{dc}(t)\big|_{T_s} \\ V_{31}(t)\big|_{T_s} &= V_{ac,3}(t)\big|_{T_s} - V_{ac,1}(t)\big|_{T_s} = \big(d_3(t) - d_1(t)\big)V_{dc}(t)\big|_{T_s} \end{aligned} \tag{3.25}$$

Substituting Equations 3.23 and 3.24 into the first equation of the set 3.25 reveals that

$$V_M\big(\sin(\omega t) - \sin(\omega t - 120°)\big) = \tfrac{1}{2} D_M \big(\sin(\omega t) - \sin(\omega t - 120°)\big)V_{dc}(t)\big|_{T_s} \tag{3.26}$$

Combining the sine terms, Equation 3.26 becomes

$$V_M = \tfrac{1}{2} D_M V_{dc}(t)\big|_{T_s} \tag{3.27}$$

This solution can be found for each of the equations in set 3.25. Thus, dc voltage, the voltage after passing through the rectifier bridge is simply a function of the peak line voltage and the modulation index of the PWM [7]. Therefore, the *DCElectricalEffortModel* employs the *setModulationIndex()* function to establish the modulation index.

Resistor

Resistors are modeled as class *Resistor*, which is derived from the base class *LinearFlowModel*. The constitutive relationship for a resistor is the following:

$$V = i_R R \tag{3.28}$$

where V is the voltage across the resistor, i_R, is the current through the resistor, and R is the resistance. Thus, current as a function of the voltage across a resistor is the following:

$$i_R = V / R \tag{3.29}$$

The net resistance can be found using the following expression:

$$R = \frac{l \cdot \rho}{A} \tag{3.30}$$

where l is the length of the conductor, A is its cross sectional area, and ρ is the electrical conductivity. The resistance is temperature dependent and is typically expressed in the following way:

$$R = R_{ref}\left(1 + \alpha(T - T_{ref})\right) \tag{3.31}$$

where R_{ref} is the resistance at the reference temperature T_{ref} (typically room temperature), T is the current temperature, and α is a constant based on material properties [19].

Heat losses associated with the resistor are termed Joule heating, which are characterized by the following expression:

$$P_{loss} = V^2 / R \tag{3.32}$$

where V is the voltage across the resistor. In *DTMS*, the class *Resistor* is used in a variety of models including resistive elements, switching losses, circuit breakers, resistive loads, and impedance.

Inductor

The ideal inductor is modeled as class *Inductor* and is derived from the base class *InertialFlowModel*, the generic inertial flow model. The constitutive relationship for an ideal inductor exhibiting no resistance or capacitance is the following:

$$L \cdot i_L = \lambda \tag{3.33}$$

where L is the inductance, i_L is the current through the inductor, and λ is the flux linkage (equivalent to momentum in the mechanical domain). Knowing that voltage is the integral of the flux linkage with respect to time, current can be determined using the following,

$$i_L = \frac{1}{L}\int V dt \tag{3.34}$$

where *V* is the voltage across the inductor.

First, second, third and fourth order numerical integration schemes (Closed Newton-Cotes Methods [26]) were implemented in the *DTMS* framework to solve for the current. The current is set for the initial time step by specifying the initial flux linkage in the function *setFluxLinkage*. Current flow is integrated over the first time step using the first order scheme (Trapezoid Rule):

$$i_1 = i_0 + \frac{h}{2L}(V_0 + V_1) \tag{3.35}$$

where i_1 is the present value of the current, V_1 is the current voltage across the inductor, i_0 is the initial current, V_0 is the initial voltage across the inductor, and *h* is the time step. At the second time step, a second order scheme is used (Simpson's Rule) where the current is integrated over the first two time steps:

$$i_2 = i_0 + \frac{h}{3L}(V_0 + 4V_1 + V_2) \tag{3.36}$$

where i_2 is the present value of the current and V_2 is the present voltage across the inductor. At the third time step, a third order scheme is used (Simpson's 3/8's Rule) where the current is integrated over the first three time steps:

$$i_3 = i_0 + \frac{3h}{8L}(V_0 + 3V_1 + 3V_2 + V_3) \tag{3.37}$$

where i_3 is the present value of the current and V_3 is the present voltage across the inductor. At the fourth time step, a third order scheme is used (Boole's Rule) where the current is integrated over the first four time steps:

$$i_4 = i_0 + \frac{2h}{45L}(7V_0 + 32V_1 + 12V_2 + 32V_3 + 7V_4) \tag{3.38}$$

where i_4 is the present value of the current and V_4 is the present voltage across the inductor. After the fourth time step, this four step algorithm is repeated where i_4 and V_4 are now the new initial values, i_0 and V_0 respectively, for the next four time steps.

Table 3.1 lists the four numerical integration schemes and their respective error terms. For example, the error for the fourth order Boole's rule is on the order of h^7, where h is the time step. Thus, error is a function of the time step.

Table 3.1: Four step numerical integration scheme.

Numerical Integration Scheme [26]		**Error**
1st Order (Trapezoid Rule)	$i_1 = i_0 + \frac{h}{2L}(V_0 + V_1)$	$\vartheta(h^3)$
2nd Order (Simpson's Rule)	$i_2 = i_0 + \frac{h}{3L}(V_0 + 4V_1 + V_2)$	$\vartheta(h^5)$
3rd Order (Simpson's 3/8 Rule)	$i_3 = i_0 + \frac{3h}{8L}(V_0 + 3V_1 + 3V_2 + V_3)$	$\vartheta(h^5)$
4th Order (Boole's Rule)	$i_4 = i_0 + \frac{2h}{45L}(7V_0 + 32V_1 + 12V_2 + 32V_3 + 7V_4)$	$\vartheta(h^7)$

Inductors are not ideal and exhibit losses in the form of winding losses and core losses. Winding losses consist of dc and ac losses. Conductive winding losses related to dc losses are similar to those of a resistor and can be characterized by equation 3.32. Due to the skin effect (discussed in Section 3.2), winding losses related to the ac component of the current decay with depth into the wire. This loss is difficult to characterize and manufacturers do not make this data readily available. Core losses are associated with hysteresis and can be characterized by the following expression:

$$P_{core} = k \cdot f^{x} B^{y} \cdot V_{e} \tag{3.39}$$

where k is a constant based on the material, f is the frequency, B is the peak flux density, x is the frequency exponent, y is the flux density exponent, and V_e is the effective core volume. Manufacturers provide core losses more readily than they provide the constants necessary to use Equation 3.39 [6]. Induction losses in *DTMS* are modeled using a resistor in series with the inductor.

Capacitor

The ideal capacitor is modeled as *class Capacitor* and is derived from class *CapacitiveFlowModel*, the generic capacitive flow model. The constitutive relationship for an ideal capacitor with no resistance or inductance is the following:

$$q = C \cdot V \tag{3.40}$$

where C is the capacitance, q is the charge through the inductor, and V is the voltage across the capacitor. Assuming the capacitance is not a function of time, the current through a capacitor, i_c, can be found as follows:

$$i_c = C\frac{dV}{dt} \tag{3.41}$$

First, second, third and fourth order numerical differentiation schemes were implemented in the *DTMS* framework to solve for this flow. Current flow is set for the initial time step by specifying the charge in the function *setCharge*. At the first time step, a first order backwards differencing method is used to solve for current:

$$i_1 = \frac{C}{h}(V_1 - V_0) \tag{3.42}$$

where i_1 is the present value of the current, V_1 is the present voltage across the capacitor, V_0 is the initial voltage, and h is the time step. At the second time step, a second order backward differencing scheme is used to solve for current:

$$i_2 = \frac{C}{2h}(3V_2 - 4V_1 + V_0) \tag{3.43}$$

where i_2 is the present value of the current and V_2 is the present voltage across the capacitor. At the third time step, a third order scheme is used to solve for current:

$$i_3 = \frac{C}{6h}(11V_3 - 18V_2 + 9V_1 - 2V_0) \tag{3.44}$$

where i_3 is the present value of the current and V_3 is the present voltage across the capacitor. At the fourth time step, a fourth order scheme is used to solve for current:

$$i_4 = \frac{C}{12h}(25V_4 - 48V_3 + 36V_2 - 16V_1 + 3V_0) \tag{3.50}$$

where i_4 is the present value of the current and V_4 is the present voltage across the capacitor. Each time step after the fourth time step now has four previous values of the voltage. Therefore, the fourth order backwards differencing scheme is used from the fourth time step until the end of the simulation. Table 3.2 lists the four backwards differencing schemes.

Table 3.2: Four numerical differentiation schemes.

Numerical Differentiation Scheme [25]	
1st Order	$i_1 = \frac{C}{h}(V_1 - V_0)$
2nd Order	$i_2 = \frac{C}{2h}(3V_2 - 4V_1 + V_0)$
3rd Order	$i_3 = \frac{C}{6h}(11V_3 - 18V_2 + 9V_1 - 2V_0)$
4th Order	$i_4 = \frac{C}{12h}(25V_4 - 48V_3 + 36V_2 - 16V_1 + 3V_0)$

The capacitance of a capacitor is also dependent on temperature. While this relationship is highly nonlinear at extreme temperatures, within the range of 20 to 100°C the function is linear for most dielectric materials [23]. In this range, the following expression describes the behavior of capacitance as a function of temperature:

$$C = C_{ref}\left(1 + \beta(T - T_{ref})\right) \tag{3.51}$$

where C_{ref} is the capacitance at the reference temperature T_{ref} (typically room temperature), T is the current temperature, and β is a constant based on material properties.

The temperature range of 20 to 100°C is adequate to cover the expected temperature range of shipboard converters due to the fact that at 100°C, electrical degradation occurs [27]. This linear relationship allows for both an increase and decrease of capacitance with increasing temperature. With an increase in temperature from 20°C to 100°C, capacitors typically increase or decrease in capacitance within ± 3 percent of the reference capacitance. The default temperature dependence for the class *Capacitor* is an increase of 1 percent in capacitance with an increase in temperature from 20°C to 100°C.

Validation

To demonstrate the validity of the fourth order schemes for integration and differentiation, an exact solution is derived below for a simple RCL circuit as shown below and then compared to the numerical solution in *DTMS*. Figure 3.2 displays the circuit.

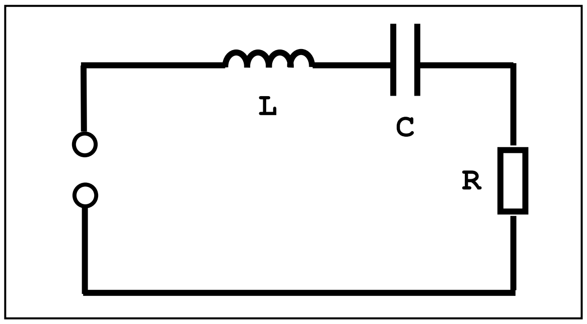


Figure 3.2: RCL series circuit.

By summing voltage drops across the circuit, the following second-order differential equation is obtained:

$$Lq'' + Rq' + \frac{1}{C}q = 0 \tag{3.52}$$

where L is the inductance, R is the resistance, C is the capacitance, q is the charge, and q' is the current. The solution to differential Equation 3.52 is

$$q = Ae^{m_1 t} + Be^{m_2 t} \tag{3.53}$$

where m_1 and m_2 are found by solving the following quadratic equation:

$$m_{1,2} = \frac{-R \pm \sqrt{R^2 - \frac{4 \cdot L}{C}}}{2 \cdot L} \tag{3.54}$$

The constants *A* and *B* are found by applying initial conditions. Table 3.3 lists the input parameters and initial conditions to obtain an analytical solution.

Table 3.3: Input parameters for RCL validation circuit.

Input Parameters for Equations 3.28 and 3.29	
R	3 ohms
C	2 farads
L	4 henries
q(t=0)	500 coulombs
q’(t=0)	0 amperes

The constants m_1 and m_2 are found to be -0.25 s^{-1} and -0.5 s^{-1}, respectively. The equations for *q* and *q’* are the following:

$$q = 1000e^{-0.25t} - 500e^{-0.5t} \tag{3.55}$$

$$q = -250e^{-0.25t} + 250e^{-0.5t} \tag{3.56}$$

Figure 3.3 below compares the exact solution with the numerical solution found using *DTMS* for both charge, *q*, and current, *q'*. The numerical solution lies directly on top of the exact solution. The relative error is found to be less than 0.02 % when comparing the numerical and exact solutions.

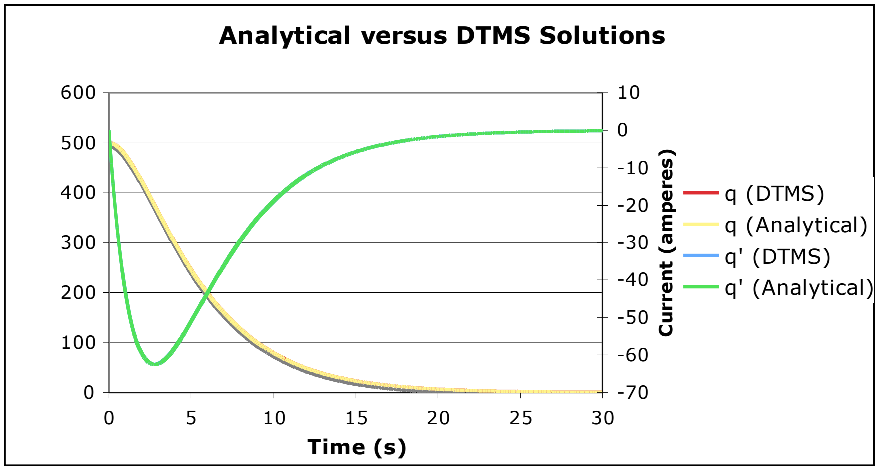


Figure 3.3: Exact versus numerical solutions of RCL Circuit.

Switching Elements

As semiconductor devices become smaller and switching frequencies increase, the switching losses associated with these elements are becoming more problematic. While instantaneous switching losses are small, the power losses become significant over time. Switching times are approaching extremely small periods, on the order of nanoseconds [7].

Because the electrical time constant is typically orders of magnitude smaller than the thermal time constant, an averaged switch modeling strategy is used in what follows. In *DTMS*, switches are modeled as a generic transistor and a diode with losses averaged over the switching time period.

Diode

During the switching event a diode goes through several stages. When the diode is reverse-biased (blocking reverse voltage, off-state) charge is stored until the voltage is positive and current flows. When the diode achieves charge equilibrium (on-state), a negative load current or the recombination of electrons initiates turn-off of the diode, and the stored charge is removed. The time required for the diode to recover is deemed the reverse recovery time, T_r. Switching losses in the diode are a combination of the removal of the recovered charge, Q_r, and the concurrent reverse recovery time, t_r [7]. The current loss, i_{loss}, associated with the diode during the switching event can be found using the following expression:

$$i_{loss}\big|_{T_s} = \frac{1}{T_s}\int_0^{T_s} i_1(t)dt = \frac{1}{T_s}(Q_r + t_r \cdot i_2\big|_{T_s}) \tag{3.57}$$

where T_s is the switching period, i_1 is the current of the diode in its on-state and i_2 is the current of the transistor (the switch counterpart to the diode) in its on-state. The total power consumed by switching losses can be found by the following equation:

$$P_{sw} = V\left(\frac{Q_r}{T_s} + \frac{t_r}{T_s} \cdot i_2\big|_{T_s}\right) \tag{3.58}$$

In *DTMS* the model classes *DiodeReverseRecovery* and *DiodeRecoveredCharge* account for these losses. Both are derived from the base class *LinearFlowModel*.

Transistor

Transistors block current during the off-state and conduct current during the on-state. The period of time, τ, that a transistor is conducting is a function of the switching period, T_s, and the duty cycle, D, such that:

$$\tau = D \cdot T_s \tag{3.59}$$

During the off-state, the resistance of a transistor approaches infinity while during the on-state the transistor is said to have an on-resistance, R_{on} [7]. Thus, the current, i_T, through a transistor is modeled in *DTMS* as a resistor that accounts for the duty cycle as seen in the following expression:

$$i_T = \frac{V}{D \cdot R_{on}} \tag{3.60}$$

where *V* is the voltage across the transistor. The class *TransistorOnResistance* represents the switching loss of a transistor in *DTMS* and is derived from the class *Resistor.*

Ideal Transformer

The ideal transformer in *DTMS* is modeled as class *Transformer* derived from the base class *ResistiveNetworkGroupModel.* The function of the ideal transformer is to step voltage up or down while transferring electrical power from one circuit to another. The *Transformer* model is structured as a group model to accommodate one instance of each of the following models: *TransformerEffort1*, *LinearFlowModel*, *ElectricalEffortModel*, and *TransformerFlow2.* The class *TransformerEffort1* is derived from the class *ElectricalEffortModel* while the *TransformerFlow2* is derived from the class *Resistor.*

The following are the constitutive relationships for an ideal transformer:

$$V_s = \frac{N_s}{N_p} V_p \tag{3.61}$$

$$i_s = \frac{N_p}{N_s} i_p \tag{3.62}$$

where *V* is voltage, *i* is current, *N* is the number of windings, the subscript *p* represents the primary circuit and the subscript *s* represents the secondary circuit. The secondary voltage, V_s, is proportional to the primary voltage, $V_{p,}$ based on the turns ratio (N_s divided by N_p). Thus, the secondary voltage may be stepped up or stepped down depending upon the construction of the windings in the transformer.

Figure 3.4 illustrates interactions within the transformer model in accordance with Bond Graph theory. Following causality, the source of effort, S_e, provides the primary voltage, V_p, to the transformer, which is stepped up or down to the secondary voltage, V_s, according to Equation 3.61. Once the voltage is set in the secondary circuit, the current flow, i_s, is determined by its interaction with its respective circuit. Then following causality, flow i_s determines flow i_p according to Equation 3.62.

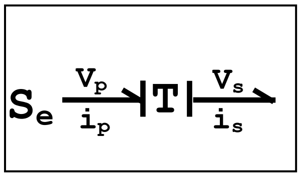


Figure 3.4: Bond graph of a transformer element.

The transformer model employs the C++ method of containment, where the *Transformer* model acts as a wrapper to relate two circuit networks that are solved separately by the network solver, but at the same time intimately linked by Equations 3.61 and 3.62. Table 3.4 lists the four models contained in *Transformer*, the respective effort or flow that they represent within the transformer, and their primary function in transferring lossless power.

Table 3.4: Contained Models in *Transformer and respective functions*

Contained models in *DTMS* model *Transformer*		
TransformerEffort1	V_p	*calculateEffort2(), setEffort2()*
LinearFlowModel	i_p	Flow is set by *TransformerFlow2*
ElectricalEffortModel	V_s	Effort is set by *TransformerEffort1*
TransformerFlow2	i_s	*calculateFlow1(), setFlow1()*

The transformer model is used in the power converter models to simulate ideal (lossless) switching devices.

Transformer losses consist of core, eddy current and winding losses. Losses via conduction through the core consist of sinusoidal hysteresis and eddy current losses. Hysteresis losses in the core are similar in nature to Equation 3.39 for an inductor and are modeled as follows:

$$P_{core} = k \cdot f^{x} B^{y} (aT^{2} - bT + c) \tag{3.63}$$

where *k* is a constant based on the material, *f* is the frequency, *B* is the peak flux density, *x* is the frequency exponent, *y* is the flux density exponent, *T* is temperature, and *a*, *b*, and *c* are constants. Winding losses are caused by Joule heating and therefore Equation 3.32 is appropriate to describe these conduction losses. Eddy current losses per volume can be expressed as a function of the frequency and total flux as follows:

$$P_{e} = \frac{\pi^{2} f^{2} B_{\max}^{2} a^{2} \sigma^{2}}{6} \tag{3.63}$$

where *f* is frequency, *B* is the maximum flux density, *a* is the lamination thickness, and σ is the thermal conductivity. Since the loss is a function of the lamination thickness, constructing the transformer core out of thin lamination rather than using a solid core will make eddy current losses negligible.

3.4 Conclusion

This chapter presents the physics-based approach utilized in *DTMS* to simulate the electrical distribution system. Thus it provides an understanding of how factors such as rms values and harmonics are modeled in *DTMS*. The methodology includes constitutive relationships involving current flow and heat generation. Temperature dependence of each of the circuit elements is also provided to aid in understanding thermal impact on electrical behavior. Chapters 4 and 5 build on the creation of circuit elements discussed in this chapter to model and simulate systems and subsystems of the electrical distribution system aboard a future all-electric ship. Chapter 6 discusses heat flow from the electrical components and how thermal feedback is provided to the electrical models discussed in this chapter to alter the electrical parameters and ultimately to alter performance of the electrical distribution system.

4. Power Converters

The power converter modules (PCM), switchboards, and cabling represent the bulk of the electrical distribution components contained in each zone of an all-electric ship (AES). The family of PCMs consists of the DC-DC converter module (PCM-1), an inverter module (PCM-2) and a rectifier module (PCM-4). Internal to these modules, the losses consist of heat dissipation via Joule heating in resistive elements, switching losses, core and winding losses of transformers, and reactive power losses through energy storage in inductors and capacitors.

The goal of this electro-thermal co-simulation is to model the bulk heat load associated with the electrical power conversion network. To this end, the *DTMS* converter models described in this chapter utilize an averaged switch model with rms values for voltage and current to minimize simulation run time while maintaining reasonable fidelity in representation of the bulk heat load generated by the onboard electrical distribution system. Heat generation and heat flux due to resistive and switching losses and associated thermal management strategies are discussed in Chapter 6. Temperature feedback is employed within the converter models to update the properties of converter elements.

4.1 PCM-4 Modeling

Electrical Configuration

The PCM-4 is a transformer and a 12-pulse rectifier that steps down three-phase ac voltage from 4,160 Vrms to 500 Vrms at a frequency of 60 Hz, which is then converted to dc power at 1,000 Vrms by the rectifier bridge (boost converter) [16]. A wye transformer with a rating of 3 MVA performs the three-phase ac voltage step down. A snubber consisting of a resistor and capacitor is used to mitigate voltage transients. The resistor and capacitor have a resistance of 50 ohms and 60 μFarads, respectively. Figure 4.1 depicts the electrical configuration of the PCM-4.

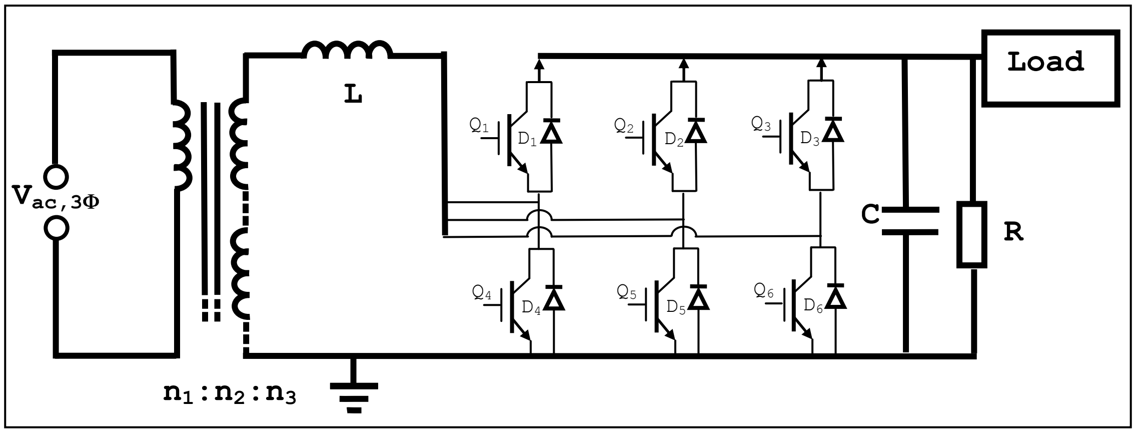


Figure 4.1: Electrical configuration of PCM-4.

Averaged Switch Model

The realization of an electro-thermal co-simulation only requires that a converter model contain elements sufficient to represent the dynamics of electrical power conversion. From a thermal perspective, it is not necessary to model the complex switching aspects of a converter as long as resistive and switching heat losses are taken into account. Thus, an averaged switch model is appropriate for simulating the dynamics of power conversion while simultaneously accounting for energy lost during the process.

To convert to an averaged switch model, the PCM-4 requires only two simplifications. Obviously, the first simplification is to use rms values of voltage rather than the actual three-phase sinusoidal voltage source. The use of rms values is illustrated in Figure 4.2, where the voltage source is labeled Vac,rms. The second simplification is to transform the complex switching elements into simple linear circuit elements. The diodes and transistors that compose the 12-pulse rectifier bridge are converted into current sources (in actuality they are losses) and resistances, respectively. In this manner, the model is able to predict the low frequency response of the system, ignore the high frequency harmonics, and still accurately represent the behavior of the system [7]. In the averaged switch model, the switching losses of the diode are averaged over one switching period. The current losses are composed of a recovered charge, Q_r, divided by the switching period, T_s, and reverse recovery time, t_r, over the switching period multiplied by the current in the ac circuit, i_2. The power losses associated with the transistors are modeled with a resistor that simulates the on-resistance of the transistor, R_{on}. This resistance accounts for the percentage of time the transistor conducts during one switching period.

Finally, the PCM-4 model must employ a new method to convert the ac power to dc power, due to the switching devices being modeled as current sources and an on-resistance. To do so, the model assumes that the act of converting ac power to dc power is an ideal process, where losses are later accounted for in the dc circuit. The model utilizes a loss-free resistor, R_e (an emulated resistance), to simulate this ideal process. Figure 4.2 illustrates the conversion of ac power, P_{total}, to a dc voltage, V_{dc}, via the loss-free resistor, R_e. The following expression describes the conversion from ac power to dc power.

$$P_{ac} = \frac{V_{dc,rms}^2}{R_e(t)} = P_{dc} \tag{4.1}$$

where $P_{ac} = P_{dc} = P_{total}$ across the rectifier bridge.

Figure 4.2 is a representation of the modeling techniques previously described to simulate the PCM-4 as an averaged switch model employing rms values and the loss-free resistor.

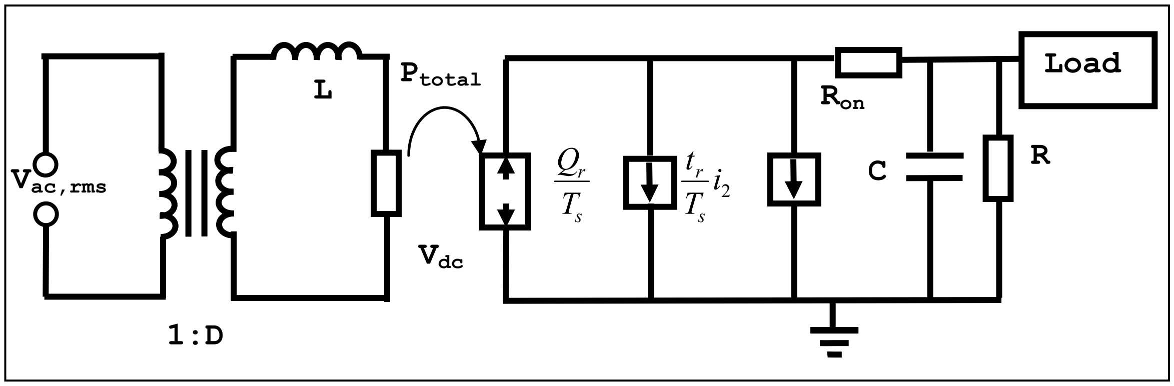


Figure 4.2: Averaged switch model of the PCM-4. [7]

Dynamic Response

To demonstrate the modeling capabilities in *DTMS* with respect to the PCM-4, an event was simulated to exhibit the behavior of the PCM-4 during a step load from a part load to full rated power. Table 4.1 lists electrical parameters for the PCM-4.

Table 4.1: PCM-4 parameters.

PCM-4 Electrical Parameters	
Primary Winding Voltage	4160 $V_{ac,rms}$
Secondary Winding Voltage	500 $V_{ac,rms}$
Snubber Capacitance	60 μFarads
Snubber Resistance	50 ohms
Rated Power	3 MVA

DTMS provides the capability to simulate a step load through a *setEvent* function contained in the *Resistor* model. The function allows the user to input the desired event time and desired resistance to simulate a step load. Currently in the *DTMS* framework, each PCM model must be provided a reasonable input value for the flux linkage (analog to momentum in the mechanical domain) of the inductor before running the simulation. Table 4.2 lists user inputs regarding the load, event time, and PCM-4 initial conditions.

Table 4.2: PCM-4 part load to full load simulation user inputs.

PCM-4 Simulation User Inputs	
Initial Load Resistance (t=0)	2.5 Ohms
Event Load Resistance	0.5 Ohms
Event Time	5 sec
Inlet Voltage	4160 $V_{ac,rms}$
Power Factor	0.96
Initial Inductor Flux Linkage (t=0)	0.042 weber-turns
Simulation Time Step	0.0005 sec

Figure 4.3shows the efficiency of the PCM-4 as simulated in *DTMS*. The step load occurs at 5 seconds and introduces a load at the rated capacity of 3 MVA. In the *DTMS* Framework, the transient recovery time from the step load in this simulation is approximately 0.015 seconds. The run time for the simulation is 1.6 seconds. The converter reaches an efficiency of about 96.2% at full load.

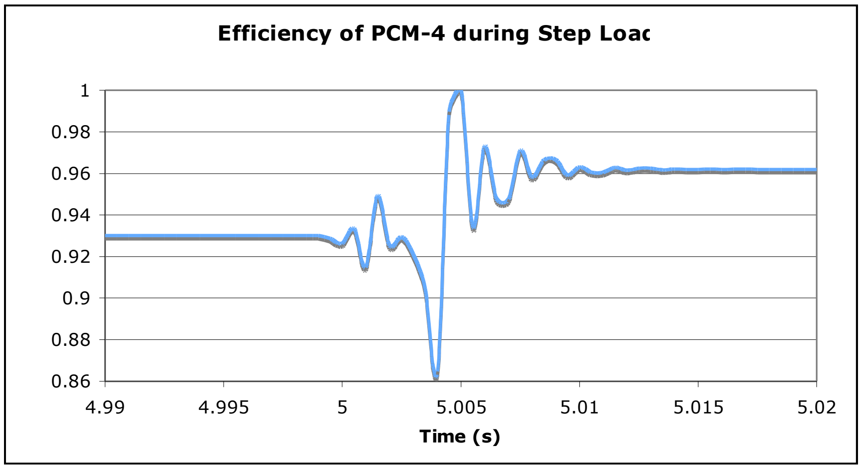


Figure 4.3: Efficiency of the PCM-4 during step loading, partial load to full load.

4.2 PCM-1 Modeling

Electrical Configuration

The PCM-1 is a DC-DC converter module that converts 1000Vdc to 800 and 775Vdc, nominally [16]. Efficiencies are not generally available, but typical values for these devices range from 96 to 98 percent [18]. The converter is a typical buck converter consisting of a gate turn-on thyristor, diode, and a snubber capacitor and resistor to mitigate voltage transients [7]. Figure 4.4 displays the electrical configuration of the PCM-1.

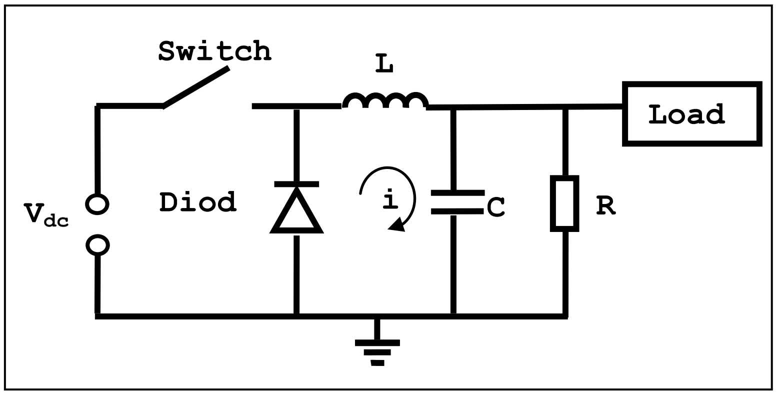


Figure 4.4: Electrical configuration of PCM-1.

Averaged Switch Model

Once again the averaged switch model is employed to quantify switching losses for the PCM-1. As with the PCM-4, the switching elements (transistors and diodes) are converted into time-invariant voltage and current sources. In a DC-DC converter, the output voltage, V_{out}, and output current, i_{out}, are functions of the incoming voltage, V_{in}, and incoming current, i_{in}, and the converter ratio as seen in Equations 4.2 and 4.3. In the case of the PCM-1, this ratio is called the duty cycle, *D*.

$$V_{out} = DV_{in} \tag{4.2}$$

$$i_{in} = Di_{out} \tag{4.3}$$

These expressions are characteristic of an ideal transformer. Thus, the switching elements can now be modeled as an ideal transformer with a resistor and current sources to represent the switching losses [7].

Figure 4.5 illustrates the averaged switch model electrical configuration for the PCM-1 with the switches represented as an ideal transformer and with the switching losses represented as resistive and current losses. Switching losses in the diode and transistor are found in equations 3.58 and 3.60.

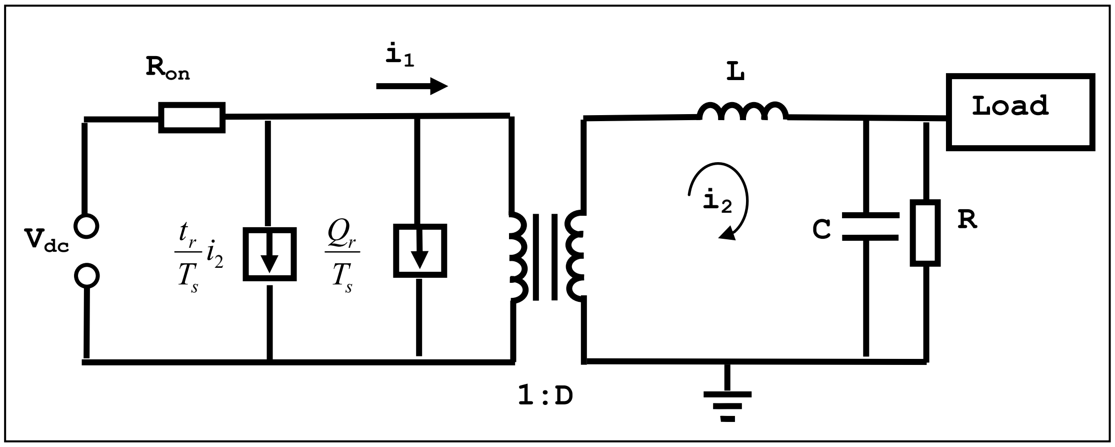


Figure 4.5: Averaged Switch Model of PCM-1.

Dynamic Response

To test the model of the PCM-1 dynamically, a step load was given to the averaged switch model. Table 4.3 lists the electrical parameters for the PCM-1.

Table 4.3: Known electrical parameters of PCM-1.

PCM-1: Electrical Parameters	
Duty Cycle	0.8 (unitless)
Inductance	0.05 Henries
Capacitance	0.05 Farads
Load Resistance	5000 Ohms

Individual Ship Service Converter Modules that comprise the PCM-1 have ratings of 100 kVA. To demonstrate the modeling capabilities in *DTMS* with respect to the PCM-1, an event was simulated to exhibit the behavior of the DC-DC converter during a step load from a part load to full rated power. Table 4.4 lists the user inputs regarding the load, event time, and initial conditions.

Table 4.4: PCM-1 part load to full load simulation user inputs.

PCM-1 Simulation User Inputs	
Initial Load Resistance (t=0)	100 Ohms
Event Load Resistance	6 Ohms
Event Time	5 sec
Inlet Voltage	1000 $V_{dc,rms}$
Power Factor	0.96
Initial Inductor Flux Linkage (t=0)	0.28 weber-turns
Simulation Time Step	0.0005 sec

Figure 4.6 shows the efficiency of PCM-1 as simulated in *DTMS*. The step load occurs at 5 seconds and introduces a load at the rated capacity of 100 kVA. Transient recovery time for the PCM-1 is 0.1 seconds [18]. In the *DTMS* Framework, the transient recovery time for the step load in this simulation is approximately 0.035 seconds. The run time for the simulation is 0.9 seconds.

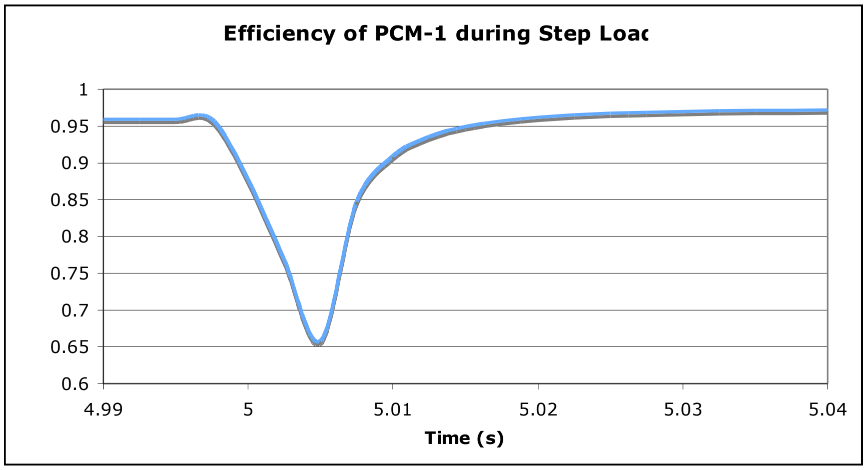


Figure 4.6: Efficiency of PCM-1 during step loading, partial load to full load.

The converter reaches an efficiency of about 97.2% when operating at full power. This is within the bounds of typical converter efficiency. The PCM-1 model can be adjusted to more accurately simulate converter efficiency, and therefore heat dissipation, when more details are known regarding actual PCM-1 data.

4.3 PCM-2 Modeling

Electrical Configuration

The PCM-2 is a dc to ac inverter that converts 800 Vdc from the port and starboard buses to 450 ac, three-phase voltage and supplies this power to inductive and resistive loads within each ship zone [16]. A snubber consisting of a capacitor and resistor is utilized to suppress voltage transients. The resistance and capacitance are nominally 500 ohms and 0.5 μFarads, respectively. Figure 4.7 displays the electrical configuration of the PCM-2.

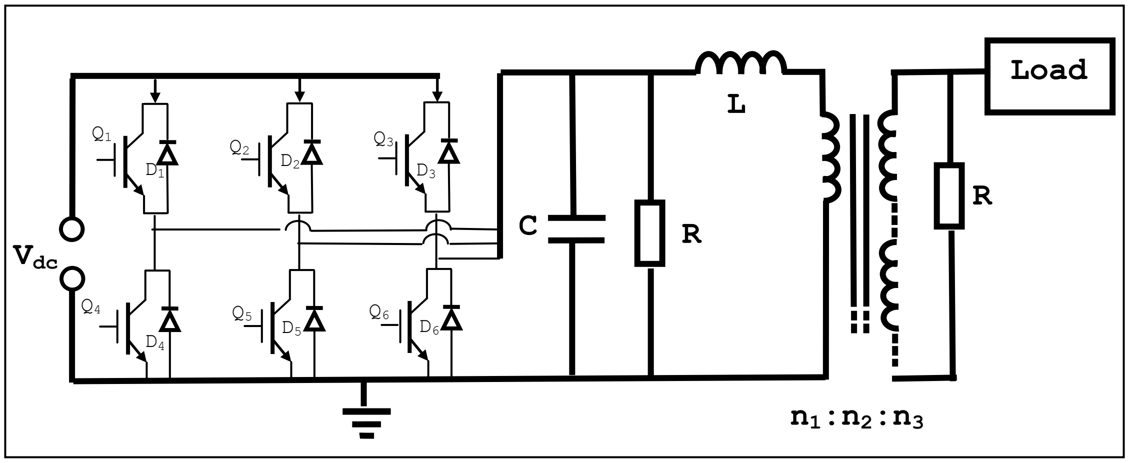


Figure 4.7: Electrical configuration of PCM2.

Averaged Switch Model

Again, *DTMS* utilizes the averaged switch method to model the PCM-2. Switching devices are simplified to account only for resistive and switching losses. Losses in the bridge consist of switching losses in the diodes and the on-resistance of the transistors. Switching losses in the diodes are accounted for by current losses associated with the diode reverse recovery time, t_r, and recovered charge, Q_r.

Once again, a loss free resistor is used to convert the total dc power, P_{total}, to bridge output voltage, $V_{ac,rms}$. The output voltage of the inverter bridge is computed as an rms value, $V_{ac,rms}$, using the following expression:

$$V_{ac,rms} = \sqrt{P_{dc} \times R_e(t)} \tag{4.4}$$

where $P_{dc} = P_{ac} = P_{total}$ across the inverter bridge.

Figure 4.8 shows a representation of the averaged switch model of the PCM-2 based on the modeling techniques previously described.

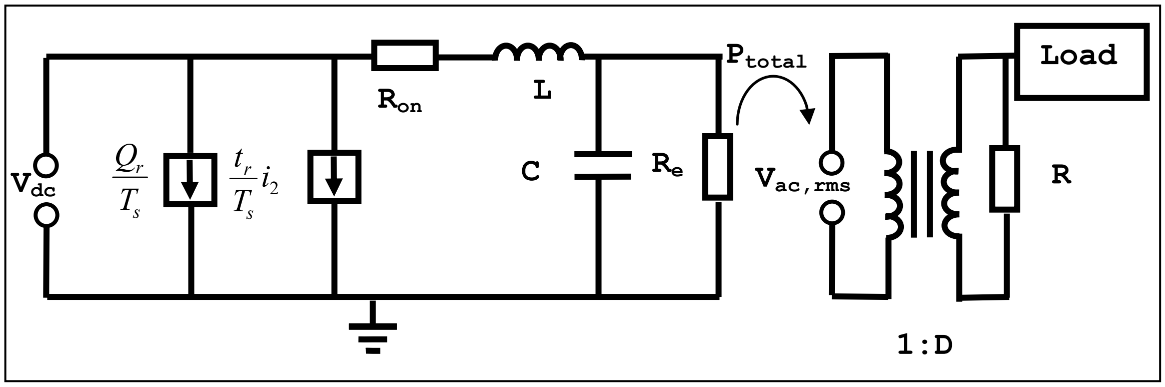


Figure 4.8: Averaged switch model of the PCM-2. [7]

Dynamic Response

To demonstrate the modeling capabilities in *DTMS* with respect to the PCM-2, an event was simulated to exhibit the behavior during a step load from part load to full capacity. Table 4.5 lists the electrical parameters used in this simulation. The inverter and transformer of the PCM-2 have a rating of 500 kVA.

Table 4.5: PCM-2 parameters.

PCM-2 Electrical Parameters	
Primary Winding Voltage	800 $V_{ac,rms}$
Secondary Winding Voltage	450 $V_{ac,rms}$
Inductor Inductance	0.02 Henries
Snubber Capacitance	0.05 μFarads
Snubber Resistance	500 ohms
Rated Power	500kVA

Table 4.6 lists the user inputs for load, event time, and initial conditions used in this simulation.

Table 4.6: PCM-2 part load to full load simulation user inputs.

PCM-2 Simulation User Inputs	
Initial Load Resistance (t=0)	24 Ohms
Event Load Resistance	0.4 Ohms
Event Time	5 sec
Inlet Voltage	800 $V_{dc,rms}$
Power Factor	0.96
Initial Inductor Flux Linkage (t=0)	12 weber-turns

Figure 4.9 shows the efficiency behavior of the PCM-2 when simulated as described above. The step load occurs at 5 seconds and introduces the full capacity load of 500kVA. The empirical value for transient recovery time of the PCM-2 is 0.3 seconds [18]. *DTMS* predicts the transient recovery time to be approximately 0.3 seconds. The converter reaches an efficiency of about 95.8%, in close agreement with the listed efficiency of 95.7% by SatCon [18].

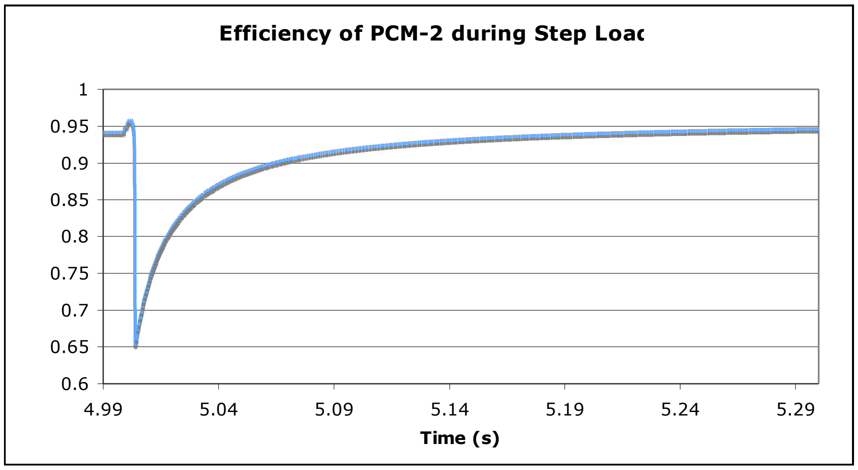


Figure 4.9: Efficiency of the PCM-4 during step loading, partial load to full load.

4.4 Conclusion

This chapter discusses modeling of the power converters as averaged switch models. Basic modeling strategies include the averaging of switching losses and the use of the loss-free resistor and the modulation index to convert ac and dc power. These strategies are solutions to the challenge of simulating the electrical distribution system with reasonable fidelity while minimizing computational run time. The performance of each converter is modeled, simulated, and validated either against previous work within the Electric Ship Research and Development Consortium or available industry data.

5. Zonal Electrical Distribution System

In contemporary warship design, geographic subdivisions (or zones) of the ship are intended to control damage in the event of a Design Threat Outcome (DTO). In line with this approach and to ensure redundancy and the ability to reconfigure in the event of damage to shipboard systems during combat, the Navy has divided electrical systems of the all-electric ship (AES) into zones and called this the Zonal Electrical Distribution System (ZEDS). These zones are designed to enhance survivability, maximize recoverability, and minimize susceptibility and vulnerability. In the event of a DTO, a zone may lose functionality but the ZEDS will attempt to prevent propagation of damage by allowing for adjacent zones to maintain ship operations and reconfiguring systems to maintain power to those systems requiring uninterrupted power.

The Navy wishes to develop a modeling and simulation environment capable of examining the behavior of various configurations of the electrical distribution system during dynamic situations. Thus, this chapter demonstrates the capabilities of *DTMS* to simulate the thermal-electrical characteristics of a notional zone of the ZEDS.

5.1 ZEDS Configuration

Once fuel is converted to mechanical power and then to electrical power in the Power Generation Modules (PGM), this power is distributed to the zonal Power Conversion Modules (PCM). The PCMs convert electrical power to the desired power quality, which is then distributed to Power Loads (PL) within the zone. Zones may also contain Energy Storage (ES) as well as other PGMs, e.g., solar cells, fuel cells, Diesel generators, etc.

A typical configuration consists of a zonal PCM-4 along with sets of PCM-1 and PCM-2 that convert and distribute the required ac and dc power to zonal loads. The PCM-4 accepts three-phase, ac power from the turbo-generator and converts this electricity to dc power. This power is then distributed to the starboard and port electrical buses within each respective zone. The PCM-1, made up of a number of Ship Service Converter Modules (SSCM), steps this dc power down from 1000 Vdc to nominally 800 Vdc for distribution either to dc loads or for further conversion to low voltage ac power. The number of SSCM, which have nominal power ratings of 100kVA, within a PCM-1 depends on the power requirements of the particular zone. The PCM-2, which is made up of a number of Ship Service Inverter Modules (SSIM), converts dc power to nominally 450 Vac. As with the PCM-1, the number of SSIM within a PCM-2 depends on the power requirements within the zone. Figure 5.1 shows a notional ZEDS configuration for the AES.

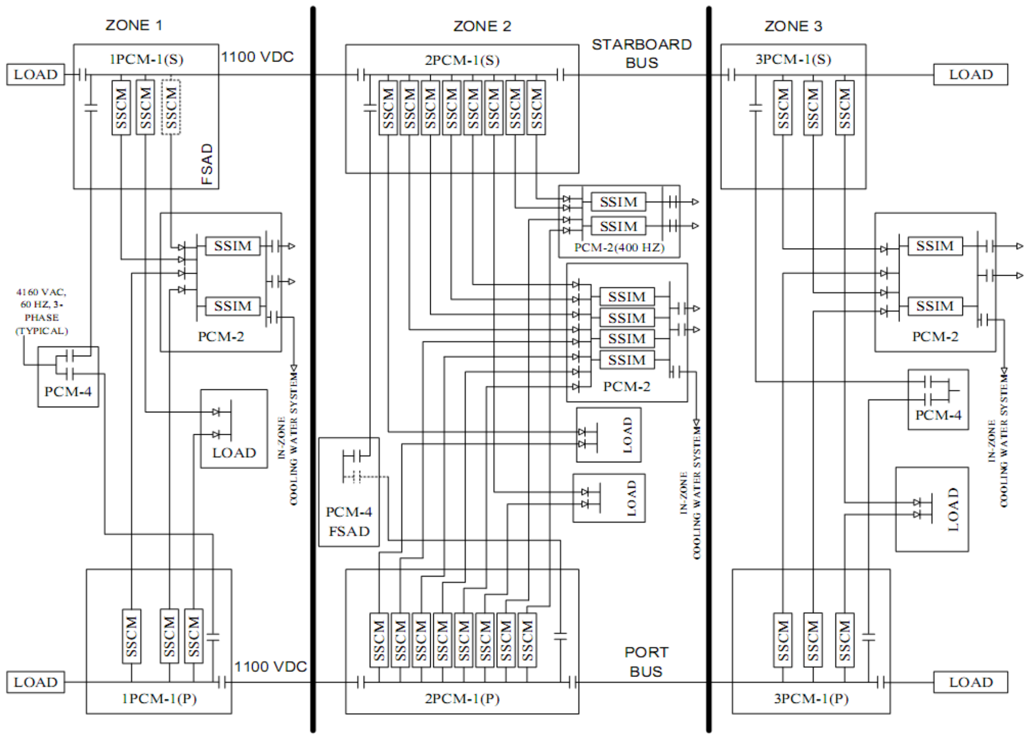


Figure 5.1: Notional Configuration of ZEDS for AES. [27]

5.2 *ZEDS* Benchmark

To test the *DTMS* electrical distribution framework, a simulation was conducted to validate against results from the Naval Combat Survivability (NCS) Testbed. Figure 5.2 from [30] illustrates the ZEDS configuration posed as the benchmark for future modeling and simulation environments.

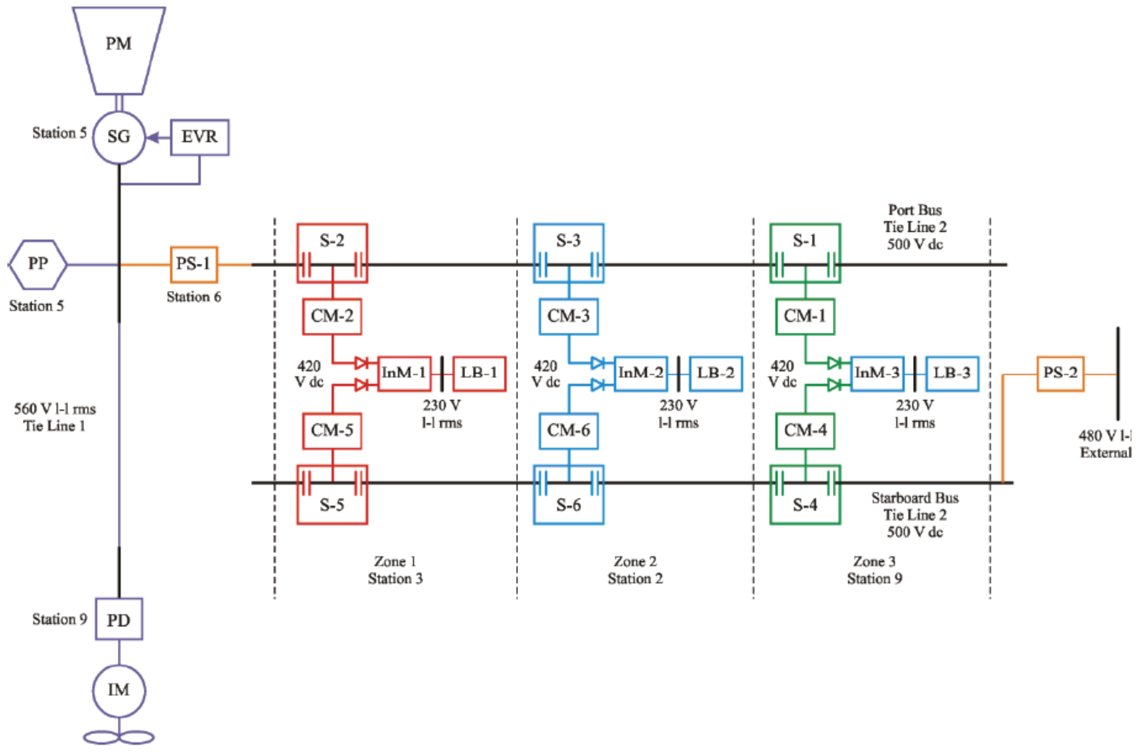


Figure 5.2: NCS Testbed configuration for benchmark simulation of ZEDS [30].

The NCS Testbed was developed in Matlab/Simulink and simulates ZEDS by either using detailed waveforms or averaged value waveforms. The former is time and effort intensive, while the latter is numerically much faster [30]. The configuration consists of two power supplies (PS-1 and PS-2) supplying 500 and 480 Vdc, respectively, to the starboard and port ship converter modules. The converter modules, CM-2, CM-3, and CM-1 in Zones 1, 2 and, 3, respectively, convert 500 Vdc from the port bus to 420 Vdc. The converter modules CM-5, CM-6, and CM-4 in Zones 1, 2, and 3, respectively, convert 480 Vdc from the starboard bus to 420 Vdc. The converters within their respective zones operate in parallel with their counterpart on either the starboard or port side to supply 420 Vdc to the inverter modules in each zone. The inverter modules, InM-

1, InM-2, and InM-3 convert 420 Vdc to 230 Vac,rms line-to-line and supply this power to load banks, LB-1, LB-2, and LB-3 within their respective zones.

The NCS Testbed benchmark electrical simulation runs for 10 seconds. At 6 seconds, the converters are operating at steady state supplying 1 kW to each of the three zonal loads. Table 5.1 lists the step loads applied to each zone from 6 seconds to the end of the simulation.

Table 5.1: NCS Testbed benchmark simulation step loads [30].

NCS Testbed Benchmark Simulation	
Time (sec)	Load
6	LB-1 to 2.5 kW
7	LB-1 to 5 kW
7	LB-2 to 2.5 kW
8	LB-2 to 5 kW
8	LB-3 to 2.5 kW
9	LB-3 to 5 kW

The following section compares NCS Testbed results with results produced using *DTMS*.

5.3 *DTMS* Benchmark Simulation

To recreate the NCS Testbed benchmark simulation, the main file *benchmark.cpp* was created. The simulation consists of two instances of *PCM4* to represent the two PS, six instances of *PCM1* to represent the six CM, three instances of *PCM2* to represent the three InM, two instances of *ElectricalEffortModel* to represent the ac source and ground, and 15 instances of *Resistor* to represent cables and the three LB. Appendix B contains the main file for the benchmark simulation. Table 5.2 lists the input parameters for the simulation. The voltage source values as well as loads match the NCS Benchmark values. The other values in the table reflect the modeling strategies of the electrical framework in *DTMS*.

Table 5.2: benchmark.cpp input parameters.

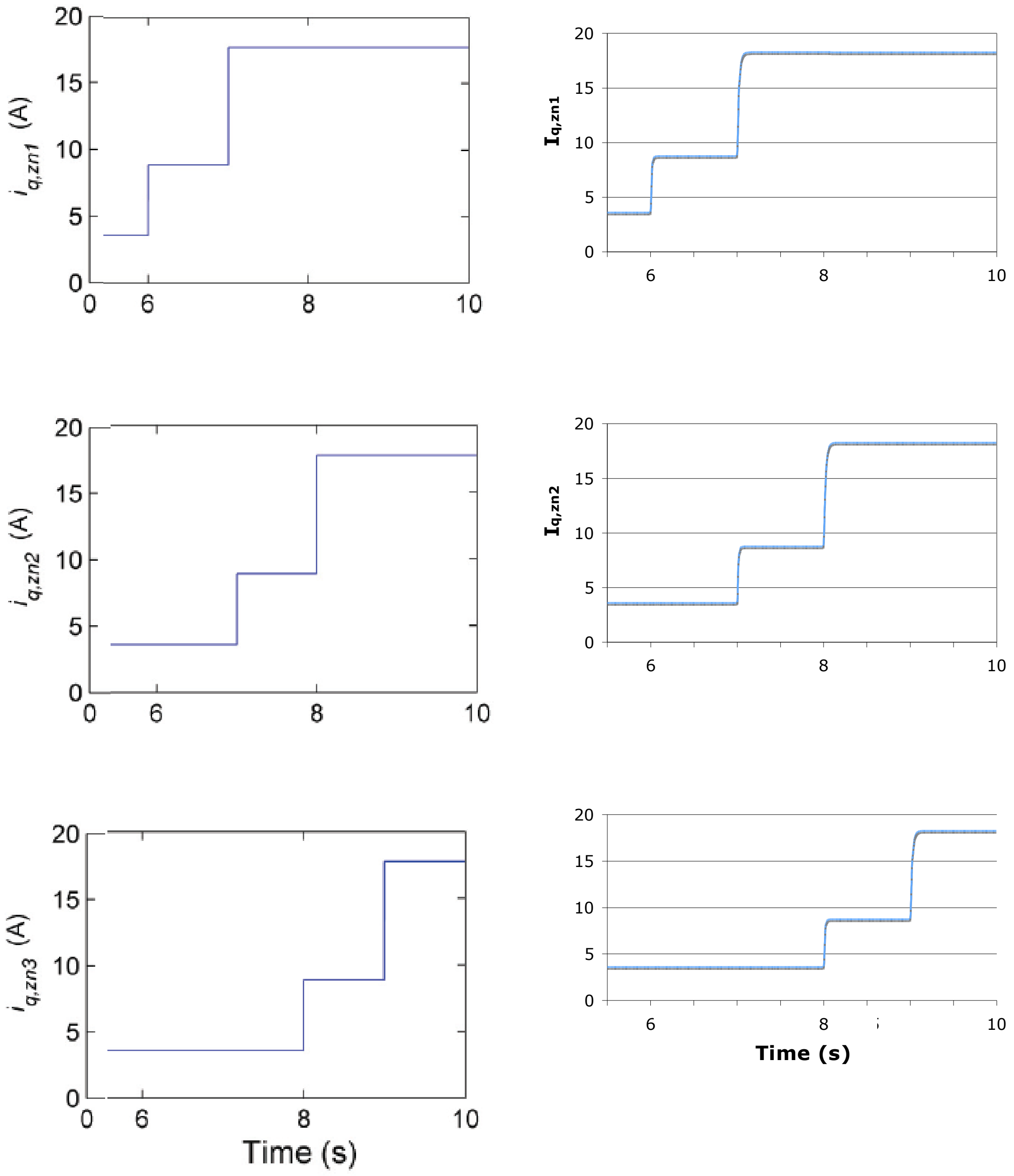
benchmark.cpp Input Parameters			
Port ac voltage source	560 Vdc		
Starboard ac voltage source	480 Vdc		
LB1 load resistances	56 ohms	23 ohms	11 ohms
LB2 load resistances	56 ohms	23 ohms	11 ohms
LB2 load resistances	56 ohms	23 ohms	11 ohms
PCM-1 initial flux linkage	0.08 weber-turns		
PCM-2 initial flux linkage	0.3 weber-turns		
PCM-1 duty cycle	0.84		
PCM-2 initial loss free resistance	157 ohms		

Both the NCS benchmark simulation and the *DTMS* framework use average value models (average switch models) to model the power electronic switching transients. The NCS benchmark simplifies the inverter as a capacitor in parallel with a power load.

Results

The results of the *DTMS* benchmark simulation are comparable to the benchmark simulation set forth in [30]. Although exact transient values are not available, the trends demonstrated by [30] for the transient nature of the converter modules correspond well with the trends demonstrated by *DTMS* with few exceptions. In Figures 5.3, 5.4, and 5.5, the results of [30] are displayed to the left while the equivalent results using *DTMS* are displayed to the right for comparison.

Figure 5.3 compares the ac current through the inverter modules (InM in [30], *PCM2* in *DTMS*) through the transient period from 5.5 seconds to 10 seconds. The results of [30] do not show any discernable transience during step loading of the inverters; the current appears to jump immediately to its steady state value. The *DTMS* results show a brief transient period of approximately 0.05 seconds to reach the new steady state value between step loads. The magnitudes of the current waveforms in both simulations appear to be the same.



NCS Benchmark *DTMS*

Figure 5.3: Comparison between benchmark results of [30] and *DTMS* for ac current through inverter modules within zones 1 (zn3), 2 (zn2), and 3 (zn3).

Figure 5.4 compares the dc zonal voltage before inversion between the converter (CM in [30], *PCM2* in *DTMS*) and inverter modules through the transient period from 5.5

seconds to 10 seconds. The results of [30] show slight perturbations of the voltage during the step loading. The major discrepancy between [30] and the results of *DTMS* is the voltage droop during transience. The *DTMS* simulation shows a droop of approximately 100 V during the step loading of the respective inverters. This discrepancy is due to the fact that minimal controls are implemented in the converter models in *DTMS* while [30] uses the Optimization of a Time-Domain based Performance Metric (OTDPM) method to optimize control parameters and provide optimal voltage and control regulation. Electrical and thermal impacts of the transient behavior and controls are discussed in the conclusion of this chapter.

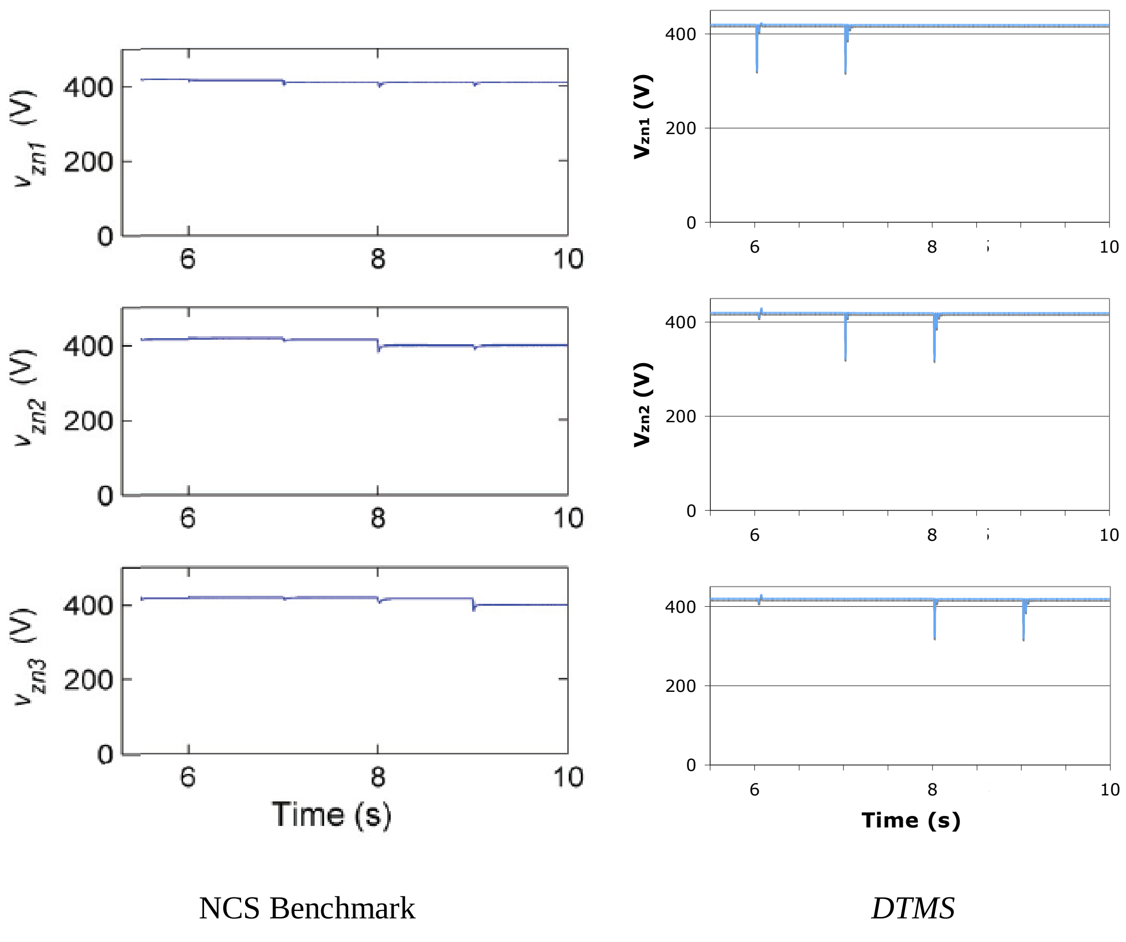
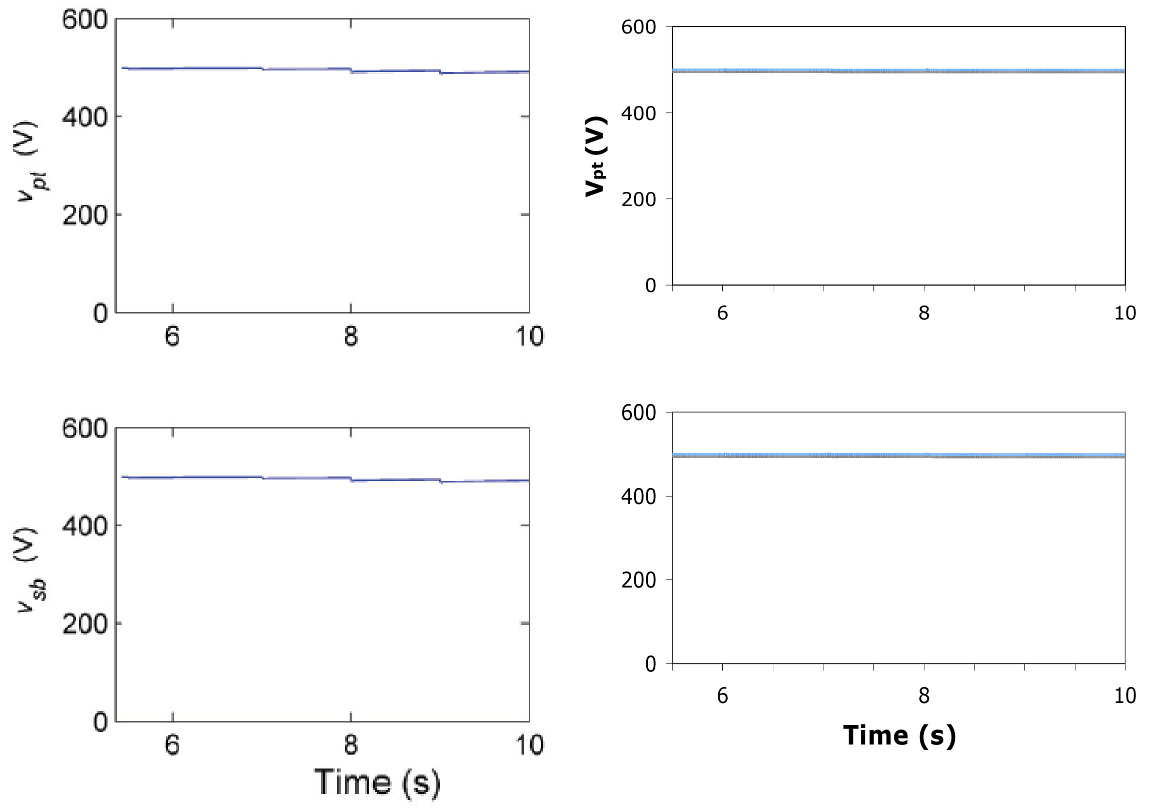


Figure 5.4: Comparison between benchmark results of [30] and *DTMS* for dc voltage of inverter modules within zones 1 (zn3), 2 (zn2), and 3 (zn3).

Figure 5.5 compares the port and starboard distribution voltages before dc-dc conversion through the transient period from 5.5 seconds to 10 seconds. The results from [30] show slight perturbations in the voltage during the step loading. The results of *DTMS* show similar perturbations, which are barely discernable over a transient time period of approximately 0.05 seconds. It appears that both models consistently regulate the port and starboard voltages against large voltage transient droop.



NCS Benchmark *DTMS*

Figure 5.5: Comparison between benchmark results of [30] and *DTMS* for port and starboard distributive voltages during dc-dc conversion.

5.4 Conclusion

The results of the *DTMS* simulation consistently match the benchmark results of [30] with the exception of the voltage droop during the transient step loading in Figure 5.4. Voltage, analogous to pressure in hydraulic energy systems, is the driving force behind current flow. When a load is applied to an electrical system, the intermediate voltages drop (droop) until the power source forces the voltage back to its steady state. As the voltages drive toward their steady state levels, the current too increases until sufficient power is provided to the load. The lack of any significant voltage droop in [30] explains the lack of current transience in the NCS benchmark graphs of Figure 5.3. Because the controls in the NCS Testbed damp out voltage spikes, the current is able to reach its steady state value faster than the *DTMS* simulations. However, the *DTMS* transient response still reflects the transient recovery time offered by industry, and discussed in Chapter 4 of this thesis [18]. Otherwise, the trends and magnitudes of the currents and voltages are consistent.

There are a few contributing factors to the observed discrepancies between the NCS Testbed and *DTMS*. The first difference is the modeling technique employed in the inverter models of NCS and *DTMS*. The NCS inverter and load banks are pared down to a simple capacitor and power load in parallel. Finally, the most important contributor to the difference in transient performance is the optimization controls method implemented in [30]. The effect of the transient behavior of the electrical system on the thermal aspects of *DTMS* is discussed in the remaining two chapters. These chapters discuss the relative importance of the heat losses during this short transient period.

6. Thermal Resistive Network and Concepts

With increasing volumetric heat generation in electronics due to increasingly faster and smaller semiconducting devices, cabinet cooling for Power Converter Modules (PCM) cabinets has become a priority. While the electrical distribution network has been completed and tested, the thermal management overlay has yet to be addressed. To this end, a framework must be constructed that links the heat dissipation from the electrical network to the fluid flow network previously constructed in *DTMS*. To do so, a thermal resistive network of heat flow was constructed. Figure 6.1 shows the thermal resistive network for a bay of electronics including feedback from the electrical and thermal management flow networks.

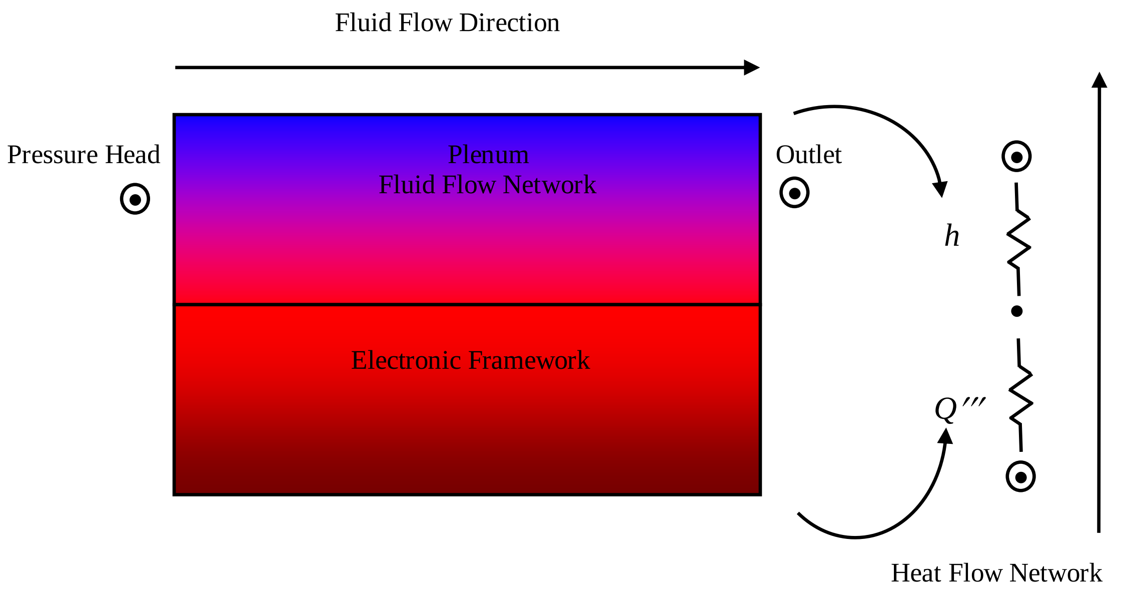


Figure 6.1: Heat flow from heat generating and energy storage electronics to plenum of cabinet.

This chapter addresses the creation of elements associated with heat generation, storage, and flow from the electronics to the plenum of air within each bay of the PCM electronic cabinets. These elements include a heat and energy storage element (the heat producing electronics), extended surface elements (fins protruding from an electronics surface to enhance heat transfer), and a generic convective/conductive medium (air flow/heat sink). In addition to the heat flow models, *DTMS* classes were created to link the heat exchange between the thermal resistive heat flow models and the fluid flow models of the existing thermal management network. These classes model feedback between the fluid and heat flow networks and involve the heat flux and heat transfer coefficient. This feedback allows the two flow networks to be connected although each network is solved separately in the network solver.

The first section of this chapter describes the models constructed in *DTMS* to simulate the heat flow through a thermal resistive network. The later sections identify cabinet cooling methods and their respective models in *DTMS*. Finally, the thermal resistive network models are validated via comparison to previous models of power conversion cabinets aboard the all-electric ship (AES).

6.1 Thermal Resistive Network Models

A thermal resistive network was created to simulate the heat flux from heat generating PCMs. The thermal resistive network directly links heat dissipated in the electrical network to the thermal management network which is analogous to the electrical network except that it utilizes efforts (temperature) and flows (heat flux). Flow models are used to provide the solvers with the flow while temperatures are adjusted in the solver until flow models converge. Thus, flow models must provide the flow equation as a function of the temperature:

$$q'' = f(\Delta T) \tag{6.1}$$

The network is one-dimensional and captures the essential transient characteristics of heat flow and energy storage.

TemperatureEffortModel

The class *TemperatureEffortModel* is utilized as the thermal effort node. It is analogous to pressure and voltage in the hydraulic and electrical domains, respectively. The model is derived from *ResistiveNetworkEffortModel* and contains the functions *setTemperature()* and *getTemperature().*

HeatFlowModel

The class *HeatFlowModel* is the base class for thermal resistive network flows. The model provides the functions to input geometry as well as the physical and thermal properties for heat transfer. It is derived from *ResistiveNetworkFlowModel.*

LinearHeatFlowModel

The class *LinearHeatFlowModel* provides the thermal resistive network with flow equations and flow derivatives to solve a linear heat flow network. The model is derived from both *HeatFlowModel* and Linear FlowModel. The former provides the derived model with geometrical and thermal properties while the latter provides the derived model with the flow network solving capabilities. These models do not contain energy storage nor do they have the capability for heat generation. The following equations provide the convective and conductive heat flux, respectively, in *LinearHeatFlowModel,*:

$$q'' = h(\Delta T) \tag{6.2}$$

$$q'' = \frac{k}{L}(\Delta T) \tag{6.3}$$

where *h* is the convection heat transfer coefficient, *k* is thermal conductivity, *L* is the characteristic length through a solid surface, and *ΔT* is the temperature difference for heat transfer. The model can be employed in either a conductive or convective heat transfer environment depending on user input. The appropriate circumstance is selected based on the function call to *setConvectionCoefficient* or *setThermalConductivity* , respectively.

FinArrayModel

The class *FinArrayModel* is utilized to simulate various arrays of fins on a flat plate. The model is derived from *LinearHeatFlowModel* and currently provides the user with three fin types: rectangular, triangular, and parabolic. The model provides the functions necessary to input fin geometry and thermal properties. The heat flow equation for an array of fins is a function of fin geometry, surface area, and fin efficiency. The governing equation combines the convective heat transfer through the fins in the first term and convective heat transfer off the bare surface in the second term as shown in Equation 6.4.

$$q = N\eta_f h A_f (\Delta T) + h A_b (\Delta T) \tag{6.4}$$

Here N is the number of fins, η_f is the fin efficiency, h is the convection coefficient, A_f is the total area of the fins, and A_b is the unfinned base area. These terms can be written in terms of the total surface area, A_t, to find the total heat flux:

$$q'' = h\left[1 - \frac{NA_f}{A_t}(1-\eta_f)\right](\Delta T) \tag{6.5}$$

The user may choose rectangular, triangular, or parabolic fin arrays by calling the functions *setRectangularFins*, *setTriangularFins*, and *setParabolicFins*, respectively. Once a fin array is selected, the instance of *FinArrayModel* calculates the efficiency of the respective fin array by interpolating Figure 3.18 of [12]. The closed-form expression for the efficiency in this figure is linearized to allow for this interpolation. The efficiencies, η_f, are functions of fin geometry, material properties, and the convection coefficient, i.e.,

$$\eta_f = f\left(L_c^{3/2}(h/kA_p)^{1/2})\right) \qquad 6.6$$

where L_c is a corrected fin length, h is the convection coefficient, k is the thermal conductivity, A_p is the profile area, and the efficiency is a percentage from 0 to 100%. Users of *DTMS* can easily create additional fin arrays for various fin arrays in a like fashion. The convection coefficient is set using the function *setConvectionCoefficient*. If the model is connected to an instance of *Pipe*, the function *setConvectionModel* is utilized in the main file. If a convection model is set in the main file, the *FinArrayModel* internally obtains the Reynolds number from the connected *Pipe* model at every time step. The Nusselt number, Nu_D, is then calculated using the following correlation for flow around a cylinder:

$$Nu_D = 1.15\,\mathrm{Re}_D^{1/2}\,\mathrm{Pr}^{1/3} \qquad 6.7$$

where Pr is the Prandtl number. The heat transfer coefficient, h, is then calculated as follows:

$$h = \frac{Nu \cdot k}{D} \qquad 6.8$$

where *k* is thermal conductivity and *D* is the diameter. The *FinArrayModel* also sets the coefficient of friction in the instance of *Pipe*. The coefficient of friction, *f*, is calculated using the following correlation [29]:

$$f = 2.202e^{-5.457(\alpha/L)} \cdot \left[\frac{\pi}{4} - D\right]^{-2.814} \tag{6.9}$$

where α is the fin height, *L* is the fin area length, and *D* is the area fin-density, defined as

$$D = n^2 \frac{\pi}{4}\left(\frac{d}{L}\right)^2 \tag{6.10}$$

where *n* is the number of fins and *d* is the hydraulic diameter of the fin. Future *DTMS* developers might find a wrapper a more useful tool to connect the fluid flow and heat flow through fins.

InternalHeatGenerationModel

The class *InternalHeatGenerationModel* was created to satisfy the need for a model to simulate heat generation in electrical components and energy storage. The model is derived from *LinearHeatFlowModel*, although the heat flux within the model itself is nonlinear. The heat equation for a finite volume within a solid experiencing energy storage and energy generation in one spatial dimension is as follows:

$$\frac{1}{\alpha}\frac{\partial T}{\partial t} = \frac{\partial^2 T}{\partial x^2} + \frac{\dot{q}}{k} \tag{6.11}$$

with the thermal diffusivity defined as

$$\alpha = \frac{k}{\rho c} \tag{6.12}$$

where *k* is the thermal conductivity, ρ is the density, *c* is the specific heat, and $\dot{q}$ is the volumetric internal heat generation. Due to the spatial and time dependency in Equation 6.11, the heat equation is a partial differential equation requiring two boundary conditions and an initial condition. To avoid the necessity for this model to require knowledge of external models within its network (other than the two boundary conditions and a previous condition, e.g., the previous temperature), Equation 6.11 is solved numerically using finite difference approximations inside *InternalHeatGenerationModel.* Accordingly, Equation 6.11 is discretized in time and space, using subscripts *p* and and *m* respectively:

$$\frac{1}{\alpha}\frac{T_m^{p+1}-T_m^p}{\Delta t}=\frac{T_{m-1}^p-T_m^p}{(\Delta x)^2}+\frac{T_m^p-T_{m+1}^p}{(\Delta x)^2}+\frac{q}{k} \tag{6.13}$$

where *p+1* is the current time, *p* is the previous time, *m* is the discretized element, *m+1* is the adjoined discretized element in the positive x direction and *m-1* is the adjoined element in the negative x direction. Thus, the new temperature, T_m^{p+1}, of the discretized element *m* is a function of the previous temperature at that element as well as the adjoining temperatures at *m-1* and *m+1* at previous time *p*. The temperature of each discretized element can be found implicitly, and therefore this method is called forward-differencing [12]. Heat transfer at the surface of the *InternalHeatGenerationModel* may account for the presence of either a conductive or convective boundary condition. The following is the discretized heat equation for a finite volume with heat generation, energy storage, and heat transfer at the boundaries of the control volume:

$$\frac{1}{2\alpha}\frac{T_m^{p+1}-T_m^p}{\Delta t}=\frac{T_{m-1}^p-T_m^p}{(\Delta x)^2}+\frac{q}{2k}-\frac{q_{surf}}{k\Delta x} \tag{6.14}$$

where q_{surf} is the heat transfer at the surface. The factor of 2 in the first and third terms accounts for the fact that the control volume consists of half convection and half heat generating solid at the boundaries.

To check the validity of the model, Example 5.8 of [12] was recreated. The following schematic illustrates a plane wall fuel cell with internal heat generation that is convectively cooled.

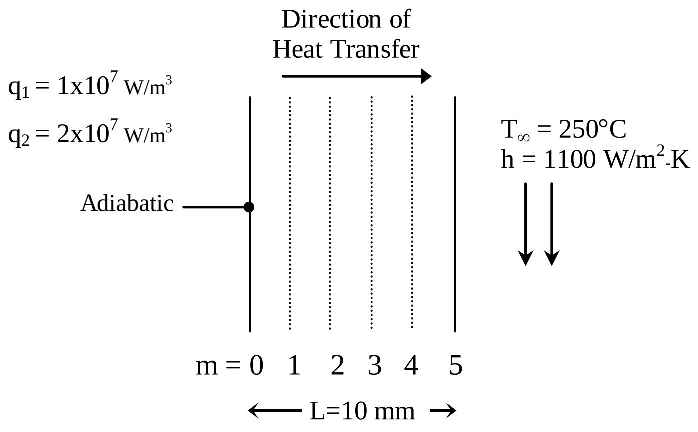


Figure 6.2: Example 5.8 of [12].

The fuel cell with a thickness of 10 mm generates 1x10^7 W/m^3 of heat while air convectively cools the surface with a convection coefficient of 1100 W/m^2-K and surrounding temperature of 250°C. From steady state (t=0), the heat generation doubles to 2x10^7 W/m^3. An adiabatic symmetry condition is imposed on the left face. Figure 6.3 illustrates the transient behavior of the surface temperature, at m=5, and the midplane temperature, at m=0. This solution has a relative error of 0.001% as compared with the solution in [12].

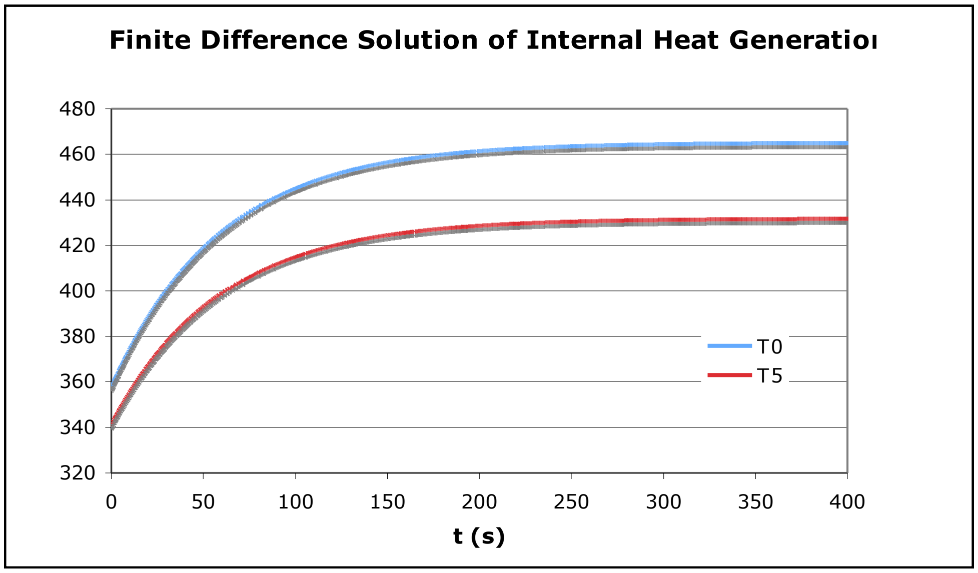


Figure 6.3: Finite difference solution of example 5.8 in [12].

Duct and Pipe Flow

Pipe and duct flow models are used to simulate the removal of heat from either a heat sink or fin array, respectively. Assuming steady flow in a pipe or duct the energy equation can be written as follows:

$$\Delta U + mg\Delta z + \frac{m\Delta v^2}{2} = \delta Q - \delta W \tag{6.15}$$

where ΔU is the change in internal energy, m, g, and z are mass, acceleration of gravity and position, v is velocity, δQ is the heat transfer into the system, and δW is the work done by the system. The second and third terms of equation 6.15 represent potential and kinetic energy respectively. By defining work into the system as work at the boundary,

$$\delta W = \Delta PV \tag{6.16}$$

where P is pressure and V is volume and defining the change in internal energy as the heat transfer into the subsystem plus energy lost to friction, E_{lost}

$$\Delta U = \delta Q + E_{lost} \tag{6.17}$$

and then substituting equations 6.16 and 6.17 into 6.15, the energy equation becomes

$$mg\Delta z + \frac{m\Delta v^2}{2} = E_{lost} - \Delta PV \tag{6.18}$$

Substituting

$$\rho = \frac{m}{V} \tag{6.19}$$

and

$$v = \frac{\dot{m}}{\rho A} \tag{6.20}$$

where *A* is the cross sectional flow area, into Equation 6.18 and assuming incompressible flow, the mass flow rate can be expressed as

$$\dot{m} = \sqrt{\frac{2\rho}{\frac{1}{A_{in}^2} - \frac{1}{A_{out}^2}}} \cdot \sqrt{\Delta P + \rho(g\Delta z + e_{lost)}} \tag{6.21}$$

Because these equations are the modeling strategy currently employed in *DTMS* class *Pipe*, this class is suitable for modeling both a pipe and duct, or specifically the plenum space surrounding electronic equipment in the PCM cabinets.

6.2 Air-Based Forced Convection

The most common method of removing heat from heat generating electronic equipment is forced convection. This heat transfer may be augmented with extended surfaces on the electronics. Cooling fans further enhance heat transfer by forcing ambient or cooled air through the plenum surrounding the electronics.

In *DTMS*, the architecture of a cabinet bay of electronics with forced convection consists of two networks: a fluid flow network and a heat flow network. The fluid flow network consists of an inlet source with a specified temperature and pressure, a blower to increase the pressure head, a mid pressure node, a plenum, and an outlet with specified pressure. The heat flow network consists of an internal heat generation model, a boundary temperature node, a convection model, and a boundary temperature node to model the temperature of the plenum. Figure 6.4 illustrates these two networks as modeled in *DTMS*.

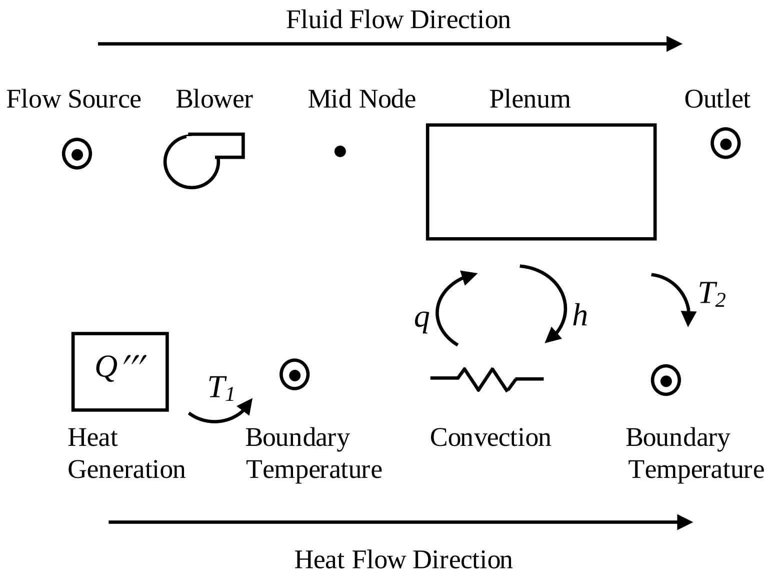


Figure 6.4: Cabinet bay model with forced convection consisting of fluid and heat flow networks.

Figure 6.4 indicates four instances of feedback in the model. The heat generation model sets the temperature, T_1, of its connected boundary node. This function is already inherent to the class *InternalHeatGenerationModel*. The feedback between the plenum, convection, and boundary temperature models are not inherent to their respective classes. The plenum receives heat from the convection model while setting the convection coefficient, *h*, in the convection model and the temperature, T_2, at the downstream boundary. To incorporate this feedback into *DTMS* while maintaining separately solved flow models, the group model *PlenumHeatExchanger* was created.

PlenumHeatExchanger

The class *PlenumHeatExchanger* is a wrapper consisting of an instance of *LinearHeatFlowModel, Pipe,* and *TemperatureEffortModel*. The *LinearHeatFlowModel* is used to simulate convective heat flow in a plenum or duct, while the *Pipe* model is used to simulate flow through the plenum or duct. The working fluid is an instance of *CaloricallyPerfectAir*. The models are intimately connected via the heat transfer coefficient within the plenum, but the fluid flow in the duct and the heat flow are resolved separately by the network solver. Thus, a wrapper is necessary to provide interaction between the heat and fluid flow models.

Each model solves for its own flow characteristics each time step. Once the flows (both fluid flow and heat flow) of the plenum are resolved during a time step, the *PlenumHeatExchanger* model calculates the convection coefficient of the plenum flow based on the then current Reynolds and Nusselt numbers. The Reynolds number is calculated in the base class *Pipe* according to the following:

$$\mathrm{Re} = \frac{\dot{m} D_h}{\mu A_c} \tag{6.22}$$

where $\dot{m}$ is the mass flow rate, D_h is the hydraulic diameter, μ is the dynamic viscosity, and A_c is the cross-sectional area. For a fully wetted rectangular duct, the hydraulic diameter is

$$D_h = \frac{2LW}{L+W} \tag{6.23}$$

where *L* is the length and *W* is the width of the plenum. Assuming the flow in the plenum to be turbulent [2], Incropera and DeWitt [12] suggests using the Dittus-Boelter equation to solve for the Nusselt number in the duct.

$$Nu_D = 0.023\,\mathrm{Re}_D^{4/5}\,\mathrm{Pr}^n \tag{6.24}$$

where the Prandtl number, Pr ≅ 0.71 for air in the plenum, and *n* equals 0.4 for a surface temperature greater than the mean temperature of the air, i.e., heating. The convection coefficient is then given by:

$$h = \frac{Nu \cdot k}{D_h} \tag{6.25}$$

where *k* is the thermal conductivity of the air.

The *PlenumHeatExchanger* then updates the convection coefficient using an instance of the *LinearHeatFlowModel*. The *LinearHeatFlowModel* updates the enthalpy of the air in the *Pipe* as a function of the heat input. Finally, the temperature of the plenum flow is updated using an instance of *TemperatureEffortModel.* The *TemperatureEffortModel* is set as an independent effort, thus providing the temperature as a boundary condition for solving the heat flow network.

6.3 Liquid-Based Forced Convection

Liquid forced convection is a more appealing solution to increasing heat flux in electronic devices. The appeal is due to the greater heat capacity of liquids as opposed to air. A common methodology for removal of heat is to attach a heat sink to the base plate of the electronic device and force water through a conduit in direct contact with the heat sink. Freshwater is typically used because of its non-corrosive nature and fact that the water can be treated (de-ionized) to ensure electrical isolation [28].

In *DTMS*, the architecture of a cabinet bay of electronics and a connected heat sink with liquid forced cooling again is made up of two networks: a fluid flow network and a heat flow network. The fluid flow network consists of an inlet source with a specified temperature and pressure, a pump to increase the pressure head, a mid pressure node, a pipe, and an outlet with specified pressure. The heat flow network consists of an internal heat generation model, a boundary temperature node, a conduction model, and a boundary temperature node to model the temperature of the plenum. Figure 6.5 illustrates these two networks as modeled in *DTMS*.

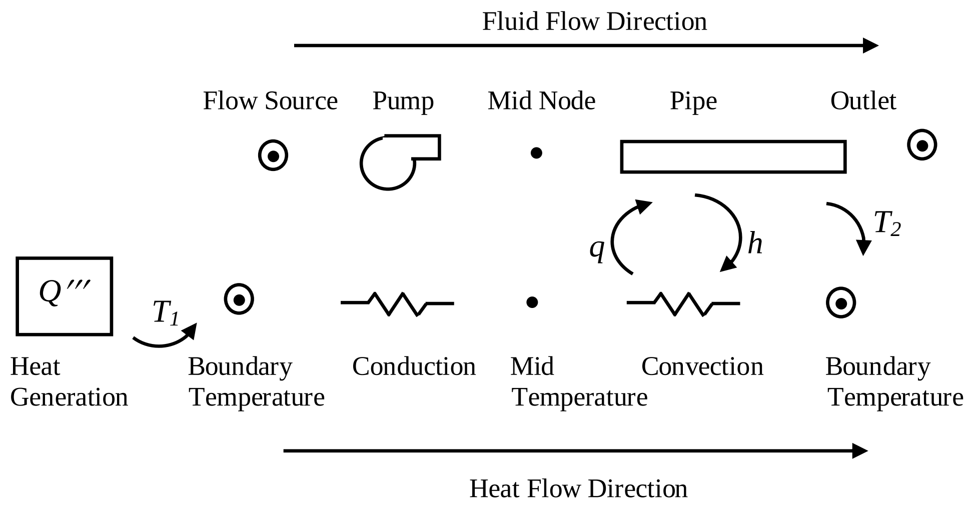


Figure 6.5: Cabinet bay model with liquid forced convection consisting of fluid and heat flow networks.

Figure 6.5 shows that there is feedback between the pipe, convection, and boundary temperature models, again not inherent to their respective classes. The pipe receives heat from the convection model while setting the convection coefficient, *h*, in the convection model and the temperature, T_2, at the downstream boundary. To incorporate this feedback into *DTMS* a group model *HeatSinkToPipeHeatExchanger* was created.

HeatSinkToPipeHeatExchanger

The class *HeatSinkToPipeHeatExchanger* is a wrapper consisting of two instances of *LinearHeatFlowModel,* two instances of *TemperatureEffortModel*, and a single instance of *Pipe*. The class models heat exchange from a heat sink to a pipe via direct conduction. Again, as with *FinsToPlenumHeatExchanger*, the model uses a wrapper to communicate the heat transfer occurring between the models, although they are resolved as different flow networks. The working fluid is an instance of *CaloricallyPerfectWater*, where internal energy and enthalpy are functions of temperature, and specific heats are constant.

The heat flow network consists of a *LinearHeatFlowModel* to simulate conduction through the heat sink, a *LinearHeatFlowModel* to simulate convection from the pipe flow to the metal mass of the pipe, a *TemperatureEffortModel* to simulate the boundary temperature in the pipe, and a *TemperatureEffortModel* to connect the conduction and convection models.

After the flow networks (heat flow and fluid flow) are resolved during a time step, the models must communicate with one another. Again, Equations 6.22, 6.24, and 6.25 are used to solve for the convection coefficient in the pipe. In Equation 6.22, the hydraulic diameter, D_h, is now the diameter of the pipe.

The *HeatSinkToPipeHeatExchanger* updates the convection coefficient of the convection model, the enthalpy of the working fluid, and the boundary temperature of the independent instance of *TemperatureEffortModel.*

6.4 Validation of Thermal Resistive Network Models

To validate the thermal resistive network for heat flow, a network was constructed to simulate airflow through a PCM-2 cabinet bay. Results of this simulation are then compared with [2]. The network consists of electronic conversion equipment dissipating heat into the plenum of its respective bay while a blower forces cool air across the electronics. Figure 6.6 illustrates the top bay of a cabinet with cool air entering the bay from the left, the air convectively cooling the electronics, and finally the hot air exiting the bay to be re-circulated once the air is cooled again by an air-to-water heat exchanger in the bottom bay of the cabinet. The cabinet is sealed to prevent air from entering or leaving.

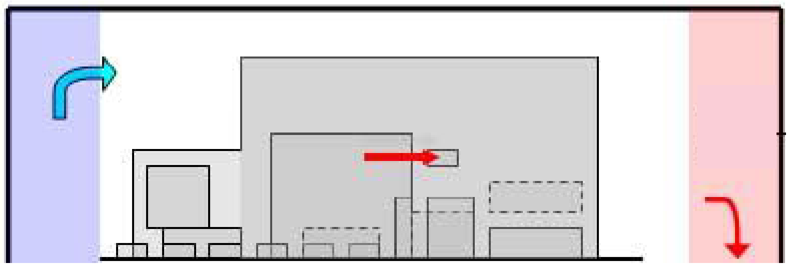


Figure 6.6: Bay of PCM-2 cabinet with forced air convectively cooling heat-generating electronics [2].

In this simulation, the electronics are producing a constant heat flux, and the blower produces a constant mass flow rate of air through the bay. Table 6.1 lists the input parameters for the fluid flow, cabinet environment, and constant heat generation of the electronics as set forth in [2].

Table 6.1: Input parameters for dynamic simulation of bay electronics with heat generation.

Input Parameters for Dynamic Simulation of Bay Electronics with Heat Generation	
Mass flow rate	0.17 kg/s (0.141 cfm)
Temperature of inlet air	317 K (~44°C)
Initial temperature of bay and electronics	317 K (~44°C)
Cabinet pressure	101625 Pa (atmospheric)
Internal heat generation	3022 W (~20 kW/m^3)

Figure 6.7 displays the transient behavior of the average temperature of the convectively cooled electronics. The electronics reach a temperature of 100°C after approximately 5.5 hours of operation at full capacity. The electronics reach a steady state temperature of 116°C, assuming the electronics can still operate above the 100°C threshold.

The convection coefficient reaches a steady state value of 96 W/m^2-K, very close to the value of 100 W/m^2-K set in [2]. The temperature of the air exiting the plenum reaches a steady state value of 61°C, two degrees greater than the expected value set forth in [2].

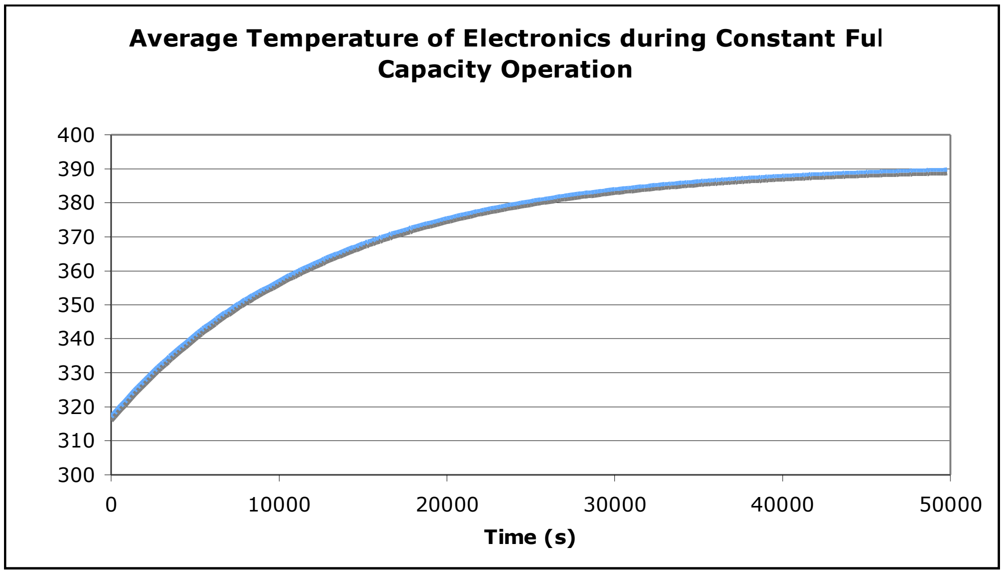


Figure 6.7: Average temperature of PCM-2 electronics at full capacity in convectively cooled bay.

6.5 Conclusion

This chapter focuses on the methods used to link the electrical distribution network and the thermal management framework. The models of the new heat flow network expand the *DTMS* framework into a new energy domain while maintaining modularity in a physics-based simulation environment. The models were validated either through previous literature examples or simulations in previous efforts through the Electric Ship Research and Development Consortium (ESRDC). The example simulation offered in Section 6.4 shows the capability of the network to accurately model the available steady state data. The transient aspects of the network were validated through examples obtained in physics-based textbooks. If transient data becomes available through the ESRDC regarding the thermal transient behavior of the cabinets, the accuracy of the thermal simulations of the cabinet models may be realized.

Finally, a discussion is not complete without considering the runtimes of the new thermal resistive network. The simulation discussed in Section 6.4 took 2.96 seconds to run a simulation for 50,000 seconds of data. This speed will be an asset later as electronic simulation will be shown to be costly.

7. Thermal-Electrical Co-Simulation

An energy-based, system-level model of a typical U.S. Navy warship must include the prime movers, energy generating, conversion, and storage devices, electrical distribution, propulsion and electrical loads, and the thermal management network. Integral to a system-level simulation is the relationship and feedback between thermal and electrical subsystems. Each subsystem cannot operate autonomously, and thus the dynamic interactions of the subsystems are crucial to the overall performance of the thermal-electrical system.

Chapters 3, 4, and 5 laid out the electrical framework, components, subsystems, and system configurations. Chapter 6 discussed the creation of a heat flow network to link the thermal management fluid flow network with the heat produced via resistive and switching losses during operation of the electrical framework. This chapter links the three flow networks (fluid, heat, and electrical) into a single dynamic thermal-electrical co-simulation. This chapter also addresses current and projected heat loads, the thermal dependency of electrical components, and finally other challenges and issues associated with a complete thermal-electrical co-simulation.

7.1 Thermal Dependency

When creating and simulating a thermal-electrical flow network overlaid with a thermal management network, it is crucial to understand the importance of the temperature dependency in the electrical models and the feedback between the fluid, heat, and electrical flow networks. This section addresses the impact of thermal effects by presenting a simulation of converter efficiencies, bulk loads, and temperature transience without thermal dependency and then comparing this to a simulation, of the same configuration, with thermal dependency.

The configuration for this simulation consists of a PCM-1 converting and providing dc power to a PCM-2 for further conversion to ac power. The converters are initially operating at a partial load of 2.5 kW. This creates heat loads of 85 and 135 W within the PCM-1 and PCM-2, respectively. These loads have an insignificant effect on the temperature of the electronics, as the thermal management network is able to maintain the converters within one degree Celsius of the starting temperature of 44°C (the temperature of the forced air [2]). The PCM-1 and PCM-2 are subjected to a load of 100 kW (the rated capacity of the PCM-1) after an arbitrary time of 30 seconds of running at partial load. The simulation time is 7200 seconds (2 hours) total. Figure 7.1 displays the configurations for both the thermally and non-thermally dependent cases. The PCM-4 is not included because sufficient data regarding geometry and thermal management aspects were not available at the time this thesis was written to make judgments regarding the operation of this component.

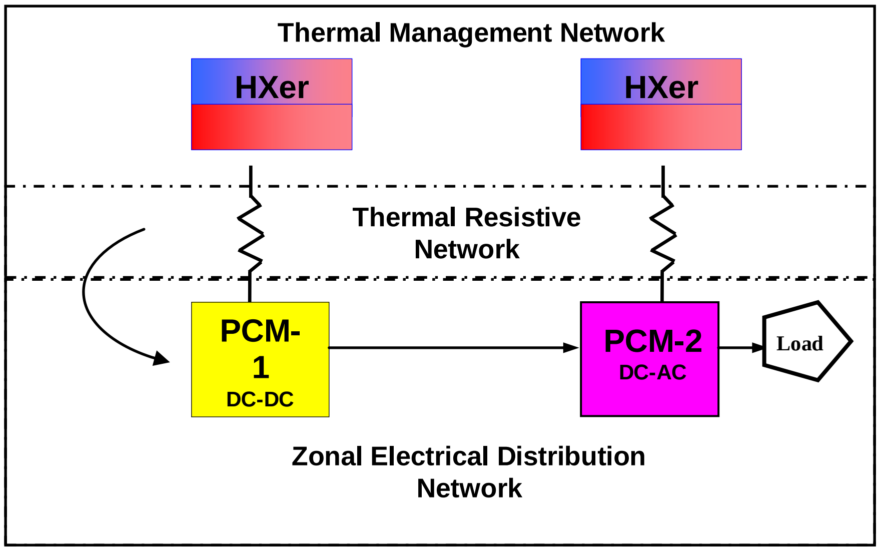

Thermal Management Network
HXer
HXer
Thermal Resistive Network
PCM-1
DC-DC
PCM-2
DC-AC
Load
Zonal Electrical Distribution Network

Figure 7.1: PCM configuration used to test thermal dependency of the converters.

During the simulation, temperatures of electrical components of the PCMs are updated every 50 time steps. The PCM model (*PCM1*, *PCM2*, or *PCM4*) is tasked with supplying the *InternalHeatGenerationModel* (discussed in section 6.1) with the appropriate heat load and then obtaining the feedback temperature from the *InternalHeatGenerationModel*. These steps are accomplished by using the function *setInternalHeatGenerationModel* available in each of the PCM models in the main file. The structure of this function may be seen in Appendix C, which contains the thermally dependent simulation main file.

To distinguish the thermal dependency or the lack of thermal dependency in each simulation, the function *setThermalDependency* is provided within the classes *PCM1*, *PCM2*, and *PCM4*. The function requires a Boolean value of either *true* (thermal dependency) or *false* (no thermal dependency), where *true* is the default value set internally in the converter classes. Table 7.1 presents the input parameters for the two thermal dependence cases.

Table 7.1: Input parameters for converters to address the effects of thermal dependence.

Input Parameters for Simulation Comparing Thermal Dependence	
Load	100 kW per bay of electronics
PCM-1 Update frequency	50 time steps per update
PCM-2 Update frequency	50 time steps per update
PCM-2 Initial loss free resistance	157 ohms
PCM-1 Air flow rate	0.052 kg/s
PCM-2 Air flow rate	0.182 kg/s

The temperatures in all of the figures in this chapter are the bulk internal temperature of the electronics. The electronics are modeled in *DTMS* as the class *InternalHeatGenerationModel*, discussed in Chapter 6, which outputs this internal bulk temperature. More detailed models of individual electrical components within the bays of the converter cabinets indicate that the temperatures of the electrical components vary no more five degrees Celsius from component to component [2]. While the individual components of the electronics have various specific heats, [21] suggests using a composite specific heat. The default value for the specific heat of the components is 385 J/kg-K. Figure 7.2 shows the efficiency (primary y-axis) and temperature (secondary y-axis) of the PCM-1 during the first 30 minutes of operation in the thermally dependent case. Similarly, Figure 7.3 shows the efficiency and temperature of the PCM-2 electronics during the first hour and six minutes of operation in the thermally dependent case.

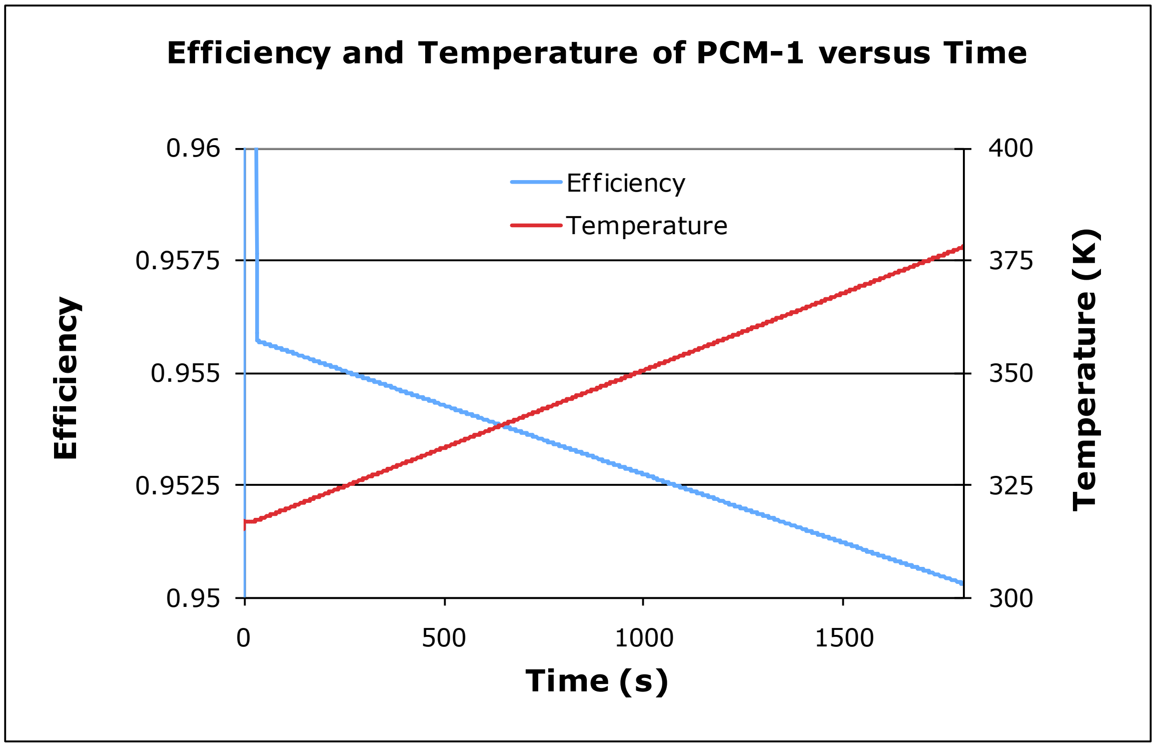


Figure 7.2: Efficiency and temperature of the thermally dependent PCM-1 during operation.

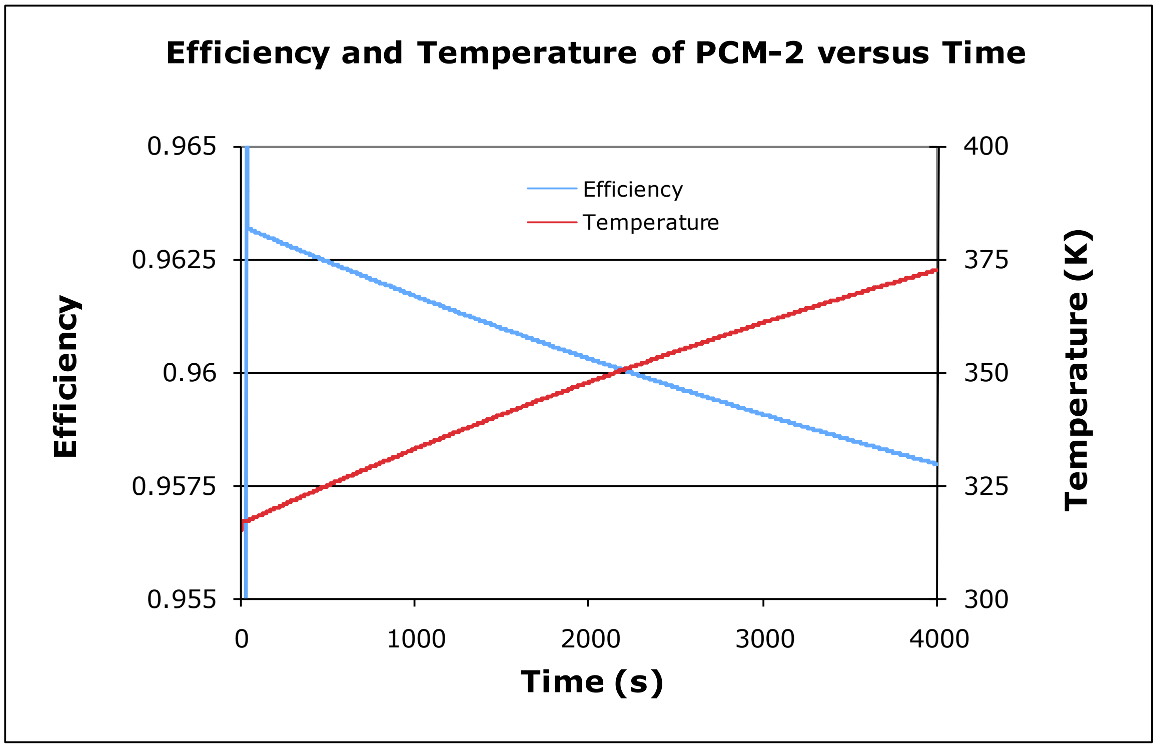


Figure 7.3: Efficiency and temperature of the thermally dependent PCM-2 during operation.

Figures 7.2 and 7.3 indicate a steady decline in efficiency as the temperature within the bays of each cabinet increases. The bulk internal temperature of the electronics within the PCM-1 reaches the threshold value of 100°C within 27 minutes. This temperature is used as a threshold due to the likelihood of malfunctioning transistors when they reach a junction temperature of 100°C [2]. At this critical temperature, the efficiency has dropped from 0.9558 to 0.9507, a difference of one-half percent. The PCM-2 reaches 100°C within 67 minutes. At this time, the efficiency has dropped from 0.9632 to 0.9578, again a difference of about one-half percent. While the lines in these figures appear linear, they are in fact slightly curved and would reach a steady state value after a very long time if the threshold temperature did not limit converter operation. In comparison, during the non-thermally dependent case the PCM-1 and PCM-2 maintain a steady state efficiency of 0.9559 and 0.9634, respectively, while reaching 100°C within 30 minutes and 74 minutes, respectively. In the thermally dependent case, the PCM-1 reaches 100°C three minutes faster than the PCM-1 in the non-thermally dependent case. Likewise, the thermally dependent PCM-2 reaches 100°C seven minutes faster than the PCM-2 in the non-thermally dependent case. The vertical blue line on the left of the each figure reflects the transience of the efficiency when the load is increased to the full capacity. The fact that the temperature and efficiency of each converter appears linear indicates that the cooling airflow rate and/or temperature of the forced air are too low to maintain the converters at a sufficiently low temperature to maintain operation.

The driving force behind the relatively poorer performance of the thermally dependent case study is the bulk heat load associated with each converter. Figure 7.4 shows the heat load associated with the PCM-1 and PCM-2 during operation. As the temperature increases, the efficiency drops, which in turn increases the bulk heat load, which in turn increases the temperature. Figure 7.4 shows that the heat load of the PCM-1 is increasing more rapidly than the heat load of the PCM-2. This difference is due to the fact that the PCM-1 in this particular simulation increasingly operates at a higher temperature than the PCM-2 because the PCM-1 is heating up more quickly. The rate at which the converters heat up determines the rate at which the converters produce waste heat. The vertical axis of the figure is resolved to a range of 3500 to 5000 kW to focus on the waste heat produced during operation at the rated capacity. The heat loads produced at the partial load are 1 to 2 magnitudes smaller than at full load capacity, and barely impact the operating temperatures of the electronics within the cabinets. The vertical lines in the figure indicate the transient period by which the heat load is increased.

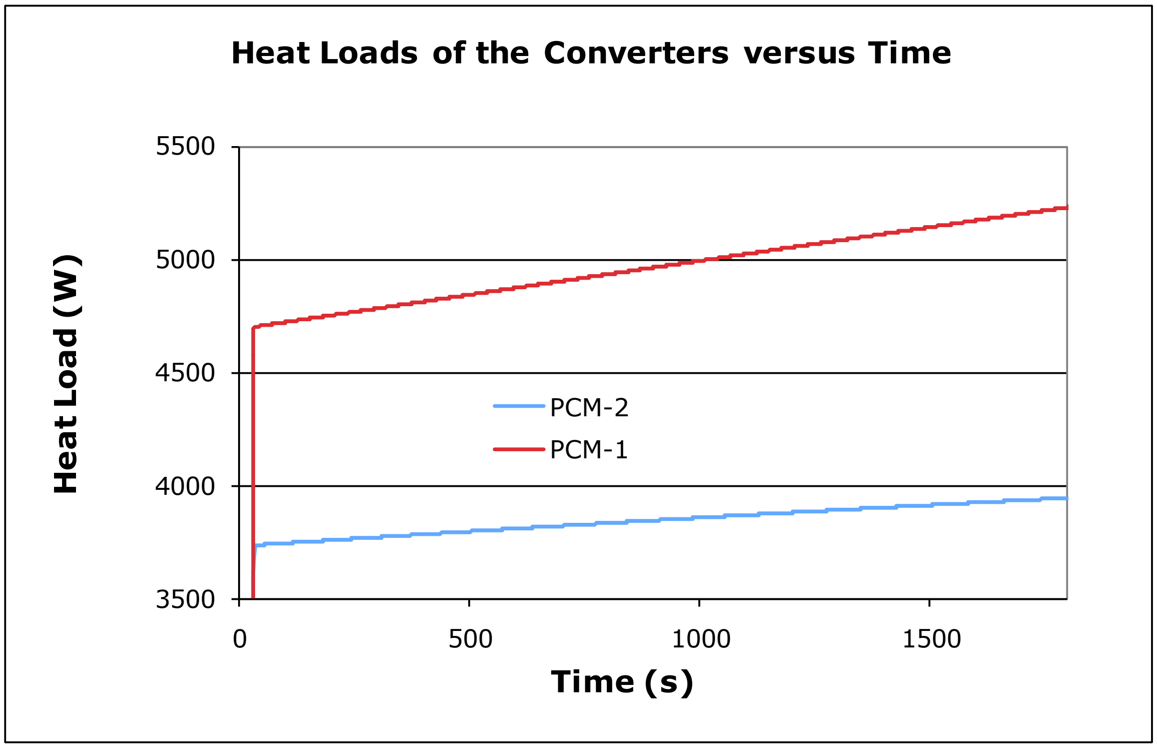


Figure 7.4: Heat loads (W) of the converters during conversion of 100kW.

The cycle of increasing temperature of the electronics mentioned in the previous paragraph leads to overheating of the converters, particularly in this case where the airflow rate of the cooling fan is too low. The airflow rate of the converters was held constant at the rate reported in [2] and simulated in Section 6.4. The PCM-2 reached the threshold of 100°C within 67 minutes, 233 minutes sooner in this simulation than that performed in Section 6.4 due to the fact that the heat loads have increased but the airflow has not. In [2], the power densities (heat per square foot of floor area of the converter) are reported to be 0.825 kW/ft^2 and 1.5 kW/ft^2 for the PCM-1 and PCM-2, respectively, at full load. A straightforward estimate indicates that at full capacity the PCM-1 and

PCM-2 generate 2.8 kW and 5 kW of excess heat per bay, respectively. This heat transfer rate converts to power densities of 1.4 kW/ft^2 and 2.5 kW/ft^2 for the PCM-1 and PCM-2, respectively. These power densities are considerably lower than the values of 5 to 8 kW/ft^2 expected by the year 2014 [2].

Possibly the most significant trend to understand regarding the thermally dependent case is the efficiency behavior of the converters versus the operating temperature of the electronics. Both the PCM-1 and PCM-2 show a drop in efficiency of 0.5% as the cooling fluid temperature rises from 44°C to the operating threshold of 100°C. While a 0.5% drop in converter efficiency may not seem significant, these efficiencies compound to create greater losses at a system level. If all 36 MW of rated power of the MT30 were converted from ac to dc (in a PCM-4), high voltage dc to low voltage dc (in a PCM-1), and finally dc to ac (in a PCM-2) and if the PCM-4, PCM-1, and PCM-2 were all operating at 100°C, this lower efficiency would create an additional 0.54 MW of waste heat. Maintaining the converters at a lower temperature, however, requires more power dedicated to thermal management. This situation creates the interesting optimization issue of determining a coolant flow rate that minimizes the power load dedicated to cooling, while also minimizing power losses due to increased temperatures in the converter cabinets.

A final aspect observed in these simulations is the difference in simulation time between the thermally dependent case and the non-thermally dependent case. The thermally dependent case had a computational runtime of 1840 seconds, almost twice as long as the non-thermally dependent case at 1080 seconds. This runtime difference is due to the fact that the thermally dependent case must update the electrical constants at an update frequency chosen by the user. Although this runtime is not prohibitive considering the amount of information gained during a 7200 second simulation, the importance of thermal feedback must be weighed against runtime and cost of the simulation, particularly when part of larger, more complex system simulations.

7.2 Multiple Zone Simulation

A system-level simulation framework should be able to model multiple zones of an all-electric ship. This demand is necessitated by the requirement that the future generation naval fleet be reconfigurable in the event one zone is damaged and power must be routed through other zones for uninterruptible power.

To demonstrate the capability of the *DTMS* Framework to model and simulate multiple zones in a thermal-electrical co-simulation, a benchmark model consisting of three zones was created. The simulation was run for air-cooled bays as well as elements of the system that are cooled by water. The system shown in Figure 5.2 consists of two power supplies, PS-1 and PS-2, feeding dc power to the port and starboard buses. These buses feed three zones, which contain two dc-dc converter modules (CM) and operate in parallel to feed dc power to the inverter modules (InM). The voltage levels are consistent with benchmark simulations previously discussed in Sections 5.2 and 5.3.

The biggest difference between the benchmark simulations run earlier and these thermal-electrical co-simulations are the simulation runtimes. The benchmark simulation discussed in Section 5.2 completes in 10 seconds. In the multiple zone simulation, the first 10 seconds remain the same, but at 10 seconds the PCM-2 models are subjected to full capacity loads for 24 minutes. Table 7.2 shows additional parameters not previously included in the benchmark simulation of Section 5.2. The update frequency in the table indicates the frequency at which the temperature of the electronics is updated. Simulations run with a simple configuration of one PCM-4, one PCM-2, and one PCM-1 converge while updating temperature at each time step. More complicated simulations, such as this multiple zone configuration, require less frequent updates to ensure convergence. If the update frequency is too low, the simulation does not converge. However, a greater update frequency loses accuracy. For example, in the thermal dependent case of Section 7.1, a simulation with an update frequency of 500 time steps per update produces temperatures 1.23% less than the simulation with an update frequency of 50 time steps per update.

Table 7.2: Electrical parameters for the electrical-thermal co-simulation of multiple zones.

Thermal-Electrical Co-Simulation: Electrical Parameters	
PCM-2 Full Load	125 kW per bay electronics
PCM-1 Update Frequency	100 time steps per update
PCM-2 Update Frequency	75 time steps per update
PCM-4 Update Frequency	200 time steps per update

The heat flow network is the equivalent of the simulation previously described in Section 6.4. The electronics are modeled as internally heat generating solids with energy storage. As mentioned in the case of thermal dependence in Section 7.1, the PCM models determine the internal heat generation rate. The heat flows through the thermal network until transferred to the fluid flow network in the class *PlenumHeatExchanger* or *HeatSinktoPipeHeatExchanger* depending on the case, as discussed in Section 6.2. The geometry of the bay and electronics in the simulation are based on values set forth in [2] and can be found in Appendix D, which contains the main file for this simulation.

The fluid flow network is the same as discussed previously in Section 6.4. Air at approximately 44°C (317 K) is forced through each bay of the PCM [2]. There are four identical bays of electrical equipment in each PCM and each PCM is designed to share the load equally between the electronics of each bay. Thus, an identical amount of heat must be removed from each bay. Because these bays are designed for load sharing, the simulation considers only one bay of each of the PCMs to eliminate repetition. It is known that the inverter (PCM-2) and dc-dc converter (PCM-1) cabinet electronics are divided into four bays. It is assumed in the simulation that the ac-dc converter (PCM-4) is also divided into four bays. Table 7.3 shows the parameters for the fluid flow network. The airflow rate of the PCM-4 is unknown, thus a nominal value was chosen.

Table 7.3: Fluid input parameters for air-cooled, thermal-electrical, multi-zone simulation.

Thermal-Electrical Co-Simulation, Air-Cooled: Fluid Parameters	
PCM-1 design flow rate	0.052 kg/s
PCM-2 design flow rate	0.185 kg/s
PCM-4 design flow rate	1.5 kg/s
Initial fluid temperature	317 K (~44°C)

The total heat load for the three zones amounts to approximately 28.3 kW or about 9.4 kW per zone. Figure 7.5 shows the temperature of the 11 converters during operation over 24 minutes while Figure 7.6 shows the efficiencies of these converters. The trends are consistent with the trends found in Section 7.1. As the cabinets heat up, the efficiencies drop thus creating more heat to remove from the cabinets. In Figure 7.5, the PS-1 and PS-2 heat up more quickly than the other converters both reaching 430 K after 1200 seconds of the simulation. This greater rise in temperature is due to the fact that these converters are generating greater bulk heat loads. This greater rise in temperature in the PS-1 and PS-2 also corresponds with their greater drop in efficiency as compared to the other converters in Figure 7.6. The port side CM and each zonal InM are lumped together in Figure 7.5 each reaching 340 K after 1200 seconds of operation. The starboard CMs operate at the lowest temperatures because they are generating the least amount of waste heat. These converters reach 330 K after 1200 seconds of operation. Each of the CM and InM operate at efficiencies between 95 and 97%.

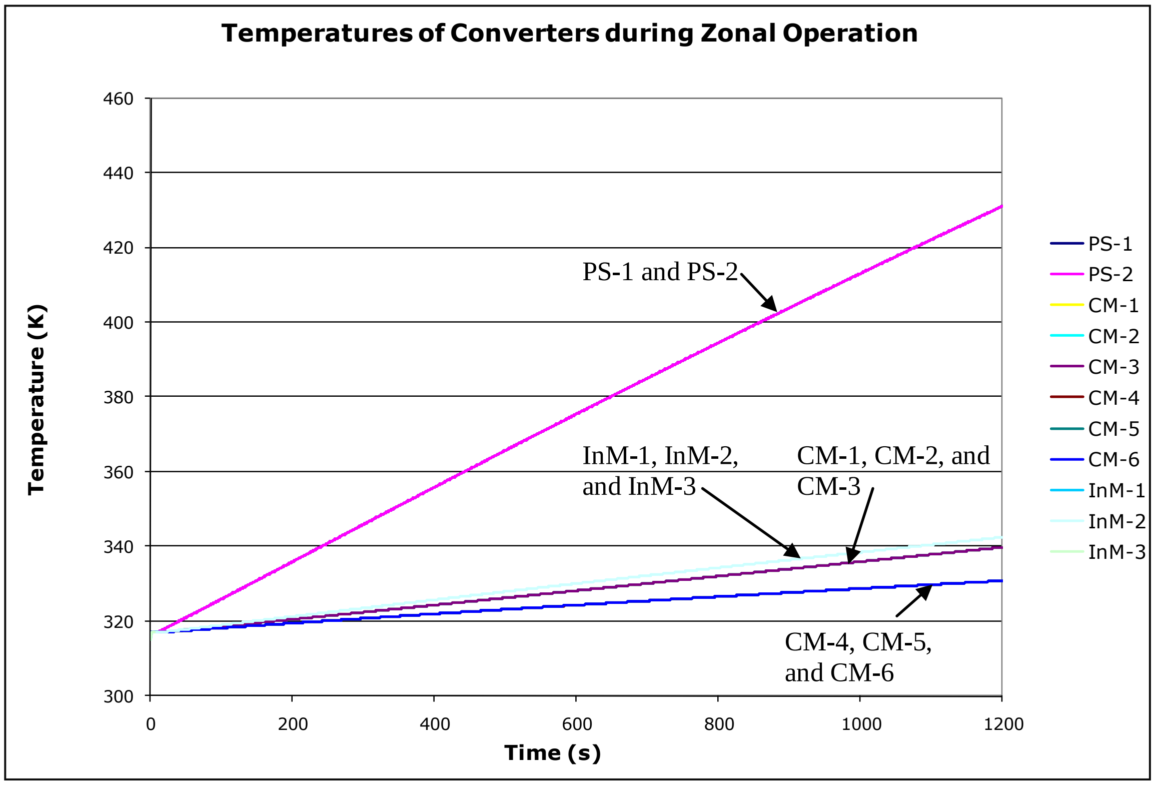


Figure 7.5: Temperatures of the thermally dependent converters during zonal operation. Temperatures of the PS-1 and PS-2 are nearly identical and increase the fastest. Temperatures of the CM-1 thru 3 and InM 1 thru 3 are nearly identical and increase slightly faster than the temperatures of the CM-4 thru 6, which are all nearly identical.

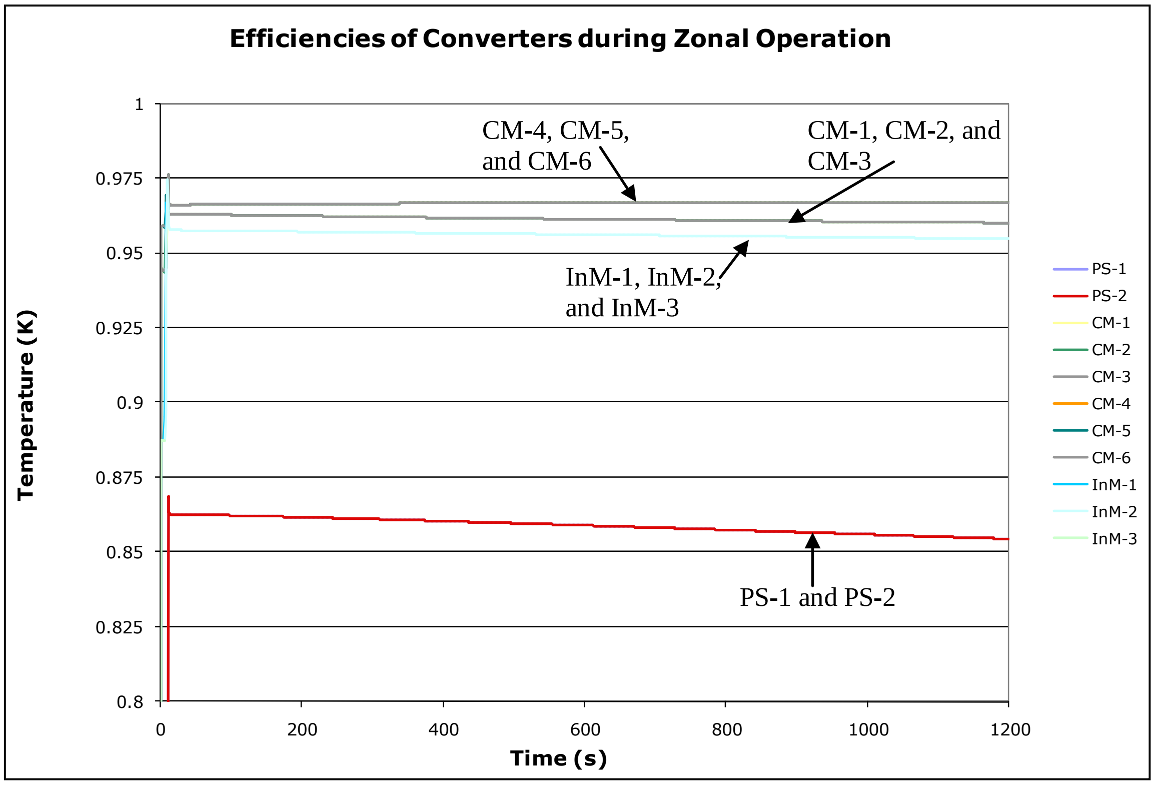


Figure 7.6: Efficiencies of the thermally dependent converters during zonal operation. Efficiencies of the PS-1 and PS-2 are nearly identical and are the lowest. Efficiencies of CM-4 thru 6 are nearly identical and slightly higher than CM-1 thru 3, which are nearly identical and slightly higher than InM1 thru3, which are also nearly identical.

The most significant parameter in these simulations is the heat transfer coefficient of the airflow across the electronics. The heat transfer coefficient of the fluid flow as calculated in these simulations reaches about 100 W/m^2-K. However, with proper design, forced heat transfer in turbulent air can achieve values as high as 250 W/m^2-K [12]. *Case 1* in Section 7.3 discusses the potential for, as well as the limitations of, increasing the convective heat transfer coefficient across the electronics.

An important observation in this simulation is that the PCM-4 behaves in the same manner as the PCM-1 and PCM-2. This behavior was expected as the greater temperatures cause greater internal resistance, which decreases the efficiencies. Figure 7.7 shows the efficiency (primary y-axis) and temperature (secondary y-axis) of the PS-1 (the starboard PCM-4). The efficiency drops by 0.37% from the starting temperature of 44°C to the point at which the cabinet reaches 100°C. Another observation is that the PS-1 and PS-2 are converting only 200kW/bay of power each, far below their rated capacity of 750 kW/bay. This situation occurs because each InM can only convert 125 kW at rated capacity. Thus, with three InM in a zone, the maximum load per bay for the two PS is about 200kW/bay. To ensure the greatest efficiency, zones should be designed to operate such that each converter concurrently operates at its rated capacity. As a result of operating at less than rated capacity, the efficiency is at best 86.2% at a temperature of 44°C. This low efficiency comes about because the PCM-4 has internal resistance and will always be producing waste heat as long as a potential exists across the converter.

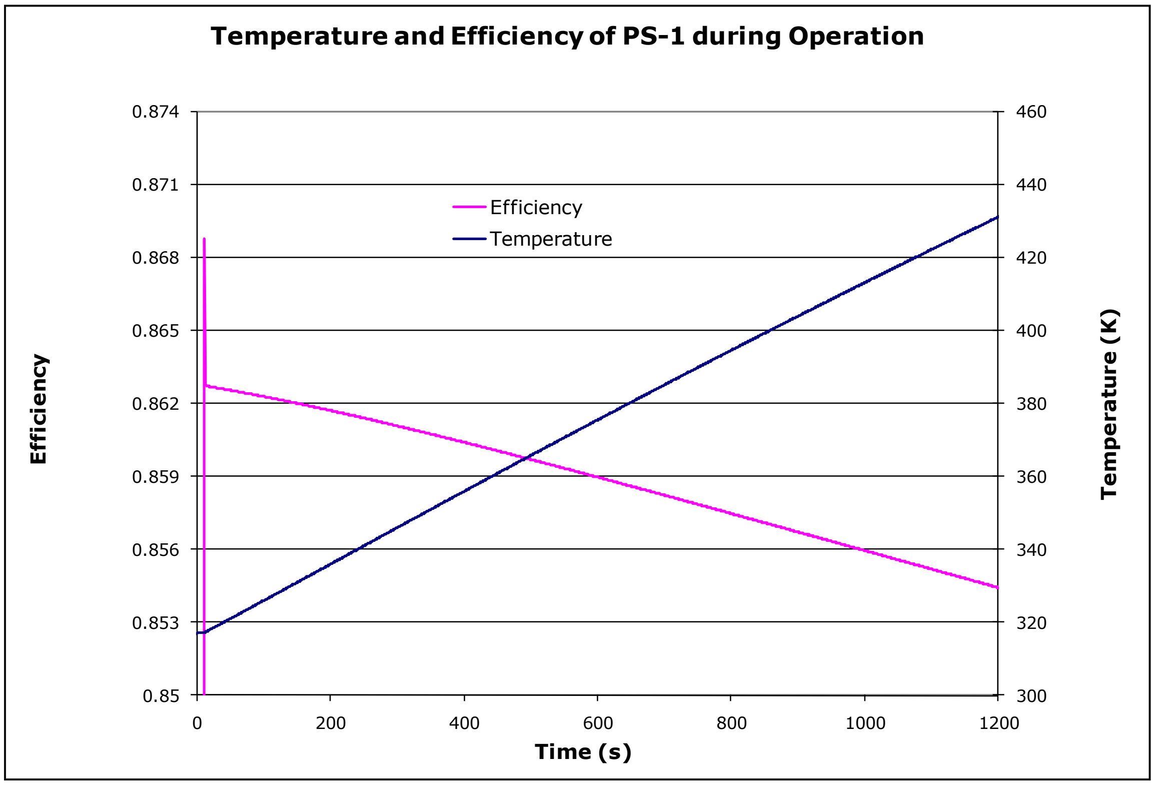


Figure 7.7: Efficiency and temperature of the thermally dependent PCM-4 during operation.

Finally, it is of interest to discuss the runtime of the multiple zone simulation. This simulation took three orders of magnitude longer to complete for the same simulation time. The simulation runtime is of grave concern because the intent of *DTMS* is to provide a simulation environment that can handle various scenarios with high accuracy and low runtime. Notably, it is observed that individual converters perform nearly identically in a multiple zone simulation as they do individually. To demonstrate, the PCM-2 of the multiple zone simulation was isolated to observe its behavior under the same conditions, but separated from the multiple zone configuration. Figure 7.8 shows the efficiency (primary y-axis) and temperature (secondary y-axis) of the PCM-2 in the isolated converter scenario.

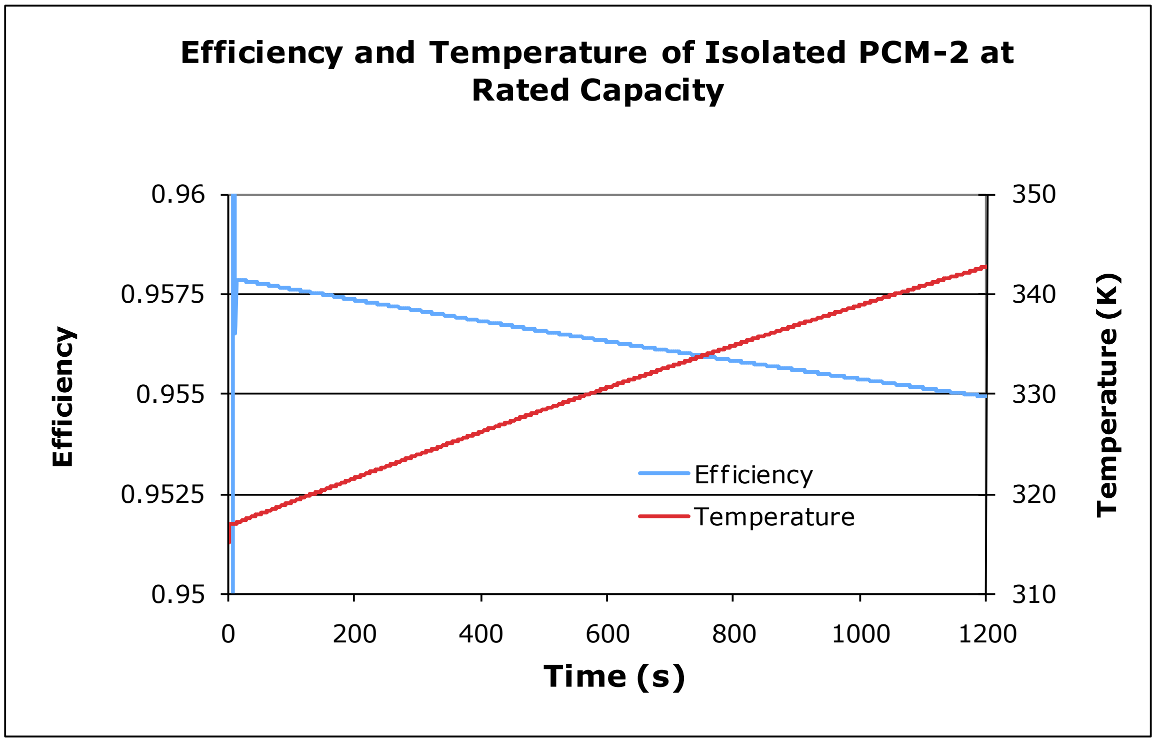


Figure 7.8: Efficiency and temperature of the isolated PCM-2 during operation.

The behavior of the PCM-2 in Figure 7.8, which shows the performance of the PCM-2 in the zonal configuration, is nearly identical to behavior of the PCM-2 (InM) in the multiple zone simulation. At 1200 seconds, the isolated converter operates at an efficiency of 95.4921% while the zonal converter operates at an efficiency of 95.4949%. The difference in efficiencies amounts to a relative difference of 0.002%. The isolated converter reaches a temperature of 342.734°C at 1200 seconds while the zonal converter reaches a temperature of 342.544°C. The difference in temperature amounts to a relative difference of 0.055%. This observation suggests that all things equal, the performance of a converter within a zone is essentially equivalent to an isolated converter.

7.3 Case Studies

Merging of the thermal management, thermal resistive, and electrical frameworks allows for optimization of the shipboard systems. The modularity and reconfigurability of the *DTMS* models gives freedom and endless possibilities of geometry, flow rates, and other parameters to find an optimal configuration. This section looks into the key parameters that play a role in determining thermal performance. These parameters include airflow rate, water-cooled versus air-cooled bays, and the influence of power factor.

Case 1: Increased Airflow Rate

By increasing the flow rate in the thermally dependent case of Section 7.1, the optimal airflow rate to maintain the converters below 100°C can be found. Running the thermally dependent case while increasing airflow rates to 0.22 and 0.2 kg/s for the PCM-1 and PCM-2, versus 0.052 kg/s and 0.182 kg/s previously, produces the results shown in Figure 7.9 and 7.10 for the temperature in the bays of these cabinets. These airflow rates reflect values that produce heat transfer coefficients approaching the upper limits feasible for forced convection of gases. A typical range of values of the convection heat transfer coefficient for forced convection of gases is 25 to 250 W/m^2-K [12]. In this simulation, the heat transfer coefficients for the PCM-1 and PCM-2 are 247 and 243 W/m^2-K, respectively. To increase the airflow rate, the PCM-1 fan must increase its power usage by 400 W, while the PCM-2 fan must increase its power usage by 280 W. This increase in the airflow reduces the rate at which the electronics heat up and reduces the rate at which the efficiencies decrease. The power saved through increasing the airflow rate and maintaining a higher efficiency only amounts to 55 W and 10 W for the PCM-1 and PCM-2, respectively. This fact leads to the conclusion that maintaining the converters at a low temperature does not equate to greater system-level efficiency. The power usage by the fans is greater than the power savings in the converters.

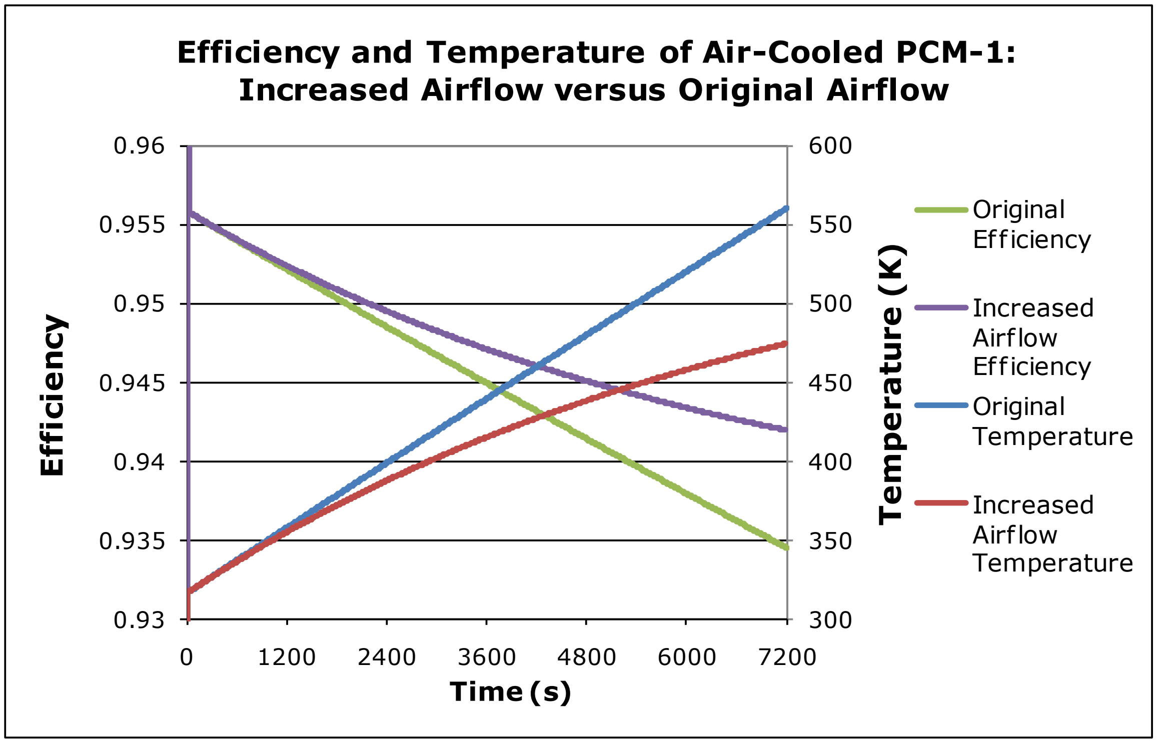


Figure 7.9: Efficiency and temperature of PCM-1 with original versus increased airflow rates.

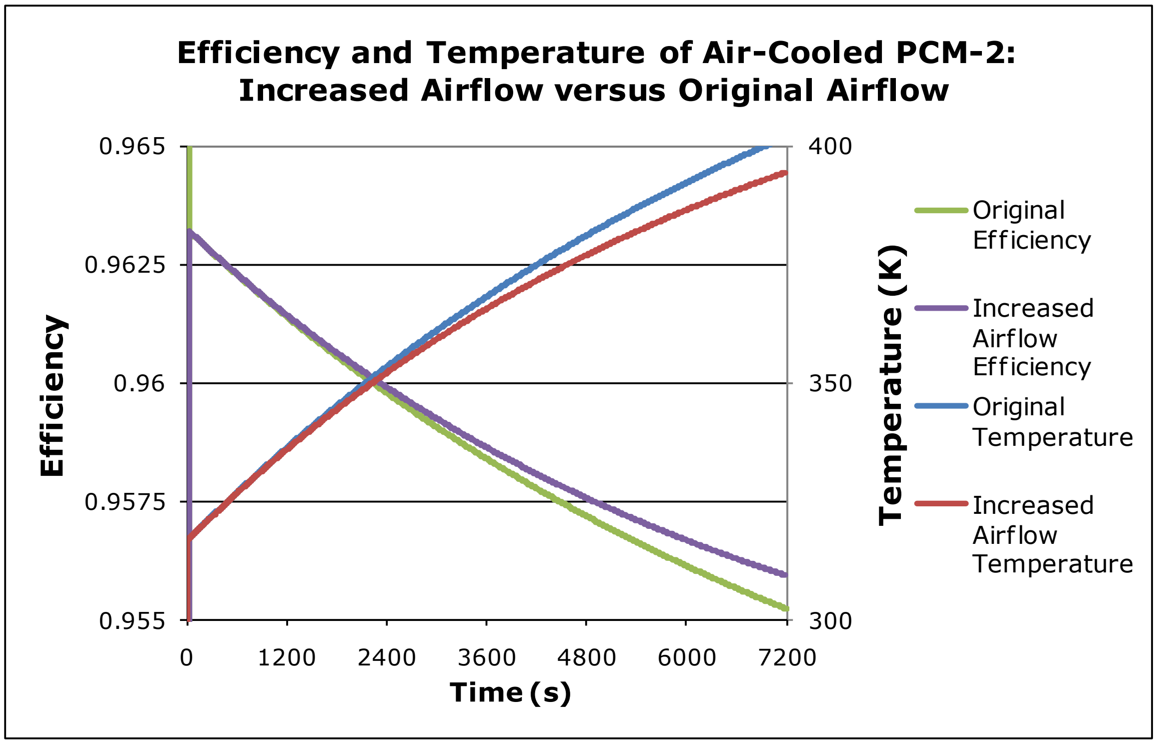


Figure 7.10: Efficiency and temperature of PCM-2 with original versus increased airflow rates.

At the full load capacity of the PCM-1, this level of forced convection is only able to maintain the temperature of the electronics below 100°C for 31 minutes. Similarly, the PCM-2 exceeds the temperature threshold after 72 minutes. The increased airflow rate allows the converters to eventually reach a steady state operating temperature. This temperature, however, is much greater than 100°C. As mentioned in Chapter 1, experience has indicated that the converters occasionally cannot operate at full capacity with the current air-cooled thermal management unless the panels are removed from the cabinet [27]. This simulation confirms that air-cooled converter cabinets are not viable with expected power densities aboard the AES.

Case 2: Liquid Forced Convection

Liquid forced convection is an attractive alternative to air-cooled cabinets because the local heat transfer coefficient in the vicinity of the electronics can be much higher than in the air-cooled case. Liquid forced convection can achieve heat transfer coefficients of up to 20,000 W/m^2-K, which in this case is achievable using high flow rates, but unrealistic in converter applications.

In the next simulation, the fluid flow network is similar to that discussed in Section 6.3. The electrical configuration is identical to that discussed in Section 7.1. However, instead of cooling the electronics using air-based convection, they are now cooled using water-based convection. Chilled water at approximately 6.6°C (280 K) is pumped through a pipe in direct contact with a thermal heat sink, which is in direct contact with the electronics of each cabinet bay. Again one bay from each cabinet is simulated to eliminate repetition of computation in the simulation, and the cabinets are assumed to consist of four bays. Table 7.4 shows the parameters used in the fluid flow network. The nominal design flow rates reflect those values suggested in [27].

Table 7.4: Fluid input parameters for liquid-cooled, thermal-electrical multi-zone simulation.

Thermal-Electrical Co-Simulation, Liquid Cooled: Fluid Parameters	
PCM-1 design flow rate	1.39 kg/s
PCM-2 design flow rate	1.39 kg/s
Initial fluid temperature	280 K (~6,6°C, ~44°F)

Figures 7.11 and 7.12 compare the bulk internal temperatures and efficiencies of the electronics of the water-cooled converters to those of the air cooled-converters of Section 7.1. The forced convection of water successfully removes heat from the heat sink at a rate much greater than its counterpart, the forced convection of air. The greater capacity to remove heat comes from the greater specific heat and larger heat transfer coefficient of the water, which achieves a convection coefficient of 4825 W/m^2-K, a full order of magnitude greater than the heat transfer coefficient for the forced air. However,

fouling and metal resistance lower the overall or effective heat transfer coefficient to 312 W/m^2-K between the heat sink and fluid in the pipe. The temperature of the PCM-1 reaches 100°C after 48 minutes, 17 minutes longer than the air-cooled PCM-1 of *Case 1*. The PCM-2 appears to be headed towards a steady state temperature below this threshold. The efficiencies of the converters mirror the temperature behavior of the respective converters and reach steady state values concurrently. Clearly, from the behavior of the converters in this water-cooled simulation, the Navy must move toward cooling methods employing greater heat transfer coefficients. Techniques using forced air as a heat transfer medium are inadequate, and the Navy must consider alternate means such as heat sinks with chilled water or phase-change and immersion cooling, to address the greater future heat loads expected in these cabinets [28].

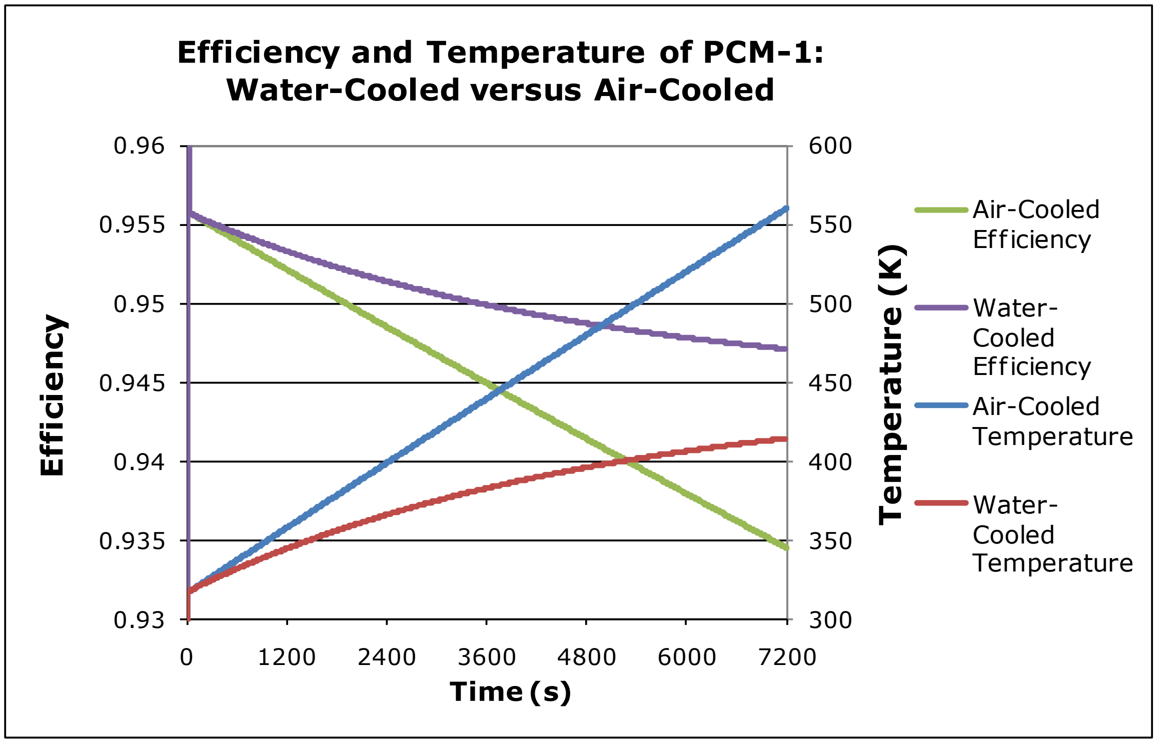


Figure 7.11: Temperature and efficiency of water-cooled and air cooled PCM-1.

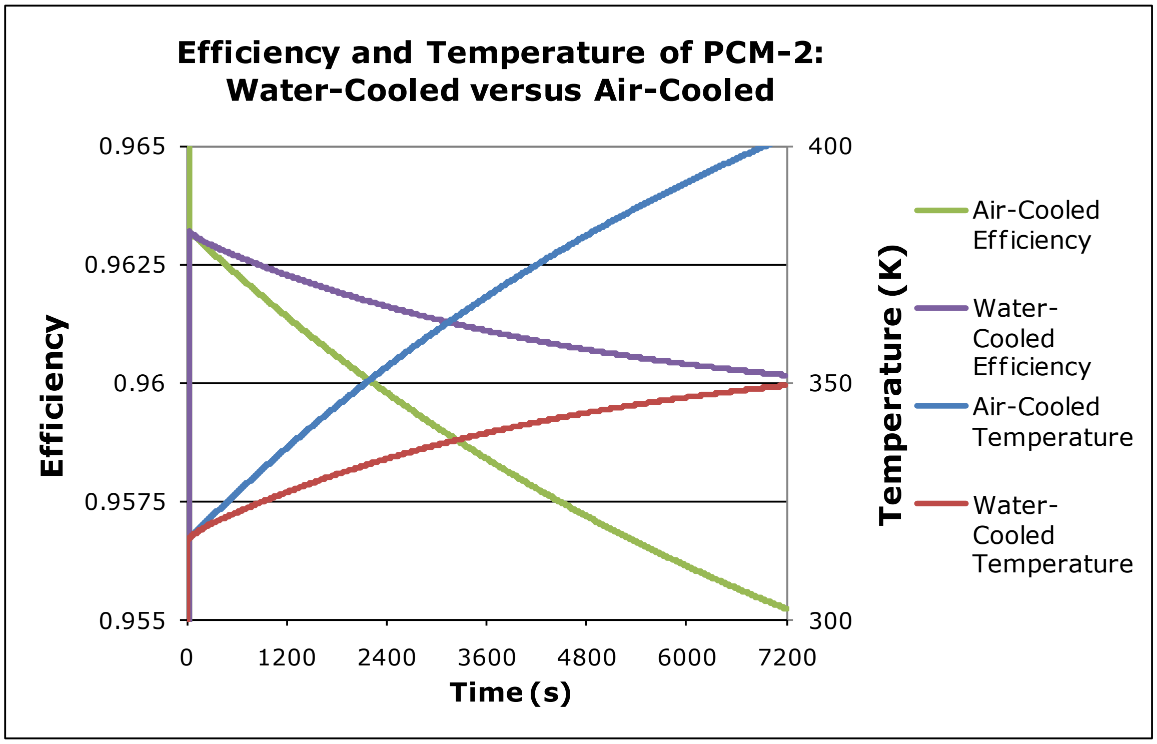


Figure 7.12: Temperature and efficiency of water-cooled and air-cooled PCM-2.

Case 3: Power Factor

This case investigates the influence of the power factor. Again, the power factor is the ratio of the real power flowing through a system to the apparent power. Nonlinear loads that distort the waveforms and energy storage devices, which store and return power to the source, create this apparent power, which fortunately does not produce heat. Intuitively a low power factor means that there is less power that can be dissipated. However, a load requires a certain amount of real power, so a network with a low power factor must produce more real power to make up for the increased apparent power.

To test the effect of the power factor, the configuration in Section 7.1 was selected. The power factor for each converter was set at 0.8, lower than the default power factor up to this point of 0.96. Figure 7.13 displays the temperatures of PCM-1 and PCM-2 under this circumstance.

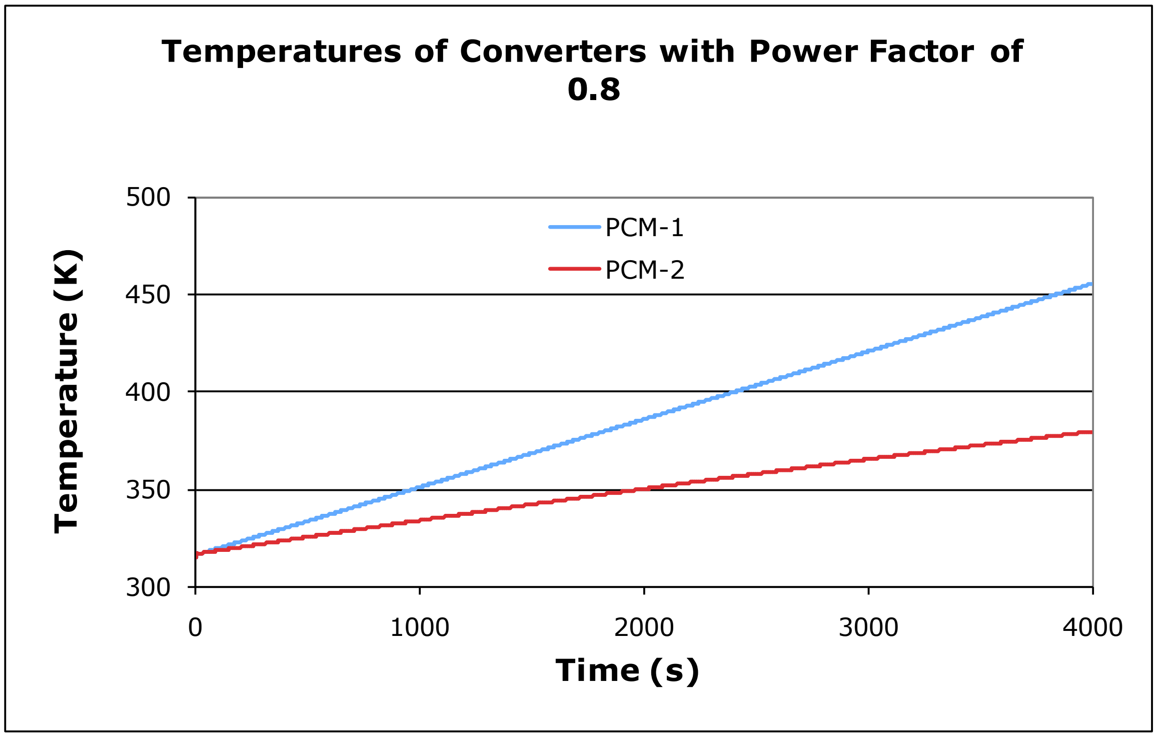


Figure 7.13: Temperatures of converters with power factors of 0.8.

This figure, when compared with Figures 7.2 and 7.3, shows that there is no appreciable thermal effect based solely on the power factor. This observation is not surprising considering that apparent power does not dissipate as thermal energy but is returned to the source. The difference lies in the amount of power provided to the network from the source (e.g., the generator). The generator must produce 1.25 times as much power to account for the gap in delivered power. Although no appreciable difference in the performance of the converters was observed, extending this conclusion to the generator would obviously be incorrect.

Case 4: Multiple Transient Events

This final case demonstrates the capabilities of handling multiple events in *DTMS*. This case uses the same configuration as *Case 2*, but several events, listed in Table 7.5, are added to simulate the resultant transient temperatures of the converters.

Table 7.5: Time and event load.

Time and Event Load for Water-Cooled Converter Simulation	
Time	**Load**
0 sec	2.5 kW
30 sec	100 kW
1800 sec	10 kW
3600 sec	2.5 kW
7200	10 kW
18000	100 kW
19800	2.5 kW

Figure 7.14 displays the transient temperatures of the converters during operation. It is evident from the figure that the greatest rate of the temperature increase corresponds to periods of operation at full capacity. It is also evident that the periods of operating at partial load do not require cooling water flow rates as high as 1.39 kg/s. *DTMS* can be used as an optimization tool to find ideal flow rates to maintain the converters at specified operating temperatures.

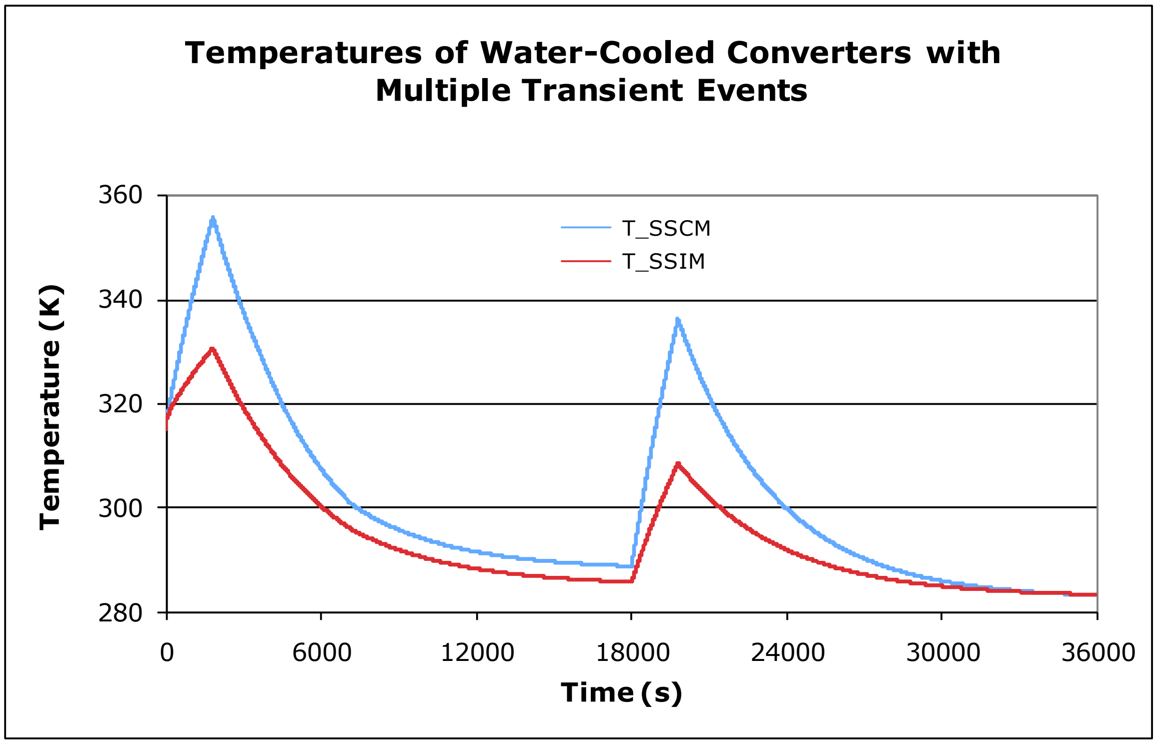


Figure 7.14: Temperatures of water-cooled converters during several transient events.

7.3 Conclusion

While the simulations discussed in this chapter prove the capabilities of the new electrical and thermal resistive networks to simulate system-level networks, several aspects of these simulations deserve some comment and discussion. These issues are addressed below.

The simulations reported here show the ability to link the thermal and electrical aspects of a system with reasonable fidelity. Thermal dependence is clearly an important issue in the design and optimization of these systems because electrical performance is intimately linked to thermal operating temperatures. Another important question addressed here is the importance of an ac or dc network as related to the thermal management system. The type and quality of power flowing through the electrical distribution network does not weigh as heavily as the bulk heat loads and efficiencies associated with power distribution. These heat loads are associated with resistive and switching losses, not the type of power (e.g., ac or dc) flowing through a converter. The power flowing between the converters experiences Joule heating losses whether the power is ac or dc. Transmission distances of the electrical power are negligible and will introduce no appreciable difference in transmission losses between ac and dc power. Thus, the best way to reduce system-level electrical conversion losses is to reduce the number of units or improve the conversion efficiency of those necessary to manage an electrical distribution system. The thermal management framework in *DTMS* views the electrical system as a black box producing a heat load. Minimizing losses through improved conversion or by optimizing power distribution is the principal means for optimization of the thermal management network.

One must weigh the pros and cons of a system-level thermal-electrical co-simulation given the costly nature of electrical simulations. With a series configuration of converters, demonstrated in Section 7.1, the simulation runtimes are short. However, when converters run in parallel and zones are introduced, the runtime of the simulation becomes a limiting factor to design and optimization.

A few reasons came to mind as to why the electrical simulations run more slowly than thermal simulations. First, the time step of the electrical network is many orders of magnitude smaller than the time step of the thermal management network. The numerical differentiation and integration techniques within the electrical network require smaller time steps and ultimately slow down the simulation. Table 7.6 shows the breakdown of computing time for a strictly electrical simulation, the benchmark simulation in Section 5.3, and a strictly thermal simulation, the thermal simulation in Section 6.4. In the electrical simulation, essentially 100% of the computing time is dedicated to the solvers. The thermal simulation only dedicates 14.5% of the computing time to the solvers. "Other" computation time includes setting defaults, initialization, and message between models.

Table 7.6: Breakdown of computing time in *DTMS* models.

Breakdown of Computing Time in *DTMS* Models		
Electrical	System Solver	99.84%
	Calculate States	0.09%
	Other	0.07%
Thermal	System Solver	14.5%
	Calculate States	37.7%
	Other	47.8%

Updates of electrical parameters affecting thermal feedback and the loss-free resistor also slow down the simulation, which was evident in the thermally dependent cases. Potential solutions to these issues are offered in the concluding Chapter 8. This thermal feedback, however, weighs heavily on the internal resistance of a converter. The temperature-dependence increases the internal resistance of the transformer, transistor, and snubber resistor by 21.8% as the temperature increases from 44°C to 100°C. This increase in internal resistances significantly impacts efficiencies of the converters during operation, as was evident in the simulations presented in this chapter. These drops in efficiency require the gen-sets to produce more power and further increase power losses upstream at the source of the power. Temperature-dependent changes in the capacitance of each converter did not have significance in a thermal management sense. While the capacitance increased due to temperature changes, once the capacitor is charged after a transient event it does not further contribute to the bulk heat load. While material properties of the electronics play a significant role in the heat capacity of the components, material selection is limited due to a material's electrical behavior. Thus, from a thermal standpoint material selection is limited.

8. Conclusions and Future Work

The simulations in Chapter 7 provide evidence that *DTMS* is viable as a framework to aid in the design and optimization of the next generation AES. The work presented in this thesis has addressed the implementation of an electrical and thermal resistive network, both integral elements of a system-level simulation environment. The work presents conclusive evidence supporting the idea that a large scale electrical simulation is not complete without thermal feedback. While the thermal-electrical aspects of a system-level simulation have been addressed, several areas of the resulting network must be addressed as *DTMS* moves closer to being a complete system-level, thermal simulation framework for all applicable energy domains.

8.1 Electrical Domain

Electrical system simulation can be costly and certainly consumes valuable user and computational time. To assist in the design and optimization of the all-electric ship, the Navy requires a system-level, physics-based simulation framework capable of evaluating the pros and cons of a large number of system configurations with accuracy and in a time effective manner. Thus, an electrical simulation must balance both accuracy and responsiveness. The newly developed electrical framework in *DTMS* currently is accurate but time-intensive. Power converters in series and linked to the thermal management framework prove to converge rapidly and provide both accurate and useful results. However, placing the converters in parallel, as is required to conduct zonal simulations, has resulted in a significant slowdown in computation runtime. There are several options for confronting these limitations. Two alternatives are discussed here:

averaged value models and metamodels.

Currently, the solvers in *DTMS* do not provide a method for solving the system-level problem analytically. Thus, models are required to provide the solver with its flow as a function of the efforts surrounding the flow model. The current flow for the inductors within each PCM model in *DTMS* is solved for by using numerical integration methods discretized in time. Numerical integration techniques are an intensely studied area in computer science; yet these techniques are known to be slow and computationally costly. An alternative to these costly numerical integrations is to utilize averaged value, or pseudo steady state, models. These models would be similar to the hydraulic fluid flow models already employed in *DTMS*, which neglect inertial momentum. From a thermal standpoint, the transience of electrical systems is a minor perturbation to the thermal management network. As the simulations within this thesis have shown, bulk heat loads, not transient heat loads, are the primary factor in the thermal dynamics of the system. As an example, Figure 8.1 shows the efficiency of the PCM-1 during the transient period from partial to full load as shown earlier in Figure 4.6, but now overlaid with a pseudo steady state efficiency. Figure 8.2 shows the resulting heat load of both the transient PCM-1 and the pseudo steady state PCM-1 simulations.

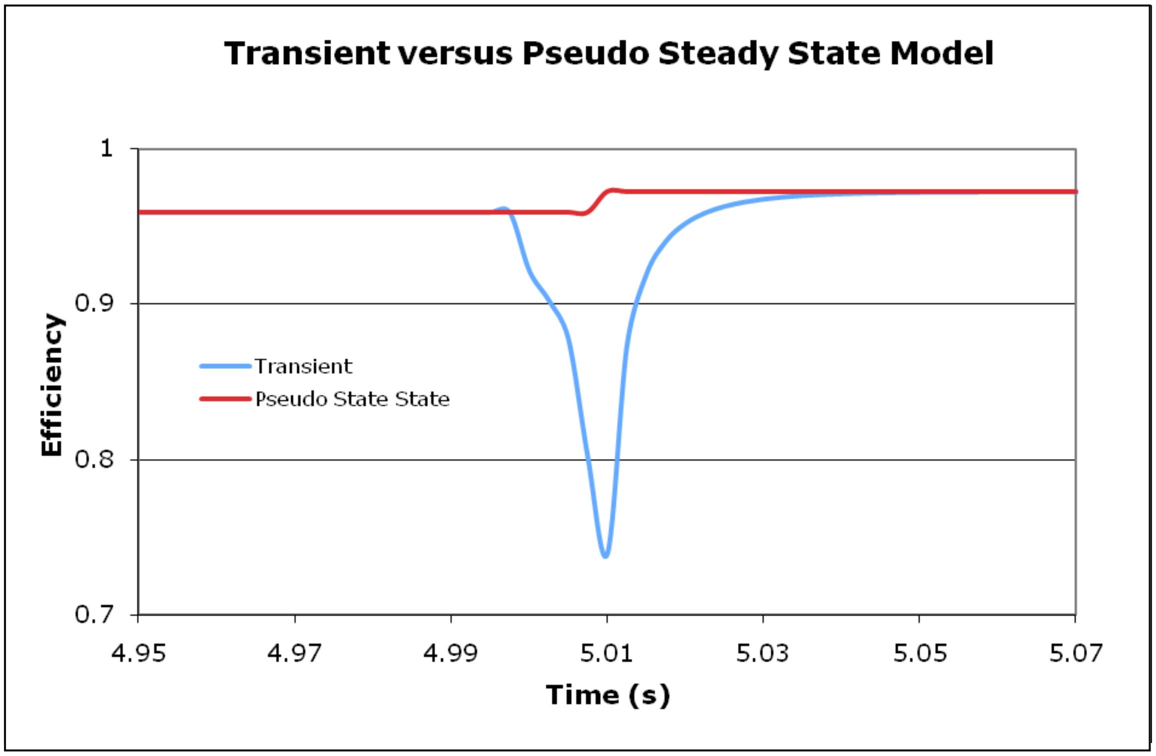


Figure 8.1: Efficiency of the PCM-1 for transient and pseudo steady state models.

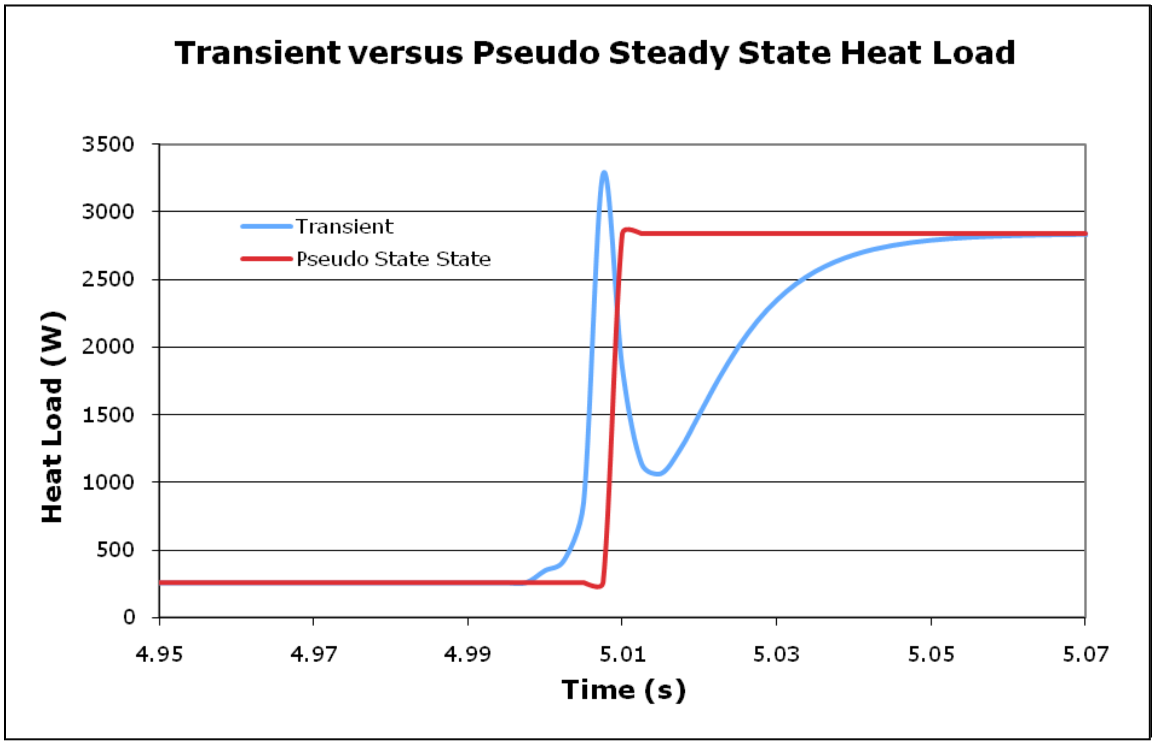


Figure 8.2: Heat load of the PCM-1 for transient and pseudo steady state models.

During the period from 4.95 to 5.07 seconds, the difference in energy, i.e., the power integrated over time, amounts to 20 Joules. This equates to a difference of 0.00125 °C on the thermal side. From a thermal management perspective, this is insignificant. However in certain applications, such as the immense heat created by the electro-magnetic rail gun, this discrepancy could be the difference between thermal management maintaining functionality or leading to the failure of the weapon system.

The most promising alternative to the detailed and accurate, but expensive, electrical models is the use of statistical approximation techniques. The application of these techniques to the electrical distribution network allows for creation of surrogate models called metamodels that can approximate the detailed thermal-electrical behavior. Metamodels are capable of maintaining the overall accuracy of the physics-based models with the benefit of an executable running nearly instantaneously [20]. These models allow the user to explore the design space while manipulating design parameters to facilitate design optimization. Statistical approximation, for example, could allow the user to address the thermal-electrical problem of finding the optimal converter operating temperature to minimize converter losses and thus the power dedicated to thermal management.

The *DTMS* framework came about because of the need to understand the dynamics of system-level thermal management. However, as system-level simulations in *DTMS* have become more complex, the need for sophisticated control techniques has increasingly becoming more apparent as, for example, was the case in the benchmark simulations of Chapter 5. Controls provide faster numerical simulations while enhancing the entire analysis and design of shipboard systems. Because system-level simulations require the mating of several energy domains and their associated controls, the need for collaboration within the consortium is now more important than ever.

Design decisions about an ac or dc power architecture for the electrical distribution system aboard an all-electric ship do not affect the physics-based modeling of the electrical, thermal resistive, and thermal management frameworks in *DTMS*. The final design of the thermal management network overlaying the electrical distribution system will be predicated on the thermal loads created by the electrical components. The number of converters and their configuration, as well as bulk heat loads they present, will be the driving force behind the configuration of a thermal management system. The efficiency of the electrical system, however, plays an integral role in determining the heat loads and the creation of the most efficient configuration of the thermal management network.

8.2 Future Improvements to the *DTMS* Framework

While the *DTMS* simulation environment continues to have great promise, several areas of improvement came to mind in the course of implementing the thermal-electrical networks. These areas not only stand to improve the overall fidelity of the electrical framework, but also potentially mitigate future issues associated with the performance of *DTMS*. These areas include time stepping, integration techniques, and model-level connections.

Increasing the size of a time step during a simulation decreases the calculations required to complete a simulation, thus decreasing simulation time. The thermal time constant is many orders of magnitude larger than the electrical time constant. Thus time steps for thermal fluid and resistive networks can range from 1 to many 10's of seconds. Electrical simulations in *DTMS* require time steps on the order of 0.0005 seconds. This small time step is crucial for obtaining transient information during a dynamic event in the electrical network. However, when the dynamic event is complete, it is no longer necessary to use the time-consuming and costly integration methods required to resolve electrical transients. Thus, introducing a method into the *DTMS* framework to adapt to the transients and increase the time step during steady state periods in the electrical network could significantly decrease runtimes to be on par with the current runtimes of the thermal fluid and resistive networks. In this way, the thermal-electrical feedback, which has been demonstrated to be of great importance, can remain within the *DTMS* framework while gaining fidelity in the simulation and simultaneously decreasing runtimes.

The integration method employed in the *DTMS* framework, which is the direct cause of slow runtimes, is an area that could improve the fidelity of the electrical simulations. The numerical integration technique will be of particular importance if future *DTMS* developers choose to integrate inertial or mass momentum into the mechanical, rotational, or hydraulic energy domains. A possible alternative is a solver that handles system-level differential equations rather than solving the system numerically.

The connection of models in a *DTMS* simulation is set in the main file. Once the simulation starts, the models cannot be reconfigured. In the zonal electrical benchmark run in [30], the converters within the zones were turned on and off over a period of 5 seconds. This capability does not exist currently in *DTMS*. A method for changing the resistance of the cables, which connect the converters, to large values (~10,000 ohms) equivalent to disconnection proved to cause instabilities in the simulation. This capability is essential to providing the Navy with a system-level simulation environment capable of addressing reconfigurability.

8.3 Consortium

The ESRDC has proven to be an asset in the development of the AES particularly since the universities have shifted towards system-level modeling and simulation. Because the challenges of the AES now span several energy domains, modeling and simulation must confront these challenges holistically. As the energy requirements of the AES increase due to emerging technologies, so too must the capabilities of the thermal management system increase. The thermal-electrical co-simulations presented in this thesis demonstrate how the electrical distribution system and thermal management network must work harmoniously to ensure system stability.

Collaboration among members of the consortium is important for continuing to understand the dynamic nature of the AES and how major systems interact. A mutual understanding of how the various energy domains interact with each other is vital to system-level simulation. While each university research group clearly employs highly competent specialists, each group also has limited knowledge of other systems and subsystems of the AES. Collaboration will only enhance maturing simulation environments such as *DTMS*. Though *DTMS* has proven to effectively model and simulate system-level thermal management from the chiller to the heat loads of system-level electrical distribution, work must continue in the area of controls and program fidelity.

The majority of the work contained in this thesis is validated by analytical solutions and prior work within the consortium, but the availability of data regarding the performance of shipboard systems and subsystems would only enhance the accuracy of the models in the computer-based simulation environments. Consortium activities, such as a recent website for sharing research results, would be very helpful for distributing benchmark data and findings from consortium members and the Navy itself.

Appendix A: Nomenclature

Note: Unless indicated otherwise, variables may be assumed dimensionless.

Variables

A area [m^2]
B flux density [weber/m^2]
C capacitance [farad]
C conductance matrix
c specific heat [kJ/kg·K]
d line duty cycle
D diameter [m]
D duty cycle
D_o dc bias
E energy [J]
F flow vector
f frequency [s^{-1}]
g gravity [m/s^2]
h harmonic integer
H head vector
h time step [s]
h convection coefficient [W/m^2·K]
i current [A]
J Jacobian matrix
K constant
k thermal conductivity [W/m·K]
L length [m]
L inductance [H]
l length [m]
m mass [kg]
N number of fins
Nu Nusselt number
p pulse per cycle
P pressure [N/m^2]
P power, real power [W]
pf power factor
Pr Prandtl number
q charge [coulombs]
q heat [J]
Q heat [J]
Q recovered charge [coulomb]
Q reactive power [var]
R resistance [ohms]
Re Reynolds number
S apparent power [VA]
T temperature [K]
T period [s]
t time [s]
U internal energy [J]
v velocity [m/s]
V voltage [V]
W work [J]
W width [m]
z position [m]
Z impedance [ohms]

α thermal diffusivity [m^2/s]
η efficiency
Φ phase angle [deg]
λ flux linkage [weber-turns]
μ dynamic viscosity [N·s/m^2]
ρ density [kg/m^3]
ρ electrical conductivity [S/m]
σ thermal conductivity [W/m·K]
τ transistor period [s]
ω frequency [s^{-1}]

Subscripts

ac	alternating current
b	base
c	cross sectional
c	corrected
C	capacitance
cu,loss	copper loss
core	core
D	Dittus-Boelter
dc	direct current
DielectricLoss	dielectric loss
e	effective
f	fin
fund	fundamental
h	hydraulic
h	harmonic integer
harm	harmonic
i	index
in	in
L	line
l	lost
L	inductance
Loss	losses
m	discretized element
M	modulation
on	on
out	out
p	profile
p	time (superscript)
p	peak
p	primary
r	reverse recovery
R	resistance
ref	reference
rms	root mean squared
sw	switch
s	secondary
surf	surface
T	period
TECLoss	total eddy current loss
thd	total harmonic distortion
total	total
x	x-direction
x	frequency exponent (superscript)
y	flux density exponent (superscript)

Acronyms

AES	All-Electric Ship
CM	Converter Module
DTMS	Dynamic Thermal Modeling and Simulation
ESRDC	Electric Ship Research and Development Consortium
InM	Inverter Module
IPS	Integrated Power Systems
MVDC	Medium Voltage Direct Current
ONR	Office of Naval Research
PCM	Power Conversion Module
PDM	Power Distribution Module
PS	Power Supply

Appendix B: Benchmark Simulation Main File: Benchmark.cpp

```
#include "../../DTMSFramework.h"

#include <iomanip>
#include <iostream>
#include <ctime>
#include <cmath>

using namespace std;
using namespace DTMSFramework;

int main()
{

    //Boundary Nodes
    //----------------------------------------------------------------
    //Inlet Voltage AC
    ACElectricalEffortModel portSource("PSource");
    portSource.setAmplitude(792);
    portSource.setDependent(false);

    ACElectricalEffortModel starboardSource("SBSource");
    starboardSource.setAmplitude(679);
    starboardSource.setDependent(false);

    //Outlet Ground
    ElectricalEffortModel ground("Ground");
    ground.setVoltage(0);
    ground.setDependent(false);

    //Loads
    //----------------------------------------------------------------
    Resistor LB1("LB1");
    LB1.setResistance(56);
    LB1.setEvent(6, 23);
    LB1.setEvent(7, 11);
    Resistor LB2("LB1");
    LB2.setResistance(56);
    LB2.setEvent(7, 23);
    LB2.setEvent(8, 11);
    Resistor LB3("LB3");
    LB3.setResistance(56);
    LB3.setEvent(8, 23);
    LB3.setEvent(9, 11);
    Resistor load1("Load1");
    load1.setResistance(100);
    Resistor load2("Load2");
    load2.setResistance(100);
    Resistor load3("Load3");
    load3.setResistance(100);
    //Cables
    Resistor portBus("PortBus");
    portBus.setResistance(0.0002);
    Resistor starboardBus("StarboardBus");
    starboardBus.setResistance(0.0002);
    Resistor cable1("Cable1");
    cable1.setResistance(0.0002);
    Resistor cable2("Cable2");
    cable2.setResistance(0.0002);
    Resistor cable3("Cable3");
    cable3.setResistance(0.0002);
    Resistor cable4("Cable4");
    cable4.setResistance(0.0002);
    Resistor cable5("Cable5");
    cable5.setResistance(0.0002);
    Resistor cable6("Cable6");
    cable6.setResistance(0.0002);
    Resistor cable7("Cable7");
    cable7.setResistance(0.0002);
    Resistor cable8("Cable8");
    cable8.setResistance(0.0002);
    Resistor cable9("Cable9");
    cable9.setResistance(0.0002);
    Resistor cable10("Cable10");
    cable10.setResistance(0.0002);
    Resistor cable11("Cable11");
    cable11.setResistance(0.0002);
    Resistor cable12("Cable12");
    cable12.setResistance(0.0002);

    //Converters
    //--------------------------------------------------------------
    //Port PS1 and Starboard PS2
    PCM4 PS1("_PS1");
    PS1.setPowerFactor(0.96);
    PS1.setTurnsRatio(0.632);
    PS1.setInductance(0.00001);
    PS1.setSnubberCapacitance(0.00006);
    PS1.setSnubberResistance(50);
    PS1.setInitialInductorFluxLinkage(0.00018);
    PS1.setInitialCapacitorCharge(0.03);
    PS1.setInitialLossFreeResistance(10);
    PS1.setUpdateFrequency(50);
    PCM4 PS2("_PS2");
    PS2.setPowerFactor(0.96);
    PS2.setTurnsRatio(0.738);
    PS2.setInductance(0.00001);
    PS2.setSnubberCapacitance(0.00006);
    PS2.setSnubberResistance(50);
    PS2.setInitialInductorFluxLinkage(0.00018);
    PS2.setInitialCapacitorCharge(0.03);
    PS2.setInitialLossFreeResistance(10);
    PS2.setUpdateFrequency(50);

    //Port CM
    PCM1 SSCM_1("_SSCM_1");
    SSCM_1.setPowerFactor(0.96);
    SSCM_1.setInductance(0.05);
    SSCM_1.setSnubberCapacitance(0.00000005);
    SSCM_1.setSnubberResistance(5000);
    SSCM_1.setInitialInductorFluxLinkage(0.14);
    SSCM_1.setDutyCycle(0.84);
    SSCM_1.setInitialCapacitorCharge(0.000021);
    PCM1 SSCM_2("_SSCM_2");
    SSCM_2.setPowerFactor(0.96);
    SSCM_2.setInductance(0.05);
    SSCM_2.setSnubberCapacitance(0.00000005);
    SSCM_2.setSnubberResistance(5000);
    SSCM_2.setInitialInductorFluxLinkage(0.14);
    SSCM_2.setDutyCycle(0.84);
    SSCM_2.setInitialCapacitorCharge(0.000021);
    PCM1 SSCM_3("_SSCM_3");
    SSCM_3.setPowerFactor(0.96);
    SSCM_3.setInductance(0.05);
    SSCM_3.setSnubberCapacitance(0.00000005);
    SSCM_3.setSnubberResistance(5000);
    SSCM_3.setInitialInductorFluxLinkage(0.14);
    SSCM_3.setDutyCycle(0.84);
    SSCM_3.setInitialCapacitorCharge(0.000021);

    //Starboard CM
    PCM1 SSCM_4("_SSCM_4");
    SSCM_4.setPowerFactor(0.96);
    SSCM_4.setInductance(0.05);
    SSCM_4.setSnubberCapacitance(0.00000005);
    SSCM_4.setSnubberResistance(5000);
    SSCM_4.setInitialInductorFluxLinkage(0.14);
    SSCM_4.setDutyCycle(0.84);
    SSCM_4.setInitialCapacitorCharge(0.000021);
    PCM1 SSCM_5("_SSCM_5");
    SSCM_5.setPowerFactor(0.96);
    SSCM_5.setInductance(0.05);
    SSCM_5.setSnubberCapacitance(0.00000005);
    SSCM_5.setSnubberResistance(5000);
    SSCM_5.setInitialInductorFluxLinkage(0.14);
    SSCM_5.setDutyCycle(0.84);
    SSCM_5.setInitialCapacitorCharge(0.000021);
    PCM1 SSCM_6("_SSCM_6");   SSCM_6.setPowerFactor(0.96);
    SSCM_6.setPowerFactor(0.96);
    SSCM_6.setInductance(0.05);
    SSCM_6.setSnubberCapacitance(0.00000005);
    SSCM_6.setSnubberResistance(5000);
    SSCM_6.setInitialInductorFluxLinkage(0.14);
    SSCM_6.setDutyCycle(0.84);
    SSCM_6.setInitialCapacitorCharge(0.000021);

    //InM
    PCM2 SSIM_1("_SSIM_1");
    SSIM_1.setPowerFactor(0.96);
    SSIM_1.setInductance(0.2);
    SSIM_1.setSnubberCapacitance(0.0000005);
    SSIM_1.setSnubberResistance(500);
    SSIM_1.setInitialInductorFluxLinkage(0.9);
    SSIM_1.setInitialLossFreeResistance(157);
    SSIM_1.setInitialCapacitorCharge(0.00012);
    SSIM_1.setUpdateFrequency(50);
    PCM2 SSIM_2("_SSIM_2");
    SSIM_2.setPowerFactor(0.96);
    SSIM_2.setInductance(0.2);
    SSIM_2.setSnubberCapacitance(0.0000005);
    SSIM_2.setSnubberResistance(500);
    SSIM_2.setInitialInductorFluxLinkage(0.9);
    SSIM_2.setInitialLossFreeResistance(157);
    SSIM_2.setInitialCapacitorCharge(0.00012);
    SSIM_2.setUpdateFrequency(50);
    PCM2 SSIM_3("_SSIM_3");
    SSIM_3.setPowerFactor(0.96);
    SSIM_3.setInductance(0.2);
    SSIM_3.setSnubberCapacitance(0.0000005);
    SSIM_3.setSnubberResistance(500);
    SSIM_3.setInitialInductorFluxLinkage(0.9);
    SSIM_3.setInitialLossFreeResistance(157);
    SSIM_3.setInitialCapacitorCharge(0.00012);
    SSIM_3.setUpdateFrequency(50);

    //Connections
    //-----------------------------------------------------------------
    //PCM-1 to PCM-2
    portSource.addOutletFlowModel(portBus);
    starboardSource.addOutletFlowModel(starboardBus);
    PS1.addInletFlowModel(portBus);
    PS2.addInletFlowModel(starboardBus);
    PS1.addOutletFlowModel(cable1);
    PS1.addOutletFlowModel(cable2);
    PS1.addOutletFlowModel(cable3);
    PS2.addOutletFlowModel(cable4);
    PS2.addOutletFlowModel(cable5);
    PS2.addOutletFlowModel(cable6);
    SSCM_1.addInletFlowModel(cable1);
    SSCM_2.addInletFlowModel(cable2);
    SSCM_3.addInletFlowModel(cable3);
    SSCM_4.addInletFlowModel(cable4);
    SSCM_5.addInletFlowModel(cable5);
    SSCM_6.addInletFlowModel(cable6);
    SSCM_1.addOutletFlowModel(cable7);
    SSCM_1.addOutletFlowModel(load1);
    SSCM_2.addOutletFlowModel(cable8);
    SSCM_2.addOutletFlowModel(load3);
    SSCM_3.addOutletFlowModel(cable9);
    SSCM_3.addOutletFlowModel(load2);
    SSCM_4.addOutletFlowModel(cable10);
    SSCM_5.addOutletFlowModel(cable11);
    SSCM_6.addOutletFlowModel(cable12);
    SSIM_1.addInletFlowModel(cable8);
    SSIM_1.addInletFlowModel(cable11);
    SSIM_2.addInletFlowModel(cable9);
    SSIM_2.addInletFlowModel(cable12);
    SSIM_3.addInletFlowModel(cable7);
    SSIM_3.addInletFlowModel(cable10);

    //PCM-1 and PCM-2 to loads
    SSIM_1.addOutletFlowModel(LB1);
    SSIM_2.addOutletFlowModel(LB2);
    SSIM_3.addOutletFlowModel(LB3);

    //Grounding
    PS1.setOutletEffortModel(ground);
    PS2.setOutletEffortModel(ground);
    SSCM_1.setOutletEffortModel(ground);
    SSCM_2.setOutletEffortModel(ground);
    SSCM_3.setOutletEffortModel(ground);
    SSCM_4.setOutletEffortModel(ground);
    SSCM_5.setOutletEffortModel(ground);
    SSCM_6.setOutletEffortModel(ground);
    SSIM_1.setOutletEffortModel(ground);
    SSIM_2.setOutletEffortModel(ground);
    SSIM_3.setOutletEffortModel(ground);
    LB1.setOutletEffortModel(ground);
    LB2.setOutletEffortModel(ground);
    LB3.setOutletEffortModel(ground);
    load1.setOutletEffortModel(ground);
    load2.setOutletEffortModel(ground);
    load3.setOutletEffortModel(ground);

    //Simulation Set-up
    //-----------------------------------------------------------------
    DTMSSimulation SimExec("Benchmark.csv", 10, 0.0005, 0.005) ;

    // Create solver object
    GloballyConvergentResistiveSolver* simulationSolver = new
GloballyConvergentResistiveSolver() ;

    // Set solver error tolerance
    simulationSolver->setErrorTolerance(1e-4) ;

    // Add solver to simulation executive
    SimExec.addSolver(simulationSolver);

    // Add models into solver
    simulationSolver->addModel(portSource);
    simulationSolver->addModel(starboardSource);
    simulationSolver->addModel(portBus);
    simulationSolver->addModel(starboardBus);
    simulationSolver->addModel(PS1);
    simulationSolver->addModel(PS2);
    simulationSolver->addModel(cable1);
    simulationSolver->addModel(cable2);
    simulationSolver->addModel(cable3);
    simulationSolver->addModel(cable4);
    simulationSolver->addModel(cable5);
    simulationSolver->addModel(cable6);
    simulationSolver->addModel(SSCM_1);
    simulationSolver->addModel(SSCM_2);
    simulationSolver->addModel(SSCM_3);
    simulationSolver->addModel(SSCM_4);
    simulationSolver->addModel(SSCM_5);
    simulationSolver->addModel(SSCM_6);
    simulationSolver->addModel(cable7);
    simulationSolver->addModel(cable8);
    simulationSolver->addModel(cable9);
    simulationSolver->addModel(cable10);
    simulationSolver->addModel(cable11);
    simulationSolver->addModel(cable12);
    simulationSolver->addModel(SSIM_1);
    simulationSolver->addModel(SSIM_2);
    simulationSolver->addModel(SSIM_3);
    simulationSolver->addModel(LB1);
    simulationSolver->addModel(LB2);
    simulationSolver->addModel(LB3);
    simulationSolver->addModel(load1);
    simulationSolver->addModel(load2);
    simulationSolver->addModel(load3);
    simulationSolver->addModel(ground);

    // Add models to simulation executive
    SimExec.addModel(starboardSource);
    SimExec.addModel(portSource);
    SimExec.addModel(starboardBus);
    SimExec.addModel(portBus);
    SimExec.addModel(PS1);
    SimExec.addModel(PS2);
    SimExec.addModel(cable1);
    SimExec.addModel(cable2);
    SimExec.addModel(cable3);
    SimExec.addModel(cable4);
    SimExec.addModel(cable5);
    SimExec.addModel(cable6);
    SimExec.addModel(SSCM_1);
    SimExec.addModel(SSCM_2);
    SimExec.addModel(SSCM_3);
    SimExec.addModel(SSCM_4);
    SimExec.addModel(SSCM_5);
    SimExec.addModel(SSCM_6);
    SimExec.addModel(cable7);
    SimExec.addModel(cable8);
    SimExec.addModel(cable9);
    SimExec.addModel(cable10);
    SimExec.addModel(cable11);
    SimExec.addModel(cable12);
    SimExec.addModel(SSIM_1);
    SimExec.addModel(SSIM_2);
    SimExec.addModel(SSIM_3);
    SimExec.addModel(LB1);
    SimExec.addModel(LB2);
    SimExec.addModel(LB3);
    SimExec.addModel(load1);
    SimExec.addModel(load2);
    SimExec.addModel(load3);
    SimExec.addModel(ground);

    //Set Write Flags
    portSource.setWriteFlag(1);
    starboardSource.setWriteFlag(1);
    portBus.setWriteFlag(1);
    starboardBus.setWriteFlag(1);
    PS1.setWriteFlag(1);
    PS2.setWriteFlag(1);
    cable1.setWriteFlag(1);
    cable2.setWriteFlag(1);
    cable3.setWriteFlag(1);
    cable4.setWriteFlag(1);
    cable5.setWriteFlag(1);
    cable6.setWriteFlag(1);
    cable7.setWriteFlag(1);
    cable8.setWriteFlag(1);
    cable9.setWriteFlag(1);
    cable10.setWriteFlag(1);
    cable11.setWriteFlag(1);
    cable12.setWriteFlag(1);
    SSCM_1.setWriteFlag(1);
    SSCM_2.setWriteFlag(1);
    SSCM_3.setWriteFlag(1);
    SSCM_4.setWriteFlag(1);
    SSCM_5.setWriteFlag(1);
    SSCM_6.setWriteFlag(1);
    SSIM_1.setWriteFlag(1);
    SSIM_2.setWriteFlag(1);
    SSIM_3.setWriteFlag(1);
    LB1.setWriteFlag(1);
    LB2.setWriteFlag(1);
    LB3.setWriteFlag(1);
    ground.setWriteFlag(1);

    //Begin Simulation
    cout << "simulating... " << endl ;

    //Use a clock to measure elapsed time
    clock_t cBegin, cEnd ;
    double cSim ;
    cBegin = clock() ;

    SimExec.runSimulation() ;

    cEnd = clock() ;
    cSim = (cEnd-cBegin) ;
    cSim = cSim/CLOCKS_PER_SEC ;

    cout << endl << "done" << endl << endl;

    cout << "Real Time (elapsed): " << cSim << endl ;
    cout << endl;

    //Clean up allocated memory
    //N/A
}
```

Appendix C: Thermally Dependent Simulation Main File: ThermalDependence.cpp

```
#include "../../DTMSFramework.h"
#include "../../ResistiveNetwork/Electrical/TemperatureEffortModel.h"
#include "../../ResistiveNetwork/Electrical/PlenumHeatExchanger.h"

#include <iomanip>
#include <iostream>
#include <ctime>
#include <cmath>

using namespace std;
using namespace DTMSFramework;

int main()
{
    //***THERMAL MODELS***///

    // Create fluid
    CaloricallyPerfectAir workingFluid_;

    // Create bay fan
    CentrifugalPump FanPCM1("_FanPCM1") ;
    FanPCM1.setDesignDensity(1.2);
    FanPCM1.setDesignFlowRate(0.052 ) ;
    FanPCM1.setDesignPressureDifference(54) ;
    FanPCM1.setMaximumPressureDifference(1000) ;
    CentrifugalPump FanPCM2("_FanPCM2") ;
    FanPCM2.setDesignDensity(1.2);
    FanPCM2.setDesignFlowRate(0.182) ;
    FanPCM2.setDesignPressureDifference(54) ;
    FanPCM2.setMaximumPressureDifference(1000) ;

    //Pressure Nodes
    //-------------------------------------------------------------
    ThermalReservoir pIn_("_CoolIn");
    pIn_.setEffort(101625);
    pIn_.setDependent(false);
    pIn_.set(ENTHALPY, 318);

    ThermalFluidEffortModel pMidPCM1_("_pMidPCM1") ;
    ThermalFluidEffortModel pMidPCM2_("_pMidPCM2") ;

    ThermalFluidEffortModel pOut_("_HotOut");
    pOut_.setDependent(false);
    pOut_.setEffort(101625);

    // Plenum HeatExchange
    //-----------------------------------------------------------------
    PlenumHeatExchanger plenumHXerPCM1_("_HXerPCM1");
    plenumHXerPCM1_.setPlenumHeight(0.01);
    plenumHXerPCM1_.setPlenumLength(0.617);
    plenumHXerPCM1_.setPlenumWidth(0.221);
    plenumHXerPCM1_.setTemperature(317);
    PlenumHeatExchanger plenumHXerPCM2_("_HXerPCM2");
    plenumHXerPCM2_.setPlenumHeight(0.01);
    plenumHXerPCM2_.setPlenumLength((0.8));
    plenumHXerPCM2_.setPlenumWidth(0.249);
    plenumHXerPCM2_.setTemperature(317);

    ///***Heat Flow***///

    //Temperature Node
    //------------------------------------------------------------
    TemperatureEffortModel tempPCM1("_tempPCM1");
    tempPCM1.setTemperature(317);
    tempPCM1.setDependent(false);
    TemperatureEffortModel tempPCM2("_tempPCM2");
    tempPCM2.setTemperature(317);
    tempPCM2.setDependent(false);

    //PCM Internal Heat Generation and Storage
    //-------------------------------------------------------------
    InternalHeatGenerationModel PCM1_Heat("_PCM1_Heat");
    PCM1_Heat.setTemperature(317);
    PCM1_Heat.setCompositeSpecificHeat(385);
    PCM1_Heat.setDensity(8960);
    PCM1_Heat.setThermalConductivity(400);
    PCM1_Heat.setLength(0.617);
    PCM1_Heat.setWidth(0.221);
    PCM1_Heat.setHeight(0.29);
    PCM1_Heat.setDiscretizations(6);
    InternalHeatGenerationModel PCM2_Heat("_PCM2_Heat");
    PCM2_Heat.setTemperature(317);
    PCM2_Heat.setCompositeSpecificHeat(385);
    PCM2_Heat.setDensity(8960);
    PCM2_Heat.setThermalConductivity(400);
    PCM2_Heat.setLength(0.8);
    PCM2_Heat.setWidth(0.249);
    PCM2_Heat.setHeight(0.315);
    PCM2_Heat.setDiscretizations(6);

    //Connections
    //-------------------------------------------------------------
    FanPCM1.setInletEffortModel(pIn_);
    FanPCM2.setInletEffortModel(pIn_);

    FanPCM1.setOutletEffortModel(pMidPCM1_);
    FanPCM2.setOutletEffortModel(pMidPCM2_);

    plenumHXerPCM1_.setPlenumInletEffortModel(pMidPCM1_);
    plenumHXerPCM2_.setPlenumInletEffortModel(pMidPCM2_);

    plenumHXerPCM1_.setPlenumOutletEffortModel(pOut_);
    plenumHXerPCM2_.setPlenumOutletEffortModel(pOut_);

    PCM1_Heat.setOutletEffortModel(tempPCM1);
    PCM2_Heat.setOutletEffortModel(tempPCM2);

    plenumHXerPCM1_.setConvectionInletEffortModel(tempPCM1);
    plenumHXerPCM2_.setConvectionInletEffortModel(tempPCM2);

    workingFluid_.updatePropsPH(pIn_.getEffort(), pIn_.get(ENTHALPY))
;

    //Set working fluid
    pIn_.setFluid(workingFluid_);
    FanPCM1.setFluid(workingFluid_);
    FanPCM2.setFluid(workingFluid_);
    pMidPCM1_.setFluid(workingFluid_);
    pMidPCM2_.setFluid(workingFluid_);
    plenumHXerPCM1_.setFluid(workingFluid_);
    plenumHXerPCM2_.setFluid(workingFluid_);
    pOut_.setFluid(workingFluid_);

    //***ELECTRICAL MODELS***//

    //Boundary Nodes
    //-----------------------------------------------------------
    //Inlet Voltage
    ElectricalEffortModel dcSource("_dcSource");
    dcSource.setVoltage(500);
    dcSource.setDependent(false);

    //Outlet Ground
    ElectricalEffortModel ground("_Ground");
    ground.setVoltage(0);
    ground.setDependent(false);

    //Cables
    //---------------------------------------------------------------
    Resistor cable1("_Cable1");
    cable1.setResistance(0.0002);
    Resistor cable2("_Cable2");
    cable2.setResistance(0.0002);

    //Converters
    //---------------------------------------------------------------
    PCM1 SSCM_("_SSCM");
    SSCM_.setPowerFactor(0.8);
    SSCM_.setInductance(0.05);
    SSCM_.setSnubberCapacitance(0.00000005);
    SSCM_.setSnubberResistance(5000);
    SSCM_.setInitialInductorFluxLinkage(0.14);
    SSCM_.setDutyCycle(0.84);
    SSCM_.setInitialCapacitorCharge(0.000021);
    SSCM_.setUpdateFrequency(500);
    SSCM_.setInternalHeatGenerationModel(PCM1_Heat);
    SSCM_.setThermalDependency(true);

    PCM2 SSIM_("_SSIM");
    SSIM_.setPowerFactor(0.8);
    SSIM_.setInductance(0.2);
    SSIM_.setSnubberCapacitance(0.0000005);
    SSIM_.setSnubberResistance(500);
    SSIM_.setInitialInductorFluxLinkage(0.9);
    SSIM_.setInitialLossFreeResistance(157);
    SSIM_.setInitialCapacitorCharge(0.00012);
    SSIM_.setUpdateFrequency(500);
    SSIM_.setInternalHeatGenerationModel(PCM2_Heat);
    SSIM_.setThermalDependency(true);

    //Loads
    //-----------------------------------------------------------------
    Resistor LB1("_LB1");
    LB1.setResistance(25);
    LB1.setEvent(30, 0.44);

    //Electrical Connections
    //-------------------------------------------------------------
    dcSource.addOutletFlowModel(cable1);
    SSCM_.addInletFlowModel(cable1);
    SSCM_.addOutletFlowModel(cable2);
    SSIM_.addInletFlowModel(cable2);

    //PCM-1 and PCM-2 to loads
    SSIM_.addOutletFlowModel(LB1);

    //Grounding
    SSCM_.setOutletEffortModel(ground);
    SSIM_.setOutletEffortModel(ground);
    LB1.setOutletEffortModel(ground);

    //Simulation Set-up
    //---------------------------------------------------------------

    DTMSSimulation SimExec("ThermalDependence.csv", 7200, 0.0005, 5)
;

    // Create solver object
    GloballyConvergentResistiveSolver* simulationSolver = new
GloballyConvergentResistiveSolver() ;

    // Set solver error tolerance
    simulationSolver->setErrorTolerance(1e-4) ;

    // Add solver to simulation executive
    SimExec.addSolver(simulationSolver);

    // Add models into solver
    //Electrical
    simulationSolver->addModel(dcSource);
    simulationSolver->addModel(cable1);
    simulationSolver->addModel(SSCM_);
    simulationSolver->addModel(cable2);
    simulationSolver->addModel(SSIM_);
    simulationSolver->addModel(LB1);
    simulationSolver->addModel(ground);
    //Heat and Fluid
    simulationSolver->addModel(pIn_);
    simulationSolver->addModel(FanPCM1);
    simulationSolver->addModel(FanPCM2);
    simulationSolver->addModel(pMidPCM1_);
    simulationSolver->addModel(pMidPCM2_);
    simulationSolver->addModel(plenumHXerPCM1_);
    simulationSolver->addModel(plenumHXerPCM2_);
    simulationSolver->addModel(pOut_);
    simulationSolver->addModel(tempPCM1);
    simulationSolver->addModel(tempPCM2);
    simulationSolver->addModel(PCM1_Heat);
    simulationSolver->addModel(PCM2_Heat);

    // Add models to simulation executive
    //Electrical
    SimExec.addModel(dcSource);
    SimExec.addModel(cable1);
    SimExec.addModel(SSCM_);
    SimExec.addModel(cable2);
    SimExec.addModel(SSIM_);
    SimExec.addModel(LB1);
    SimExec.addModel(ground);
    //Heat and Fluid
    SimExec.addModel(pIn_);
    SimExec.addModel(FanPCM1);
    SimExec.addModel(FanPCM2);
    SimExec.addModel(pMidPCM1_);
    SimExec.addModel(pMidPCM2_);
    SimExec.addModel(plenumHXerPCM1_);
    SimExec.addModel(plenumHXerPCM2_);
    SimExec.addModel(pOut_);
    SimExec.addModel(tempPCM1);
    SimExec.addModel(tempPCM2);
    SimExec.addModel(PCM1_Heat);
    SimExec.addModel(PCM2_Heat);

    //Set Write Flags
    //Electrical
    dcSource.setWriteFlag(1);
    cable1.setWriteFlag(1);
    SSCM_.setWriteFlag(1);
    cable2.setWriteFlag(1);
    SSIM_.setWriteFlag(1);
    LB1.setWriteFlag(1);
    ground.setWriteFlag(1);
    //Heat and Fluid
    pIn_.setWriteFlag(1);
    FanPCM1.setWriteFlag(1);
    FanPCM2.setWriteFlag(1);
    pMidPCM1_.setWriteFlag(1);
    pMidPCM2_.setWriteFlag(1);
    plenumHXerPCM1_.setWriteFlag(1);
    plenumHXerPCM2_.setWriteFlag(1);
    pOut_.setWriteFlag(1);
    PCM1_Heat.setWriteFlag(1);
    PCM2_Heat.setWriteFlag(1);
    tempPCM1.setWriteFlag(1);
    tempPCM2.setWriteFlag(1);


    //Begin Simulation
    cout << "simulating... " << endl ;

    //Use a clock to measure elapsed time
    clock_t cBegin, cEnd ;
    double cSim ;
    cBegin = clock() ;

    SimExec.runSimulation() ;

    cEnd = clock() ;
    cSim = (cEnd-cBegin) ;
    cSim = cSim/CLOCKS_PER_SEC ;

    cout << endl << "done" << endl << endl;

    cout << "Real Time (elapsed): " << cSim << endl ;
    cout << endl;

    //Clean up allocated memory
    //N/A
}
```

Appendix D: Multiple Zone Simulation Main File: MultipleZone.cpp

```
#include "../../DTMSFramework.h"
#include "../../ResistiveNetwork/Electrical/TemperatureEffortModel.h"
#include "../../ResistiveNetwork/Electrical/PlenumHeatExchanger.h"

#include <iomanip>
#include <iostream>
#include <ctime>
#include <cmath>

using namespace std;
using namespace DTMSFramework;

int main()
{
    //***THERMAL MODELS***///

    // Create fluid
    CaloricallyPerfectAir workingFluid_;

    // Create bay fan
    CentrifugalPump FanPS1("_FanPS1") ;
    FanPS1.setDesignDensity(1.2);
    FanPS1.setDesignFlowRate(1.5) ;
    FanPS1.setDesignPressureDifference(54) ;
    FanPS1.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanPS2("_FanPS2") ;
    FanPS2.setDesignDensity(1.2);
    FanPS2.setDesignFlowRate(1.5) ;
    FanPS2.setDesignPressureDifference(54) ;
    FanPS2.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM1("_FanCM1") ;
    FanCM1.setDesignDensity(1.2);
    FanCM1.setDesignFlowRate(0.052) ;
    FanCM1.setDesignPressureDifference(54) ;
    FanCM1.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM2("_FanCM2") ;
    FanCM2.setDesignDensity(1.2);
    FanCM2.setDesignFlowRate(0.052) ;
    FanCM2.setDesignPressureDifference(54) ;
    FanCM2.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM3("_FanCM3") ;
    FanCM3.setDesignDensity(1.2);
    FanCM3.setDesignFlowRate(0.052) ;
    FanCM3.setDesignPressureDifference(54) ;
    FanCM3.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM4("_FanCM4") ;
    FanCM4.setDesignDensity(1.2);
    FanCM4.setDesignFlowRate(0.052) ;
    FanCM4.setDesignPressureDifference(54) ;
    FanCM4.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM5("_FanCM5") ;
    FanCM5.setDesignDensity(1.2);
    FanCM5.setDesignFlowRate(0.052) ;
    FanCM5.setDesignPressureDifference(54) ;
    FanCM5.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanCM6("_FanCM6") ;
    FanCM6.setDesignDensity(1.2);
    FanCM6.setDesignFlowRate(0.052) ;
    FanCM6.setDesignPressureDifference(54) ;
    FanCM6.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanInM1("_FanInM1") ;
    FanInM1.setDesignDensity(1.2);
    FanInM1.setDesignFlowRate(0.185) ;
    FanInM1.setDesignPressureDifference(54) ;
    FanInM1.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanInM2("_FanInM2") ;
    FanInM2.setDesignDensity(1.2);
    FanInM2.setDesignFlowRate(0.185) ;
    FanInM2.setDesignPressureDifference(54) ;
    FanInM2.setMaximumPressureDifference(10000) ;
    CentrifugalPump FanInM3("_FanInM3") ;
    FanInM3.setDesignDensity(1.2);
    FanInM3.setDesignFlowRate(0.185) ;
    FanInM3.setDesignPressureDifference(54) ;
    FanInM3.setMaximumPressureDifference(10000) ;

    //Pressure Nodes
    //-------------------------------------------------------------
    ThermalReservoir pIn_("_CoolIn");
    pIn_.setEffort(101625);
    pIn_.setDependent(false);
    pIn_.set(ENTHALPY, 318);

    ThermalFluidEffortModel pMidPS1_("_pMidPS1") ;
    ThermalFluidEffortModel pMidPS2_("_pMidPS2") ;
    ThermalFluidEffortModel pMidCM1_("_pMidCM1") ;
    ThermalFluidEffortModel pMidCM2_("_pMidCM2") ;
    ThermalFluidEffortModel pMidCM3_("_pMidCm3") ;
    ThermalFluidEffortModel pMidCM4_("_pMidCM4") ;
    ThermalFluidEffortModel pMidCM5_("_pMidCM5") ;
    ThermalFluidEffortModel pMidCM6_("_pMidCM6") ;
    ThermalFluidEffortModel pMidInM1_("_pMidInM1") ;
    ThermalFluidEffortModel pMidInM2_("_pMidInM2") ;
    ThermalFluidEffortModel pMidInM3_("_pMidInM3") ;

    ThermalFluidEffortModel pOut_("_HotOut");
    pOut_.setDependent(false);
    pOut_.setEffort(101625);

    // Plenum HeatExchange
    //-----------------------------------------------------------------
    PlenumHeatExchanger plenumHXerPS1_("_HXerPS1");
    plenumHXerPS1_.setPlenumHeight(0.01);
    plenumHXerPS1_.setPlenumLength(0.8);
    plenumHXerPS1_.setPlenumWidth(0.3);
    plenumHXerPS1_.setTemperature(317);
    PlenumHeatExchanger plenumHXerPS2_("_HXerPS2");
    plenumHXerPS2_.setPlenumHeight(0.01);
    plenumHXerPS2_.setPlenumLength(0.8);
    plenumHXerPS2_.setPlenumWidth(0.3);
    plenumHXerPS2_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM1_("_HXerCM1");
    plenumHXerCM1_.setPlenumHeight(0.01);
    plenumHXerCM1_.setPlenumLength(0.617);
    plenumHXerCM1_.setPlenumWidth(0.221);
    plenumHXerCM1_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM2_("_HXerCM2");
    plenumHXerCM2_.setPlenumHeight(0.01);
    plenumHXerCM2_.setPlenumLength(0.617);
    plenumHXerCM2_.setPlenumWidth(0.221);
    plenumHXerCM2_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM3_("_HXerCM3");
    plenumHXerCM3_.setPlenumHeight(0.01);
    plenumHXerCM3_.setPlenumLength(0.617);
    plenumHXerCM3_.setPlenumWidth(0.221);
    plenumHXerCM3_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM4_("_HXerCM4");
    plenumHXerCM4_.setPlenumHeight(0.01);
    plenumHXerCM4_.setPlenumLength(0.617);
    plenumHXerCM4_.setPlenumWidth(0.221);
    plenumHXerCM4_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM5_("_HXerCM5");
    plenumHXerCM5_.setPlenumHeight(0.01);
    plenumHXerCM5_.setPlenumLength(0.617);
    plenumHXerCM5_.setPlenumWidth(0.221);
    plenumHXerCM5_.setTemperature(317);
    PlenumHeatExchanger plenumHXerCM6_("_HXerCM6");
    plenumHXerCM6_.setPlenumHeight(0.01);
    plenumHXerCM6_.setPlenumLength(0.617);
    plenumHXerCM6_.setPlenumWidth(0.221);
    plenumHXerCM6_.setTemperature(317);
    PlenumHeatExchanger plenumHXerInM1_("_HXerInM1");
    plenumHXerInM1_.setPlenumHeight(0.01);
    plenumHXerInM1_.setPlenumLength((0.8));
    plenumHXerInM1_.setPlenumWidth(0.249);
    plenumHXerInM1_.setTemperature(317);
    PlenumHeatExchanger plenumHXerInM2_("_HXerInM2");
    plenumHXerInM2_.setPlenumHeight(0.01);
    plenumHXerInM2_.setPlenumLength(0.8);
    plenumHXerInM2_.setPlenumWidth(0.249);
    plenumHXerInM2_.setTemperature(317);
    PlenumHeatExchanger plenumHXerInM3_("_HXerInM3");
    plenumHXerInM3_.setPlenumHeight(0.01);
    plenumHXerInM3_.setPlenumLength((0.8));
    plenumHXerInM3_.setPlenumWidth(0.249);
    plenumHXerInM3_.setTemperature(317);
    ///***Heat Flow***///

    //Temperature Node
    //------------------------------------------------------------
    TemperatureEffortModel tempPS1("_tempPS1");
    tempPS1.setTemperature(317);
    tempPS1.setDependent(false);
    TemperatureEffortModel tempPS2("_tempPS2");
    tempPS2.setTemperature(317);
    tempPS2.setDependent(false);
    TemperatureEffortModel tempCM1("_tempCM1");
    tempCM1.setTemperature(317);
    tempCM1.setDependent(false);
    TemperatureEffortModel tempCM2("_tempCM2");
    tempCM2.setTemperature(317);
    tempCM2.setDependent(false);
    TemperatureEffortModel tempCM3("_tempCM3");
    tempCM3.setTemperature(317);
    tempCM3.setDependent(false);
    TemperatureEffortModel tempCM4("_tempCM4");
    tempCM4.setTemperature(317);
    tempCM4.setDependent(false);
    TemperatureEffortModel tempCM5("_tempCM5");
    tempCM5.setTemperature(317);
    tempCM5.setDependent(false);
    TemperatureEffortModel tempCM6("_tempCM6");
    tempCM6.setTemperature(317);
    tempCM6.setDependent(false);
    TemperatureEffortModel tempInM1("_tempInM1")
    tempInM1.setTemperature(317);
    tempInM1.setDependent(false);
    TemperatureEffortModel tempInM2("_tempInM2");
    tempInM2.setTemperature(317);
    tempInM2.setDependent(false);
    TemperatureEffortModel tempInM3("_tempInM3");
    tempInM3.setTemperature(317);
    tempInM3.setDependent(false);

    //PCM Internal Heat Generation and Storage
    //----------------------------------------------------------------
    InternalHeatGenerationModel PS1_Heat("_PS1_Heat");
    PS1_Heat.setTemperature(317);
    PS1_Heat.setCompositeSpecificHeat(385);
    PS1_Heat.setDensity(8960);
    PS1_Heat.setThermalConductivity(400);
    PS1_Heat.setLength(0.8);
    PS1_Heat.setWidth(0.3);
    PS1_Heat.setHeight(0.4);
    PS1_Heat.setDiscretizations(6);
    InternalHeatGenerationModel PS2_Heat("_PS2_Heat");
    PS2_Heat.setTemperature(317);
    PS2_Heat.setCompositeSpecificHeat(385);
    PS2_Heat.setDensity(8960);
    PS2_Heat.setThermalConductivity(400);
    PS2_Heat.setLength(0.8);
    PS2_Heat.setWidth(0.3);
    PS2_Heat.setHeight(0.4);
    PS2_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM1_Heat("_CM1_Heat");
    CM1_Heat.setTemperature(317);
    CM1_Heat.setCompositeSpecificHeat(385);
    CM1_Heat.setDensity(8960);
    CM1_Heat.setThermalConductivity(400);
    CM1_Heat.setLength(0.617);
    CM1_Heat.setWidth(0.221);
    CM1_Heat.setHeight(0.29);
    CM1_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM2_Heat("_CM2_Heat");
    CM2_Heat.setTemperature(317);
    CM2_Heat.setCompositeSpecificHeat(385);
    CM2_Heat.setDensity(8960);
    CM2_Heat.setThermalConductivity(400);
    CM2_Heat.setLength(0.617);
    CM2_Heat.setWidth(0.221);
    CM2_Heat.setHeight(0.29);
    CM2_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM3_Heat("_CM3_Heat");
    CM3_Heat.setTemperature(317);
    CM3_Heat.setCompositeSpecificHeat(385);
    CM3_Heat.setDensity(8960);
    CM3_Heat.setThermalConductivity(400);
    CM3_Heat.setLength(0.617);
    CM3_Heat.setWidth(0.221);
    CM3_Heat.setHeight(0.29);
    CM3_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM4_Heat("_CM4_Heat");
    CM4_Heat.setTemperature(317);
    CM4_Heat.setCompositeSpecificHeat(385);
    CM4_Heat.setDensity(8960);
    CM4_Heat.setThermalConductivity(400);
    CM4_Heat.setLength(0.617);
    CM4_Heat.setWidth(0.221);
    CM4_Heat.setHeight(0.29);
    CM4_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM5_Heat("_CM5_Heat");
    CM5_Heat.setTemperature(317);
    CM5_Heat.setCompositeSpecificHeat(385);
    CM5_Heat.setDensity(8960);
    CM5_Heat.setThermalConductivity(400);
    CM5_Heat.setLength(0.617);
    CM5_Heat.setWidth(0.221);
    CM5_Heat.setHeight(0.29);
    CM5_Heat.setDiscretizations(6);
    InternalHeatGenerationModel CM6_Heat("_CM6_Heat");
    CM6_Heat.setTemperature(317);
    CM6_Heat.setCompositeSpecificHeat(385);
    CM6_Heat.setDensity(8960);
    CM6_Heat.setThermalConductivity(400);
    CM6_Heat.setLength(0.617);
    CM6_Heat.setWidth(0.221);
    CM6_Heat.setHeight(0.29);
    CM6_Heat.setDiscretizations(6);
    InternalHeatGenerationModel InM1_Heat("_InM1_Heat");
    InM1_Heat.setTemperature(317);
    InM1_Heat.setCompositeSpecificHeat(385);
    InM1_Heat.setDensity(8960);
    InM1_Heat.setThermalConductivity(400);
    InM1_Heat.setLength(0.8);
    InM1_Heat.setWidth(0.249);
    InM1_Heat.setHeight(0.315);
    InM1_Heat.setDiscretizations(6);
    InternalHeatGenerationModel InM2_Heat("_InM2_Heat");
    InM2_Heat.setTemperature(317);
    InM2_Heat.setCompositeSpecificHeat(385);
    InM2_Heat.setDensity(8960);
    InM2_Heat.setThermalConductivity(400);
    InM2_Heat.setLength(0.8);
    InM2_Heat.setWidth(0.249);
    InM2_Heat.setHeight(0.315);
    InM2_Heat.setDiscretizations(6);
    InternalHeatGenerationModel InM3_Heat("_InM3_Heat");
    InM3_Heat.setTemperature(317);
    InM3_Heat.setCompositeSpecificHeat(385);
    InM3_Heat.setDensity(8960);
    InM3_Heat.setThermalConductivity(400);
    InM3_Heat.setLength(0.8);
    InM3_Heat.setWidth(0.249);
    InM3_Heat.setHeight(0.315);
    InM3_Heat.setDiscretizations(6);

    //Connections
    //------------------------------------------------------------
    FanPS1.setInletEffortModel(pIn_);
    FanPS2.setInletEffortModel(pIn_);
    FanCM1.setInletEffortModel(pIn_);
    FanCM2.setInletEffortModel(pIn_);
    FanCM3.setInletEffortModel(pIn_);
    FanCM4.setInletEffortModel(pIn_);
    FanCM5.setInletEffortModel(pIn_);
    FanCM6.setInletEffortModel(pIn_);
    FanInM1.setInletEffortModel(pIn_);
    FanInM2.setInletEffortModel(pIn_);
    FanInM3.setInletEffortModel(pIn_);

    FanPS1.setOutletEffortModel(pMidPS1_);
    FanPS2.setOutletEffortModel(pMidPS2_);
    FanCM1.setOutletEffortModel(pMidCM1_);
    FanCM2.setOutletEffortModel(pMidCM2_);
    FanCM3.setOutletEffortModel(pMidCM3_);
    FanCM4.setOutletEffortModel(pMidCM4_);
    FanCM5.setOutletEffortModel(pMidCM5_);
    FanCM6.setOutletEffortModel(pMidCM6_);
    FanInM1.setOutletEffortModel(pMidInM1_);
    FanInM2.setOutletEffortModel(pMidInM2_);
    FanInM3.setOutletEffortModel(pMidInM3_);
    plenumHXerPS1_.setPlenumInletEffortModel(pMidPS1_);
    plenumHXerPS2_.setPlenumInletEffortModel(pMidPS2_);
    plenumHXerCM1_.setPlenumInletEffortModel(pMidCM1_);
    plenumHXerCM2_.setPlenumInletEffortModel(pMidCM2_);
    plenumHXerCM3_.setPlenumInletEffortModel(pMidCM3_);
    plenumHXerCM4_.setPlenumInletEffortModel(pMidCM4_);
    plenumHXerCM5_.setPlenumInletEffortModel(pMidCM5_);
    plenumHXerCM6_.setPlenumInletEffortModel(pMidCM6_);
    plenumHXerInM1_.setPlenumInletEffortModel(pMidInM1_);
    plenumHXerInM2_.setPlenumInletEffortModel(pMidInM2_);
    plenumHXerInM3_.setPlenumInletEffortModel(pMidInM3_);

    plenumHXerPS1_.setPlenumOutletEffortModel(pOut_);
    plenumHXerPS2_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM1_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM2_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM3_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM4_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM5_.setPlenumOutletEffortModel(pOut_);
    plenumHXerCM6_.setPlenumOutletEffortModel(pOut_);
    plenumHXerInM1_.setPlenumOutletEffortModel(pOut_);
    plenumHXerInM2_.setPlenumOutletEffortModel(pOut_);
    plenumHXerInM3_.setPlenumOutletEffortModel(pOut_);

    PS1_Heat.setOutletEffortModel(tempPS1);
    PS2_Heat.setOutletEffortModel(tempPS2);
    CM1_Heat.setOutletEffortModel(tempCM1);
    CM2_Heat.setOutletEffortModel(tempCM2);
    CM3_Heat.setOutletEffortModel(tempCM3);
    CM4_Heat.setOutletEffortModel(tempCM4);
    CM5_Heat.setOutletEffortModel(tempCM5);
    CM6_Heat.setOutletEffortModel(tempCM6);
    InM1_Heat.setOutletEffortModel(tempInM1);
    InM2_Heat.setOutletEffortModel(tempInM2);
    InM3_Heat.setOutletEffortModel(tempInM3);

    plenumHXerPS1_.setConvectionInletEffortModel(tempPS1);
    plenumHXerPS2_.setConvectionInletEffortModel(tempPS2);
    plenumHXerCM1_.setConvectionInletEffortModel(tempCM1);
    plenumHXerCM2_.setConvectionInletEffortModel(tempCM2);
    plenumHXerCM3_.setConvectionInletEffortModel(tempCM3);
    plenumHXerCM4_.setConvectionInletEffortModel(tempCM4);
    plenumHXerCM5_.setConvectionInletEffortModel(tempCM5);
    plenumHXerCM6_.setConvectionInletEffortModel(tempCM6);
    plenumHXerInM1_.setConvectionInletEffortModel(tempInM1);
    plenumHXerInM2_.setConvectionInletEffortModel(tempInM2);
    plenumHXerInM3_.setConvectionInletEffortModel(tempInM3);

    workingFluid_.updatePropsPH(pIn_.getEffort(), pIn_.get(ENTHALPY))
;

    //Set working fluid
    pIn_.setFluid(workingFluid_);
    FanPS1.setFluid(workingFluid_);
    FanPS2.setFluid(workingFluid_);
    FanCM1.setFluid(workingFluid_);
    FanCM2.setFluid(workingFluid_);
    FanCM3.setFluid(workingFluid_);
    FanCM4.setFluid(workingFluid_);
    FanCM5.setFluid(workingFluid_);
    FanCM6.setFluid(workingFluid_);
    FanInM1.setFluid(workingFluid_);
    FanInM2.setFluid(workingFluid_);
    FanInM3.setFluid(workingFluid_);
    pMidPS1_.setFluid(workingFluid_);
    pMidPS2_.setFluid(workingFluid_);
    pMidCM1_.setFluid(workingFluid_);
    pMidCM2_.setFluid(workingFluid_);
    pMidCM3_.setFluid(workingFluid_);
    pMidCM4_.setFluid(workingFluid_);
    pMidCM5_.setFluid(workingFluid_);
    pMidCM6_.setFluid(workingFluid_);
    pMidInM1_.setFluid(workingFluid_);
    pMidInM2_.setFluid(workingFluid_);
    pMidInM3_.setFluid(workingFluid_);
    plenumHXerPS1_.setFluid(workingFluid_);
    plenumHXerPS2_.setFluid(workingFluid_);
    plenumHXerCM1_.setFluid(workingFluid_);
    plenumHXerCM2_.setFluid(workingFluid_);
    plenumHXerCM3_.setFluid(workingFluid_);
    plenumHXerCM4_.setFluid(workingFluid_);
    plenumHXerCM5_.setFluid(workingFluid_);
    plenumHXerCM6_.setFluid(workingFluid_);
    plenumHXerInM1_.setFluid(workingFluid_);
    plenumHXerInM2_.setFluid(workingFluid_);
    plenumHXerInM3_.setFluid(workingFluid_);
    pOut_.setFluid(workingFluid_);

    //***ELECTRICAL MODELS***//

    //Boundary Nodes
    //---------------------------------------------------------------------
--------------
    //Inlet Voltage AC
    ACElectricalEffortModel portSource("_PSource");
    portSource.setAmplitude(792);
    portSource.setDependent(false);

    ACElectricalEffortModel starboardSource("_SBSource");
    starboardSource.setAmplitude(679);
    starboardSource.setDependent(false);

    //Outlet Ground
    ElectricalEffortModel ground("_Ground");
    ground.setVoltage(0);
    ground.setDependent(false);

    //Cables
```

```
//-----------------------------------------------------------------
Resistor portBus("_PortBus");
portBus.setResistance(0.0002);
Resistor starboardBus("_StarboardBus");
starboardBus.setResistance(0.0002);
Resistor cable1("_Cable1");
cable1.setResistance(0.0002);
Resistor cable2("_Cable2");
cable2.setResistance(0.0002);
Resistor cable3("_Cable3");
cable3.setResistance(0.0002);
Resistor cable4("_Cable4");
cable4.setResistance(0.0002);
Resistor cable5("_Cable5");
cable5.setResistance(0.0002);
Resistor cable6("_Cable6");
cable6.setResistance(0.0002);
Resistor cable7("_Cable7");
cable7.setResistance(0.0002);
Resistor cable8("_Cable8");
cable8.setResistance(0.0002);
Resistor cable9("_Cable9");
cable9.setResistance(0.0002);
Resistor cable10("_Cable10");
cable10.setResistance(0.0002);
Resistor cable11("_Cable11");
cable11.setResistance(0.0002);
Resistor cable12("_Cable12");
cable12.setResistance(0.0002);

//Converters
//-----------------------------------------------------------------
//Port PS1 and Starboard PS2
PCM4 PS1("_PS1");
PS1.setPowerFactor(0.96);
PS1.setTurnsRatio(0.632);
PS1.setInductance(0.00001);
PS1.setSnubberCapacitance(0.00006);
PS1.setSnubberResistance(50);
PS1.setInitialInductorFluxLinkage(0.00018);
PS1.setInitialCapacitorCharge(0.03);
PS1.setInitialLossFreeResistance(10);
PS1.setUpdateFrequency(100);
PS1.setInternalHeatGenerationModel(PS1_Heat);
PCM4 PS2("_PS2");
PS2.setPowerFactor(0.96);
PS2.setTurnsRatio(0.738);
PS2.setInductance(0.00001);
PS2.setSnubberCapacitance(0.00006);
PS2.setSnubberResistance(50);
PS2.setInitialInductorFluxLinkage(0.00018);
PS2.setInitialCapacitorCharge(0.03);
PS2.setInitialLossFreeResistance(10);
PS2.setUpdateFrequency(100);
```

```
PS2.setInternalHeatGenerationModel(PS2_Heat);

//Port CM
PCM1 SSCM_1("_SSCM_1");
SSCM_1.setPowerFactor(0.96);
SSCM_1.setInductance(0.05);
SSCM_1.setSnubberCapacitance(0.00000005);
SSCM_1.setSnubberResistance(5000);
SSCM_1.setInitialInductorFluxLinkage(0.14);
SSCM_1.setDutyCycle(0.84);
SSCM_1.setInitialCapacitorCharge(0.000021);
SSCM_1.setUpdateFrequency(75);
SSCM_1.setInternalHeatGenerationModel(CM1_Heat);
PCM1 SSCM_2("_SSCM_2");
SSCM_2.setPowerFactor(0.96);
SSCM_2.setInductance(0.05);
SSCM_2.setSnubberCapacitance(0.00000005);
SSCM_2.setSnubberResistance(5000);
SSCM_2.setInitialInductorFluxLinkage(0.14);
SSCM_2.setDutyCycle(0.84);
SSCM_2.setInitialCapacitorCharge(0.000021);
SSCM_2.setUpdateFrequency(75);
SSCM_2.setInternalHeatGenerationModel(CM2_Heat);
PCM1 SSCM_3("_SSCM_3");
SSCM_3.setPowerFactor(0.96);
SSCM_3.setInductance(0.05);
SSCM_3.setSnubberCapacitance(0.00000005);
SSCM_3.setSnubberResistance(5000);
SSCM_3.setInitialInductorFluxLinkage(0.14);
SSCM_3.setDutyCycle(0.84);
SSCM_3.setInitialCapacitorCharge(0.000021);
SSCM_3.setUpdateFrequency(75);
SSCM_3.setInternalHeatGenerationModel(CM3_Heat);

//Starboard CM
PCM1 SSCM_4("_SSCM_4");
SSCM_4.setPowerFactor(0.96);
SSCM_4.setInductance(0.05);
SSCM_4.setSnubberCapacitance(0.00000005);
SSCM_4.setSnubberResistance(5000);
SSCM_4.setInitialInductorFluxLinkage(0.14);
SSCM_4.setDutyCycle(0.84);
SSCM_4.setInitialCapacitorCharge(0.000021);
SSCM_4.setUpdateFrequency(75);
SSCM_4.setInternalHeatGenerationModel(CM4_Heat);
PCM1 SSCM_5("_SSCM_5");
SSCM_5.setPowerFactor(0.96);
SSCM_5.setInductance(0.05);
SSCM_5.setSnubberCapacitance(0.00000005);
SSCM_5.setSnubberResistance(5000);
SSCM_5.setInitialInductorFluxLinkage(0.14);
SSCM_5.setDutyCycle(0.84);
SSCM_5.setInitialCapacitorCharge(0.000021);
SSCM_5.setUpdateFrequency(75);
```

```
SSCM_5.setInternalHeatGenerationModel(CM5_Heat);
PCM1 SSCM_6("_SSCM_6");   SSCM_6.setPowerFactor(0.96);
SSCM_6.setPowerFactor(0.96);
SSCM_6.setInductance(0.05);
SSCM_6.setSnubberCapacitance(0.00000005);
SSCM_6.setSnubberResistance(5000);
SSCM_6.setInitialInductorFluxLinkage(0.14);
SSCM_6.setDutyCycle(0.84);
SSCM_6.setInitialCapacitorCharge(0.000021);
SSCM_6.setUpdateFrequency(75);
SSCM_6.setInternalHeatGenerationModel(CM6_Heat);

//InM
PCM2 SSIM_1("_SSIM_1");
SSIM_1.setPowerFactor(0.96);
SSIM_1.setInductance(0.2);
SSIM_1.setSnubberCapacitance(0.0000005);
SSIM_1.setSnubberResistance(500);
SSIM_1.setInitialInductorFluxLinkage(0.9);
SSIM_1.setInitialLossFreeResistance(157);
SSIM_1.setInitialCapacitorCharge(0.00012);
SSIM_1.setUpdateFrequency(50);
SSIM_1.setInternalHeatGenerationModel(InM1_Heat);
PCM2 SSIM_2("_SSIM_2");
SSIM_2.setPowerFactor(0.96);
SSIM_2.setInductance(0.2);
SSIM_2.setSnubberCapacitance(0.0000005);
SSIM_2.setSnubberResistance(500);
SSIM_2.setInitialInductorFluxLinkage(0.9);
SSIM_2.setInitialLossFreeResistance(157);
SSIM_2.setInitialCapacitorCharge(0.00012);
SSIM_2.setUpdateFrequency(50);
SSIM_2.setInternalHeatGenerationModel(InM2_Heat);
PCM2 SSIM_3("_SSIM_3");
SSIM_3.setPowerFactor(0.96);
SSIM_3.setInductance(0.2);
SSIM_3.setSnubberCapacitance(0.0000005);
SSIM_3.setSnubberResistance(500);
SSIM_3.setInitialInductorFluxLinkage(0.9);
SSIM_3.setInitialLossFreeResistance(157);
SSIM_3.setInitialCapacitorCharge(0.00012);
SSIM_3.setUpdateFrequency(50);
SSIM_3.setInternalHeatGenerationModel(InM3_Heat);

//Loads
//-------------------------------------------------------------
Resistor LB1("_LB1");
LB1.setResistance(56);
LB1.setEvent(6, 23);
LB1.setEvent(7, 11);
LB1.setEvent(10, 0.41);
Resistor LB2("_LB1");
LB2.setResistance(56);
LB2.setEvent(7, 23);
```

```
    LB2.setEvent(8, 11);
    LB2.setEvent(10, 0.41);
    Resistor LB3("_LB3");
    LB3.setResistance(56);
    LB3.setEvent(8, 23);
    LB3.setEvent(9, 11);
    LB3.setEvent(10, 0.41);
    Resistor load1("_Load1");
    load1.setResistance(100);
    Resistor load2("_Load2");
    load2.setResistance(100);
    Resistor load3("_Load3");
    load3.setResistance(100);

    //Electrical Connections
    //-----------------------------------------------------------------
    //PCM-1 to PCM-2
    portSource.addOutletFlowModel(portBus);
    starboardSource.addOutletFlowModel(starboardBus);
    PS1.addInletFlowModel(portBus);
    PS2.addInletFlowModel(starboardBus);
    PS1.addOutletFlowModel(cable1);
    PS1.addOutletFlowModel(cable2);
    PS1.addOutletFlowModel(cable3);
    PS2.addOutletFlowModel(cable4);
    PS2.addOutletFlowModel(cable5);
    PS2.addOutletFlowModel(cable6);
    SSCM_1.addInletFlowModel(cable1);
    SSCM_2.addInletFlowModel(cable2);
    SSCM_3.addInletFlowModel(cable3);
    SSCM_4.addInletFlowModel(cable4);
    SSCM_5.addInletFlowModel(cable5);
    SSCM_6.addInletFlowModel(cable6);
    SSCM_1.addOutletFlowModel(cable7);
    SSCM_1.addOutletFlowModel(load1);
    SSCM_2.addOutletFlowModel(cable8);
    SSCM_2.addOutletFlowModel(load3);
    SSCM_3.addOutletFlowModel(cable9);
    SSCM_3.addOutletFlowModel(load2);
    SSCM_4.addOutletFlowModel(cable10);
    SSCM_5.addOutletFlowModel(cable11);
    SSCM_6.addOutletFlowModel(cable12);
    SSIM_1.addInletFlowModel(cable8);
    SSIM_1.addInletFlowModel(cable11);
    SSIM_2.addInletFlowModel(cable9);
    SSIM_2.addInletFlowModel(cable12);
    SSIM_3.addInletFlowModel(cable7);
    SSIM_3.addInletFlowModel(cable10);

    //PCM-1 and PCM-2 to loads
    SSIM_1.addOutletFlowModel(LB1);
    SSIM_2.addOutletFlowModel(LB2);
    SSIM_3.addOutletFlowModel(LB3);
    //Grounding
    PS1.setOutletEffortModel(ground);
    PS2.setOutletEffortModel(ground);
    SSCM_1.setOutletEffortModel(ground);
    SSCM_2.setOutletEffortModel(ground);
    SSCM_3.setOutletEffortModel(ground);
    SSCM_4.setOutletEffortModel(ground);
    SSCM_5.setOutletEffortModel(ground);
    SSCM_6.setOutletEffortModel(ground);
    SSIM_1.setOutletEffortModel(ground);
    SSIM_2.setOutletEffortModel(ground);
    SSIM_3.setOutletEffortModel(ground);
    LB1.setOutletEffortModel(ground);
    LB2.setOutletEffortModel(ground);
    LB3.setOutletEffortModel(ground);
    load1.setOutletEffortModel(ground);
    load2.setOutletEffortModel(ground);
    load3.setOutletEffortModel(ground);

    //Simulation Set-up
    //--------------------------------------------------------------
    DTMSSimulation SimExec("MultipleZone.csv", 3600, 0.0005, 1) ;

    // Create solver object
    GloballyConvergentResistiveSolver* simulationSolver = new
GloballyConvergentResistiveSolver() ;

    // Set solver error tolerance
    simulationSolver->setErrorTolerance(1e-4) ;

    // Add solver to simulation executive
    SimExec.addSolver(simulationSolver);

    // Add models into solver
    //Electrical
    simulationSolver->addModel(portSource);
    simulationSolver->addModel(starboardSource);
    simulationSolver->addModel(portBus);
    simulationSolver->addModel(starboardBus);
    simulationSolver->addModel(PS1);
    simulationSolver->addModel(PS2);
    simulationSolver->addModel(cable1);
    simulationSolver->addModel(cable2);
    simulationSolver->addModel(cable3);
    simulationSolver->addModel(cable4);
    simulationSolver->addModel(cable5);
    simulationSolver->addModel(cable6);
    simulationSolver->addModel(SSCM_1);
    simulationSolver->addModel(SSCM_2);
    simulationSolver->addModel(SSCM_3);
    simulationSolver->addModel(SSCM_4);
    simulationSolver->addModel(SSCM_5);
    simulationSolver->addModel(SSCM_6);
    simulationSolver->addModel(cable7);
    simulationSolver->addModel(cable8);
    simulationSolver->addModel(cable9);
    simulationSolver->addModel(cable10);
    simulationSolver->addModel(cable11);
    simulationSolver->addModel(cable12);
    simulationSolver->addModel(SSIM_1);
    simulationSolver->addModel(SSIM_2);
    simulationSolver->addModel(SSIM_3);
    simulationSolver->addModel(LB1);
    simulationSolver->addModel(LB2);
    simulationSolver->addModel(LB3);
    simulationSolver->addModel(load1);
    simulationSolver->addModel(load2);
    simulationSolver->addModel(load3);
    simulationSolver->addModel(ground);
    //Heat and Fluid
    simulationSolver->addModel(pIn_);
    simulationSolver->addModel(FanPS1);
    simulationSolver->addModel(FanPS2);
    simulationSolver->addModel(FanCM1);
    simulationSolver->addModel(FanCM2);
    simulationSolver->addModel(FanCM3);
    simulationSolver->addModel(FanCM4);
    simulationSolver->addModel(FanCM5);
    simulationSolver->addModel(FanCM6);
    simulationSolver->addModel(FanInM1);
    simulationSolver->addModel(FanInM2);
    simulationSolver->addModel(FanInM3);
    simulationSolver->addModel(pMidPS1_);
    simulationSolver->addModel(pMidPS2_);
    simulationSolver->addModel(pMidCM1_);
    simulationSolver->addModel(pMidCM2_);
    simulationSolver->addModel(pMidCM3_);
    simulationSolver->addModel(pMidCM4_);
    simulationSolver->addModel(pMidCM5_);
    simulationSolver->addModel(pMidCM6_);
    simulationSolver->addModel(pMidInM1_);
    simulationSolver->addModel(pMidInM2_);
    simulationSolver->addModel(pMidInM3_);
    simulationSolver->addModel(plenumHXerPS1_);
    simulationSolver->addModel(plenumHXerPS2_);
    simulationSolver->addModel(plenumHXerCM1_);
    simulationSolver->addModel(plenumHXerCM2_);
    simulationSolver->addModel(plenumHXerCM3_);
    simulationSolver->addModel(plenumHXerCM4_);
    simulationSolver->addModel(plenumHXerCM5_);
    simulationSolver->addModel(plenumHXerCM6_);
    simulationSolver->addModel(plenumHXerInM1_);
    simulationSolver->addModel(plenumHXerInM2_);
    simulationSolver->addModel(plenumHXerInM3_);
    simulationSolver->addModel(pOut_);
    simulationSolver->addModel(tempPS1);
    simulationSolver->addModel(tempPS2);
    simulationSolver->addModel(tempCM1);
    simulationSolver->addModel(tempCM2);
    simulationSolver->addModel(tempCM3);
    simulationSolver->addModel(tempCM4);
    simulationSolver->addModel(tempCM5);
    simulationSolver->addModel(tempCM6);
    simulationSolver->addModel(tempInM1);
    simulationSolver->addModel(tempInM2);
    simulationSolver->addModel(tempInM3);
    simulationSolver->addModel(PS1_Heat);
    simulationSolver->addModel(PS2_Heat);
    simulationSolver->addModel(CM1_Heat);
    simulationSolver->addModel(CM2_Heat);
    simulationSolver->addModel(CM3_Heat);
    simulationSolver->addModel(CM4_Heat);
    simulationSolver->addModel(CM5_Heat);
    simulationSolver->addModel(CM6_Heat);
    simulationSolver->addModel(InM1_Heat);
    simulationSolver->addModel(InM2_Heat);
    simulationSolver->addModel(InM3_Heat);

    // Add models to simulation executive
    //Electrical
    SimExec.addModel(starboardSource);
    SimExec.addModel(portSource);
    SimExec.addModel(starboardBus);
    SimExec.addModel(portBus);
    SimExec.addModel(PS1);
    SimExec.addModel(PS2);
    SimExec.addModel(cable1);
    SimExec.addModel(cable2);
    SimExec.addModel(cable3);
    SimExec.addModel(cable4);
    SimExec.addModel(cable5);
    SimExec.addModel(cable6);
    SimExec.addModel(SSCM_1);
    SimExec.addModel(SSCM_2);
    SimExec.addModel(SSCM_3);
    SimExec.addModel(SSCM_4);
    SimExec.addModel(SSCM_5);
    SimExec.addModel(SSCM_6);
    SimExec.addModel(cable7);
    SimExec.addModel(cable8);
    SimExec.addModel(cable9);
    SimExec.addModel(cable10);
    SimExec.addModel(cable11);
    SimExec.addModel(cable12);
    SimExec.addModel(SSIM_1);
    SimExec.addModel(SSIM_2);
    SimExec.addModel(SSIM_3);
    SimExec.addModel(LB1);
    SimExec.addModel(LB2);
    SimExec.addModel(LB3);
    SimExec.addModel(load1);
    SimExec.addModel(load2);
    SimExec.addModel(load3);
    SimExec.addModel(ground);
    //Heat and Fluid
    SimExec.addModel(pIn_);
    SimExec.addModel(FanPS1);
    SimExec.addModel(FanPS2);
    SimExec.addModel(FanCM1);
    SimExec.addModel(FanCM2);
    SimExec.addModel(FanCM3);
    SimExec.addModel(FanCM4);
    SimExec.addModel(FanCM5);
    SimExec.addModel(FanCM6);
    SimExec.addModel(FanInM1);
    SimExec.addModel(FanInM2);
    SimExec.addModel(FanInM3);
    SimExec.addModel(pMidPS1_);
    SimExec.addModel(pMidPS2_);
    SimExec.addModel(pMidCM1_);
    SimExec.addModel(pMidCM2_);
    SimExec.addModel(pMidCM3_);
    SimExec.addModel(pMidCM4_);
    SimExec.addModel(pMidCM5_);
    SimExec.addModel(pMidCM6_);
    SimExec.addModel(pMidInM1_);
    SimExec.addModel(pMidInM2_);
    SimExec.addModel(pMidInM3_);
    SimExec.addModel(plenumHXerPS1_);
    SimExec.addModel(plenumHXerPS2_);
    SimExec.addModel(plenumHXerCM1_);
    SimExec.addModel(plenumHXerCM2_);
    SimExec.addModel(plenumHXerCM3_);
    SimExec.addModel(plenumHXerCM4_);
    SimExec.addModel(plenumHXerCM5_);
    SimExec.addModel(pMidCM6_);
    SimExec.addModel(pMidInM1_);
    SimExec.addModel(pMidInM2_);
    SimExec.addModel(pMidInM3_);
    SimExec.addModel(plenumHXerPS1_);
    SimExec.addModel(plenumHXerPS2_);
    SimExec.addModel(plenumHXerCM1_);
    SimExec.addModel(plenumHXerCM2_);
    SimExec.addModel(plenumHXerCM3_);
    SimExec.addModel(plenumHXerCM4_);
    SimExec.addModel(plenumHXerCM5_);
    SimExec.addModel(plenumHXerCM6_);
    SimExec.addModel(plenumHXerInM1_);
    SimExec.addModel(plenumHXerInM2_);
    SimExec.addModel(plenumHXerInM3_);
    SimExec.addModel(pOut_);
    SimExec.addModel(tempPS1);
    SimExec.addModel(tempPS2);
    SimExec.addModel(tempCM1);
    SimExec.addModel(tempCM2);
    SimExec.addModel(tempCM3);
    SimExec.addModel(tempCM4);
    SimExec.addModel(tempCM5);
    SimExec.addModel(tempCM6);
    SimExec.addModel(tempInM1);
    SimExec.addModel(tempInM2);
    SimExec.addModel(tempInM3);
    SimExec.addModel(PS1_Heat);
    SimExec.addModel(PS2_Heat);
    SimExec.addModel(CM1_Heat);
    SimExec.addModel(CM2_Heat);
    SimExec.addModel(CM3_Heat);
    SimExec.addModel(CM4_Heat);
    SimExec.addModel(CM5_Heat);
    SimExec.addModel(CM6_Heat);
    SimExec.addModel(InM1_Heat);
    SimExec.addModel(InM2_Heat);
    SimExec.addModel(InM3_Heat);

    //Set Write Flags
    //Electrical
    portSource.setWriteFlag(1);
    starboardSource.setWriteFlag(1);
    portBus.setWriteFlag(1);
    starboardBus.setWriteFlag(1);
    PS1.setWriteFlag(1);
    PS2.setWriteFlag(1);
    cable1.setWriteFlag(0);
    cable2.setWriteFlag(0);
    cable3.setWriteFlag(0);
    cable4.setWriteFlag(0);
    cable5.setWriteFlag(0);
    cable6.setWriteFlag(0);
    cable7.setWriteFlag(0);
    cable8.setWriteFlag(0);
    cable9.setWriteFlag(0);
    cable10.setWriteFlag(0);
    cable11.setWriteFlag(0);
    cable12.setWriteFlag(0);
    SSCM_1.setWriteFlag(1);
    SSCM_2.setWriteFlag(1);
    SSCM_3.setWriteFlag(1);
    SSCM_4.setWriteFlag(1);
    SSCM_5.setWriteFlag(1);
    SSCM_6.setWriteFlag(1);
    SSIM_1.setWriteFlag(1);
    SSIM_2.setWriteFlag(1);
    SSIM_3.setWriteFlag(1);
    LB1.setWriteFlag(1);
    LB2.setWriteFlag(1);
    LB3.setWriteFlag(1);
    ground.setWriteFlag(1);
    //Heat and Fluid
    pIn_.setWriteFlag(1);
    FanPS1.setWriteFlag(1);
    FanPS2.setWriteFlag(1);
    FanCM1.setWriteFlag(1);
    FanCM2.setWriteFlag(1);
    FanCM3.setWriteFlag(1);
    FanCM4.setWriteFlag(1);
    FanCM5.setWriteFlag(1);
    FanCM6.setWriteFlag(1);
    FanInM1.setWriteFlag(1);
    FanInM2.setWriteFlag(1);
    FanInM3.setWriteFlag(1);
    pMidPS1_.setWriteFlag(1);
    pMidPS2_.setWriteFlag(1);
    pMidCM1_.setWriteFlag(1);
    pMidCM2_.setWriteFlag(1);
    pMidCM3_.setWriteFlag(1);
    pMidCM4_.setWriteFlag(1);
    pMidCM5_.setWriteFlag(1);
    pMidCM6_.setWriteFlag(1);
    pMidInM1_.setWriteFlag(1);
    pMidInM2_.setWriteFlag(1);
    pMidInM3_.setWriteFlag(1);
    plenumHXerPS1_.setWriteFlag(1);
    plenumHXerPS2_.setWriteFlag(1);
    plenumHXerCM1_.setWriteFlag(1);
    plenumHXerCM2_.setWriteFlag(1);
    plenumHXerCM3_.setWriteFlag(1);
    plenumHXerCM4_.setWriteFlag(1);
    plenumHXerCM5_.setWriteFlag(1);
    plenumHXerCM6_.setWriteFlag(1);
    plenumHXerInM1_.setWriteFlag(1);
    plenumHXerInM2_.setWriteFlag(1);
    plenumHXerInM3_.setWriteFlag(1);
    pOut_.setWriteFlag(1);
    PS1_Heat.setWriteFlag(1);
    PS2_Heat.setWriteFlag(1);
    CM1_Heat.setWriteFlag(1);
    CM2_Heat.setWriteFlag(1);
    CM3_Heat.setWriteFlag(1);
    CM4_Heat.setWriteFlag(1);
    CM5_Heat.setWriteFlag(1);
    CM6_Heat.setWriteFlag(1);
    InM1_Heat.setWriteFlag(1);
    InM2_Heat.setWriteFlag(1);
    InM3_Heat.setWriteFlag(1);
    tempPS1.setWriteFlag(1);
    tempPS2.setWriteFlag(1);
    tempCM1.setWriteFlag(1);
    tempCM2.setWriteFlag(1);
    tempCM3.setWriteFlag(1);
    tempCM4.setWriteFlag(1);
    tempCM5.setWriteFlag(1);
    tempCM6.setWriteFlag(1);
    tempInM1.setWriteFlag(1);
    tempInM2.setWriteFlag(1);
    tempInM3.setWriteFlag(1);


    //Begin Simulation
    cout << "simulating... " << endl ;

    //Use a clock to measure elapsed time
    clock_t cBegin, cEnd ;
    double cSim ;
    cBegin = clock() ;

    SimExec.runSimulation() ;

    cEnd = clock() ;
    cSim = (cEnd-cBegin) ;
    cSim = cSim/CLOCKS_PER_SEC ;

    cout << endl << "done" << endl << endl;

    cout << "Real Time (elapsed): " << cSim << endl ;
    cout << endl;

    //Clean up allocated memory
    //N/A
}
```

References

[1] American Bureau of Shipping., “Control of Harmonics in Electrical Power Systems”, May 2006.

[2] Burton, Ludovic. “Multi-scale Thermal Modeling Methodology for High Power-Electronic Cabinets”, Master’s Thesis, Georgia Institute of Technology, December 2007.

[3] Carroll, B.C., “Improved Thermal Management of an All-Electric Ship through Modeling and Simulation,” Master’s Thesis, The University of Texas at Austin, 2004.

[4] Cartwright, Kenneth V., "Determining the Effective or RMS Voltage of Various Waveforms without Calculus", Technology Interface, 2007.

[5] Chan, R.R., Y. Lee, S.D. Sudhoff, and E.L. Zivi, “Evolutionary optimization of power electronics based power system,” *IEEE Trans. Power Electron.*, vol. 23, no. 4, pp. 1907 - 1917, Jul. 2008.

[6] CoilCraft. “Inductor Performance in High Frequency DC-DC Converters: Understanding AC Losses”, Revised: February 18, 2009.

[7] Erickson, Robert W., Dragan Maksimovic. *Fundamentals of Power Electronics*. Second Ed. Spring Science, 2001.

[8] Fang, Ruixian, et. al. “System-Level Dynamic Thermal Modeling and Simulation for an All-Electric Ship Cooling System in VTB,” IEEE ESTS 2007 Conference, Accepted January 2007.

[9] Haag, S.T., “Steady-State and Dynamic Simulation of Large Thermal Systems.” Master’s Thesis, The University of Texas at Austin, December 2005.

[10] Hewlett, P.T., “Implementation of an In-house Framework for Dynamic Assessment of Thermal Load Management Strategies Aboard Navy Surface Ships,” Master’s Thesis, The University of Texas at Austin, December 2008.

[11] Holsonback, C.R., “Dynamic Thermal-Mechanical-Electrical Modeling of the Integrated Power System of a Notional All-Electric Naval Surface Ship,” Master’s Thesis, The University of Texas at Austin, May 2007.

[12] Incropera Frank R. and David P. DeWitt, *Fundamentals of Heat and Mass Transfer,* 5th ed., (New York: John Wiley & Sons, 2002).

[13] “Next Generation Integrated Power System: NGIPS Technology Development Roadmap”, White Paper, November 30, 2007.

[14] Paullus, P., “Creation of a Modeling and Simulation Environment for Thermal

Management of an All-Electric Ship". Master's Thesis, The University of Texas at Austin, limited availability, December 2007.

[15] Pierce, Michael, "User's Guide/Tutorial for the Dynamic Thermal Modeling and Simulation (DTMS) Framework." Version 2.1. The University of Texas at Austin. February, 2009.

[16] *RTDS Notional E-ship Model Technical Guide,* Center for Advanced Power Systems, Florida State University, Tallahassee, Florida,Version 3.0. November 2006.

[17] Rucker, Jonathan E. "Design and Analysis of a Permanent Magnet Generator for Naval Applications," Master's Thesis, Massachusetts Institute of Technology, June 2005.

[18] SatCon Applied Technology. "Integrated Fight Through Power" Systems (PCM) for Advanced Electric Ships."

[19] Serway, Raymond A. *Principles of Physics.* Second Ed. London: Saunders College Pub. 1998.

[20] Simpson, T.W., J.D. Peplinski, P.N. Koch, J.K. Allen. "Metamodels for Computer-based Engineering Design: Survey and recommendations." Georgia Institute of Technology, 2001.

[21] Smith, Andrew N., Benjamin T. McGlasson, Jack S. Bernardes. "Heat Generation During the Firing of a Capacitor-Based Railgun System", *IEEE Transactions on Magnetics,* Vol. 43, No. 1, Jan. 2007.

[22] Stevenson, William D., Jr. *Elements of Power Systems Analysis*, 3rd ed., New York: McGraw Hill, 1975.

[23] Ulaby, Fawwaz T. *Fundamentals of Applied Electromagnetics*. Upper Saddle River, New Jersey: Prentice-Hall. 1999.

[24] Webb, T.W., "Thermal Management of Pulsed Loads on an All-Electric Ship," Master's Thesis, The University of Texas at Austin, August 2006.

[25] Weisstein, Eric W. "Backward Difference." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/BackwardDifference.html

[26] Weisstein, Eric W. "Newton-Cotes Formulas." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Newton-CotesFormulas.html

[27] Zerby, Mark, et al., "Thermal Management Concepts for Configurable Zonal Systems Power Conversion Modules", White Paper.

[28] Zerby, Mark. “Summary of Thermal Management Technologies for the Configurable Zonal Ship,” White Paper, October 2000.

[29] Zheng, Ning, Richard A. Wirtz. “Cylindrical Pin-Fin Fan-Sink Heat Transfer and Pressure Drop Correlations,” *IEEE Trans.*vol. 25, no. 1, Mar. 2001.

[30] Zivi, Edwin. “Naval Combat Survivability Testbed MATLAB/Simulink Simulations”, White Paper, May 2009.

Vita

Matthew Andrew Pruske was born July 6, 1983, in San Antonio, Texas, to James and Merrilyn Pruske. He is a proud product of the Boerne Independent School District northwest of San Antonio. After graduating from Boerne High School in 2001, he attended The University of Texas at Austin until graduating in the spring of 2005. After graduation, Matthew worked with Professional Services Industries, as a Graduate Engineer and consultant in the geotechnical engineering industry. Following a hiatus from engineering, traveling in South America, and working as a preschool teacher, Matthew enrolled at the University of Texas at Austin Mechanical Engineering Graduate School with a focus in thermal fluid systems. He worked one year as a teaching assistant in a heat transfer laboratory before working on naval thermal management research under Dr. Tom Kiehne, graduating in August 2009.

Contact Information: M.A. Pruske
8040 Flagstone Hill Drive
Fair Oaks Ranch, TX 78015

Mobile: (830) 446-2453
pruske@gmail.com

This thesis was typed by the author.