

6-30-2016

An Improved Ship Design Tool for Comparing Performance of Multiple Ship Designs across User-Defined Missions

Helder Jose de Almeida Pais
University of South Carolina

Follow this and additional works at: <https://scholarcommons.sc.edu/etd>



Part of the [Electrical and Computer Engineering Commons](#)

Recommended Citation

de Almeida Pais, H. J.(2016). *An Improved Ship Design Tool for Comparing Performance of Multiple Ship Designs across User-Defined Missions*. (Master's thesis). Retrieved from <https://scholarcommons.sc.edu/etd/3371>

This Open Access Thesis is brought to you by Scholar Commons. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholar Commons. For more information, please contact digres@mailbox.sc.edu.

An Improved Ship Design Tool for Comparing Performance of Multiple Ship
Designs across User-Defined Missions

by

Helder Jose de Almeida Pais

Bachelor of Science
University of South Carolina, 2011

Submitted in Partial Fulfillment of the Requirements

For the Degree of Master of Science in

Electrical Engineering

College of Engineering and Computing

University of South Carolina

2016

Accepted by:

Roger Dougal, Director of Thesis

Andrea Benigni, Reader

Lacy Ford, Senior Vice Provost and Dean of Graduate Studies

© Copyright by Helder Pais, 2016
All Rights Reserved.

ACKNOWLEDGEMENTS

The author would like to thank Dr. Roger Dougal for his efforts in guiding this research and Dr. Andrea Benigni for assisting with the thesis review.

The author would like to thank Blake Langland, Rod Leonard and Ernie Broughton for contributing to and supporting this research.

The author would like to thank Lillian Wanda for the support and guidance provided during the process of writing this thesis.

ABSTRACT

In the early stages of ship design, engineers from many different disciplines need to have confidence that a design being produced converges across the electrical, mechanical, and thermal domains, and that the design meets the needs of the stakeholders, and, that a particular design performs better or more optimally than other designs produced to accomplish a mission. The tools currently available do not permit the engineering teams to gain this insight in a reasonable amount of time. In this thesis, we discuss the improvements that we have made to our existing ship design tools in the S3D environment, allowing for a more concurrent collaboration between the engineers from all disciplines in the ship design process.

Incorporating the notion of time into our existing steady state solvers, we developed a controller class responsible for keeping track of time-related information and scheduling time-based events using the earliest deadline first algorithm. We have also incorporated Python script instructions in the form of an attribute inserted into the equipment models in order to allow Python scripts to represent the effects of system controls. Equipment models were also modified to provide information regarding time dependent metrics, such as fuel and energy, and in order to account for interdependencies between disciplines, they were also given the capacity to inform the mission analyzer tool about whether their dependencies have been satisfied. We

developed an algorithm that uses this information to efficiently find a solution such that all dependencies between disciplines are satisfied. Implementing this algorithm, the mission analyzer is able to simulate each of the disciplines for a specified time frame and provide results that indicate whether a ship design is able to complete a mission and the possible costs such as equipment failures, and fuel consumed.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
LIST OF FIGURES	viii
LIST OF SYMBOLS	x
LIST OF ABBREVIATIONS	xii
GLOSSARY	xiii
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 SOLVER DEVELOPMENT	4
2.1 SOLVERS – TIME DEPENDENCY	4
2.2 INITIAL STATE OF SOLVERS.....	8
2.3 QUASI-STEADY STATE SOLVERS	10
2.4 ADDITION OF TOOLS FOR DEFINING MODE TRANSITIONS	13
CHAPTER 3 COMPONENT DEVELOPMENT	16
3.1 DISCRETE STATE AND TIME DEPENDENT MODIFICATIONS.....	16
3.2 COMMUNICATION ACROSS DISCIPLINES.....	23
CHAPTER 4 MISSION ANALYSER.....	25
4.1 CONVERGENCE ALGORITHM	26
4.2 MISSION ANALYZER WALK-THROUGH.....	28
4.3 MISSION EXPLORER WALK-THROUGH	31

4.4 MISSION EXAMPLE.....	34
CHAPTER 5 CONCLUSION AND FUTURE WORK	40
5.1 CONCLUSION	40
5.2 FUTURE WORK.....	41
REFERENCES	43
APPENDIX A: DIFFERENCES BETWEEN POWER LOAD AND MNA	45
APPENDIX B: MULTI INPUT ATTRIBUTES	48
APPENDIX C: INTER-DISCIPLINE CONVERGENCY GENERALIZED EXAMPLE	49

LIST OF FIGURES

Figure 2.1 Component updates within a time segment	11
Figure 2.2 Stepping event example	12
Figure 2.3 Python script tool.....	14
Figure 2.4 Python code system example	15
Figure 3.1 Electrical percent charging power SOC curve attribute	17
Figure 3.2 Energy storage electrical model flow-chart.....	19
Figure 3.3 Railgun electrical model flow-chart	20
Figure 3.4 Tank model flow-chart.....	22
Figure 3.5 Electrical discipline.....	24
Figure 3.6 machinery discipline	24
Figure 4.1 Resolving solver dependencies.....	27
Figure 4.2 Mission Analyzer principal page	28
Figure 4.3 Alignments setup	29
Figure 4.4 Setting an alignment.....	29
Figure 4.5 Output display window, current configuration	31
Figure 4.6 Changing message warning level.....	31
Figure 4.7 Mission setup.....	32
Figure 4.8 Segment setup	33
Figure 4.9 Full mission simulation output display window	33
Figure 4.10 Naval arc design ship representation	35

Figure 4.11 Propulsion system schematic	35
Figure 4.12 Electrical system schematic	36
Figure 4.13 Thermal fluid system schematic	37
Figure 4.14 Mission results	38
Figure A.1 Difference between Newton-Raphson linear and quadratic convergence.....	45
Figure A.2 Representation of MNA computing into steady state	46
Figure B.3 Spline curve of specific fuel consumption versus mechanical power	48

LIST OF SYMBOLS

C_i	Worst-case time for i^{th} step
E	Efficiency (%)
E_c	Current amount of energy stored (J)
E_{sc}	Energy storage capacity (J)
E_d	Energy discharged during last step (J)
E_p	Amount of stored energy at t_p (J)
k_{lf}	The number of iterations, using Newton-Raphson method
k_{MNA}	Number of iterations per step, using MNA
M_{FRate}	Net mass flow rate (Kg/s)
m_{MNA}	Number of steps required to achieve steady state, using MNA
n	The number of calls to the solver by equipment software models inform the solver about when they need to be called is represented by n .
P_c	Power consumed (W)
P_i	Power injection (W)
t_c	Current time (s)
T_i	Length of time to advance from the $i-1^{\text{th}}$ step to the i^{th} step (s)
$T_{AddFuel}$	Amount of fuel entering or leaving the tank
T_{TFuel}	Total amount of fuel in the tank

t_p	Previous time (s)
T_{Step}	Time step, amount of time to be advanced (s)
U	Utilization bound

LIST OF ABBREVIATIONS

EDFEarliest deadline first, a dynamic scheduling algorithm for placing a process in a priority queue [1].

ESRDC The Electric Ship Research and Development Consortium, a collaborative research group that spans multiple universities including The Florida State University, Massachusetts Institute of Technology, Mississippi State University, Naval Post Graduate School, Purdue University, United States Naval Academy, University of South Carolina, and the University of Texas at Austin, for the purpose of advancing near-term to mid-term research and technology that helps realize electric ships.

MNA Modified nodal analysis is a method by which circuit equations are formulated. This method uses nodal-voltage analysis to determine the voltage equations and Kirchhoff's current law is applied to each node other than the datum node, ground, to determine branch currents [2].

S3D Smart Ship System Design is an early stage concurrent, collaborative design environment which contain multiple discipline specific tools for early-stage ship design.

SOC State of charge, available capacity as a percentage of energy storage capacity. Typically found on energy storage devices such as batteries.

USV Unmanned Surface Vehicles. A vehicle that is capable of operating autonomously without the direct intervention of a human.

GLOSSARY

Attributes . Properties or features that help to define equipment. Example: rated power, efficiency, and rated voltage, can be used to help define the electrical properties of a generator.

Continuous rate of fire The maximum sustainable rate of fire for a weapon. In this context the weapon is an electrical weapon and the limiting factors are typically the availability of electrical energy and the time required to dissipate the heat generated.

Draining One of the three operating states used by the tank model to regulate flow direction when the tank is full. Only flow out of the tank is permitted; the tank inlet is closed, and the tank outlet is open.

Discipline The specialized engineering fields referenced in the paper, electrical, machinery, thermal, and naval architecture.

Equipment Refers to real-world devices for which abstract models will be created within the simulation environment. Example a gas turbine is a real-world device that is represented within the simulation environment with a machinery model and a thermal model.

Event ..Represents an important change in the state of an object at a particular instance of time. [3]

Filling .An operating state of the tank model used to regulate flow direction when a tank is empty. Only flow into the tank is permitted; the tank inlet is open, and the tank outlet is closed.

GensetThe assembly of multiple equipment that is treated as a single piece of equipment, one of which is a gas turbine engine and the other is an electrical generator.

Hybrid ... Default operating state that regulates flow for a tank model. Flow is allowed to happen in both directions; the tank inlet and tank outlet are both open.

Maximum rate of fire The maximum rate and time which a weapon can fire. In this context the weapon is an electrical weapon and the limiting factors are typically the availability of electrical energy and the time required to dissipate the heat generated.

MissionThe object used to define the general objective and total amount of time or distance the ship is expected to operate within.

Mission Segment ..One part of a mission which includes a time interval and an operating state for all equipment that is held essentially constant. A mission is typically made of multiple missions segments.

OnSimulationStart ..A method used by the solver to inform all simulation models that all attributes and variables used by the models should be initialized.

Settling time .. The time required for the response of the system to achieve steady state, within a margin of error.

SignalStepA method used by the solver to call into each simulation model in order to determine advance to a new operating points. During this method, simulation models modify appropriate state variable and update metrics.

Steady StateA stable condition in which the system state does not change over time, $\frac{dx}{dt} = 0$, where variable x denotes the state of the system.

Time step The length of the time segment for which the solver is advancing time. Within a quasi-steady state solver as described in this thesis, it is either the total simulation time, or the time remaining until a state change is required. Within a dynamic system, it is the incremental change in time for solving the governing equations.

CHAPTER 1

INTRODUCTION

The design of ships can be quite complicated as there are multiple disciplines involved and complex interdependencies between these disciplines, therefore, collaboration between engineers from fields such as Electrical, Naval Architecture, and Mechanical is required. For example, stakeholders first establish a list of requirements or identify a set of missions that a ship should be capable of accomplishing. This results in a set of mission payloads being identified and added to the ship design in order to provide the required capabilities. Once these high-level requirements and payloads have been identified, engineers from the various disciplines begin to build the required support systems which provide the necessary power and cooling. It is during this phase that cooperation between engineers is necessary. For instance, the electrical engineers design the electrical system to provide power to the propulsion system and mission loads, however, information needs to flow to the thermal management team such as the heat load produced by each electrical device and the amount of cooling that will be required. Conversely, the thermal management team will need to inform the electrical engineers of the amount of electrical power their pump and other electrical components require.

Until the advent of collaborative design tools, information exchange between design teams had been conducted in a manual, error prone, and inefficient manner. Each project design needed to be interpreted, translated, and moved between tool sets across each discipline [4]. This process would be repeated multiple times in an attempt to finalize the design. Providing the design teams with a concurrent, collaborative design environment enables the information to readily flow between teams, and also allows the users to immediately see and better comprehend the impacts that design decisions have on other disciplines, dramatically reducing the time required to finalize the design.

The Smart Ship System Design, S3D, project was created to allow groups of engineers to collaborate on their designs in real-time [5]. This is a project of the Electric Ship Research and Development Consortium, ESRDC, which is a combination of universities with the objective of advancing near-term to mid-term electric ship concepts [6]. The objective of the S3D design environment is to enable collaboration between engineers working on a single project, and with the help of the provided tools, users are able to create missions in order to simulate the performance of user-created ship designs and to compare those ship designs using the resulting metrics from each of those missions.

In this thesis, we build upon the already existent S3D tools to allow the computation of time dependent metrics, integrate those changes into the simulation models, and create a tool capable of simulating complete missions.

We talk about the challenges and changes necessary in order for time to be taken into account for changes of operating point in CHAPTER 2. A controller class was

created with the responsibility of informing every model about the current time, allowing simulation models to use time dependent metrics such as fuel and energy. This class is also responsible for requesting that the solver run an additional time in the case of a change in operating point. We also added a few tools to allow the user to define under what conditions mode transitions should occur.

In CHAPTER 3, we cover the improvements made to the models to enable them to determine when a change in operating point will occur and inform the solver. We also discuss the modifications required to allow communication between equipment models from different disciplines of the same equipment, such as communication between the electrical model of a generator with the machinery model of the same generator.

In CHAPTER 4, we discuss the creation of a new tool developed to allow users to evaluate performance of a design concept. This tool is able to simulate all of the disciplines' alignments and solve every existing dependency between different models of the same equipment. A walkthrough and an example followed by a discussion on the results will then be used to show how each element comes together.

CHAPTER 2

SOLVER DEVELOPMENT

2.1 SOLVERS – TIME DEPENDENCY

With the addition of time based metrics into the mission objectives, there was a need to provide the current solvers with the ability to take into account time - dependent changes so that important changes in operating point would not be ignored. We will try to determine whether a quasi-steady state solver is preferable to use compared to a time dependent solver.

For our type of simulation, we are interested in a solver that is capable of providing steady-state results as we expect long periods of time without operating point changes. For that reason, methods such as Spice Differentiation [7] and Power System Analysis Toolbox (PSAT) [8], even though they are popular solvers to use with most dynamic power systems, are not suitable for this environment.

A suitable method for our simulation engine is the modified nodal analyses (MNA), a method that requires less information than nodal analysis, does not use constant stepping, and solves the system only up to its steady state value. Since we do not require a solver that is capable of analyzing the system transient response, the power flow solver can be considered.

Both power flow and MNA solvers must solve a system of equations defining a circuit, typically using LU decomposition, an algorithm or solution time that scales at best, $O(n^{2.376})$, using the Coppersmith-Winograd algorithm, where n is the number of equations to be solved [9] [10].

Load-flow models are simplistic models, what we call “ZIP” models (the Z represents constant impedance, the I represents constant current, the P represents constant power). This means they will not typically require internal nodes. However, we solve for both steady-state voltage magnitude and angle, which are separate variables in the matrix, so n is approximately twice the number of nodes. All else being equal, the solution of one operating point using a load-flow solver is therefore 5.191x slower than using MNA, since n is twice as large given our assumptions ($2^{2.376}$).

2.1.1 Similarities between approaches

In both cases, n is typically proportional to the number of nodes. Complex models in MNA will often have internal nodes to represent and solve for internal state variables, but we will assume the level of detail we are interested in warrants simplified models, and that n is approximately equal to the number of nodes.

2.1.2 Differences between approaches

Load-flow

Load-Flow analysis is inherently non-linear. The LU decomposition therefore must be performed multiple times during the course of an iterative algorithm such as the Newton-Raphson method. We will call the number of iterations k_{lf} .

Modified nodal analyses

MNA is a time-domain method, and even if we are interested only in the steady-state result, the transient solution must be computed from a known-state (typically at $t=0$, but alternatively starting from a previously computed operating point), until the state variables reach steady-state. The number of computations required to acquire the steady-state solution using MNA is therefore dependent on two variables: the settling time T_s before the system reaches steady-state, and the time-step t_s required to maintain mathematical stability during the solution. These numbers are typically inversely related; a system with low settling-time is one with a low time-constant which requires a small time-step to maintain stability, while a system with high settling-time has a high time-constant that allows for a larger time-step. We will therefore make the simplifying assumption that regardless of system response, approximately the same number of computations will be required in order to reach the steady-state solution. We will call this number m_{MNA} . APPENDIX A contains additional information regarding this method. MNA may also require multiple iterations per time-step; we will call this number k_{MNA} .

If all models in the MNA system are represented by linear equations only, each time-step can be solved with a single iteration, and $k_{MNA}=1$.

If any one model in the MNA system requires non-linear equations, the entire matrix is non-linear, and an iterative method is required. We assume the same method is used in both the load flow and transient solver implementations, so we can assume no performance difference with the algorithm, but since the equations being solved are different, k_{MNA} will still differ from k_{lf} , most likely.

2.1.3 Comparison between methods under different scenarios

Without loss of generality, we may consider only the case where we are solving for the steady-state solution of a single operating point, and ignore the quasi steady-state implementation of the load-flow solver. We can do this because the total simulation time is proportional to the total number of operating points of interest in both solvers; the load-flow solver is simply performing another solution of the same system, while the settling time of for the transient solution will be equivalent given changing input. Therefore, we have the following cases for comparison.

MNA linearity

Whether the load-flow solution is faster is entirely dependent on the ratio of the number of steps required to reach steady-state for the system (m_{MNA}) to the number of iterations required to converge the load-flow system (k_{lf}), where $m_{MNA} \cong 5 \cdot k_{lf}$ is the break-even point. The proportionality value of 5 comes from the fact that the load-flow solver is 5.191x slower than using MNA.

MNA non-linearity

Newton-Raphson convergence is quadratic in most cases, or linear when the solution roots have multiplicity greater than 1. The break-even point occurs at equation (2.1). Assuming the initial guess for the solution is roughly equally far away in both cases:

$$m_{MNA} \cdot k_{MNA} \cong 5 \cdot k_{lf} \quad (2.1)$$

If convergence is linear in MNA, but quadratic in the Power-Flow, the break-even performance point occurs at

$$m_{MNA} \cdot k_{MNA} \cong 5 \quad (2.2)$$

If convergence is quadratic in MNA, but linear in the Power-Flow, the break-even performance point occurs at equation (2.3), matching the case where MNA is linear.

$$m_{MNA} \cong 5 \cdot k_{lf} \quad (2.3)$$

Assuming convergence speed in both solvers proceeds at the same rate, the number of iterations in both solvers will be roughly the same, k_{lf} and k_{MNA} cancel each other out, and the break-even point is entirely dependent on m_{MNA} , with the break-even point being at approximately 5 steps.

Considering that our electrical discipline has components with complex impedances, the MNA is expected to require multiple steps before achieving steady state, and so the power flow solver is the best solver to use.

An implementation of a power flow solver modified to take into account state changes has been used to calculate the power losses in a microgrid while taking into consideration the time associated with the charging and discharging cycles of the energy storages [12].

2.2 INITIAL STATE OF SOLVERS

Initially in the project, we used two steady state solvers, a power flow solver to simulate electrical and machinery disciplines using Newton-Raphson or Gauss-Seidel methods to achieve a steady state, and a hybrid, modified nodal analysis solver in conjunction with a signal solver to simulate the thermal fluid disciplines [13].

The power flow solver models the load flow models as sources, loads, or transformers. Sources will set the voltage and power supplied, loads will set the amount of power draw or impedance, and the transformers are used to change the base voltage. In the electrical discipline, components can only be modeled as three-phase for the AC [14], because only that way can we ensure that all three phases are balanced, and thus, the impedances in all three phases must be equal. Meanwhile, the DC electrical and the machinery models are single phase.

The great advantage of using a power flow solver is that it does not depend on time, allowing it to provide the operating point while ignoring localized changes within the system. The disadvantages are that this solver does not take into consideration operating point changes, and that the power flow solver is not able to provide a solution for a system where a major voltage instability or voltage collapse has occurred. More details on solver limitations can be found in “Optimal Power Flow by Newton Approach” [14].

The hybrid cooling solver used by the cooling discipline, Thermal Fluid, consisted of a single step nodal analysis solver [3] communicating the fluid properties and directing the signal solver propagating the temperature. With this combination, the signal solver uses the nodal analysis solver to indicate the direction at which the temperature should move. The equipment models, as seen by the modified nodal analysis solver, comprise of flow sources, which can set the pressure or flow rate, and loads, which set a resistance to the flow. For the signal solver, the software models can be modelled as temperature sources, which set the initial temperature of the

simulation, and thermal loads, which are the models that add or remove heat in the system. Each engine calls both solvers before any result is determined.

Similar to the power flow solver, the hybrid cooling solver provides the steady state solution and thus on its own is not able to determine metrics that are time dependent.

2.3 QUASI-STEADY STATE SOLVERS

In order to solve for time dependent metrics, a new class, controller class, was created. The new class is responsible for keeping track of time and requesting that the solvers determine the solution for the time interval of interest. In order to determine the length of time to step, the controller class collects information from every time dependent equipment and uses the EDF [1] algorithm for scheduling time events. The EDF can be defined by equation (2.4).

$$U = \sum_{i=1}^N \frac{C_i}{T_i} \leq 1 \quad (2.4)$$

n – The number of calls to the solver by equipment software models inform the solver about when they need to be called is represented by n.

C_i – Worst-case simulation time for i^{th} step

T_i – Length of time to advance from the $i-1^{\text{th}}$ step to the i^{th} step (s)

U – Utilization bound

The value of time in the new step is assessable by every component, so they can use it to recalculate their time dependent metrics, and, if necessary, request a new step. The flow chart seen in Figure 2.1 illustrates how the stepping and metric updating process functions within our solvers.

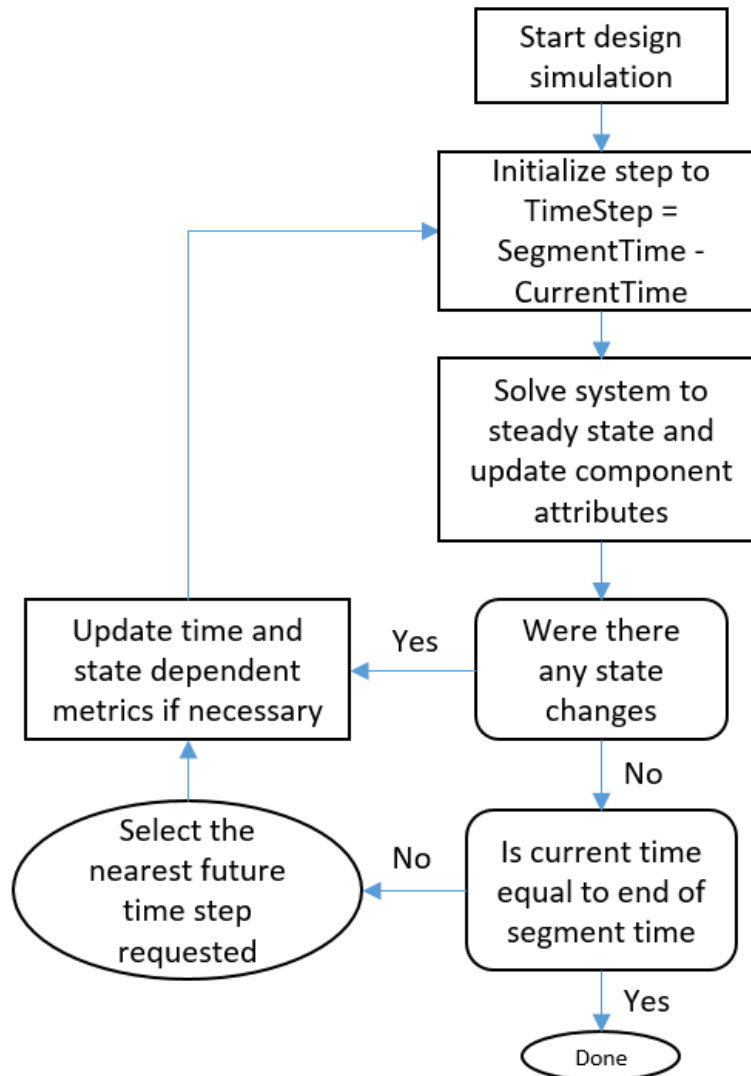


Figure 2.1 Component updates within a time segment

The following example illustrates how the process works when in the presence of multiple time dependent equipment.

E.g. Consider a system in which we have two energy storages providing power to a single load. In this example, we will consider two energy storages with different energy capacities and with instructions defining when changes between two of their

three modes, charging, discharging and offline, should occur within a single mission segment.

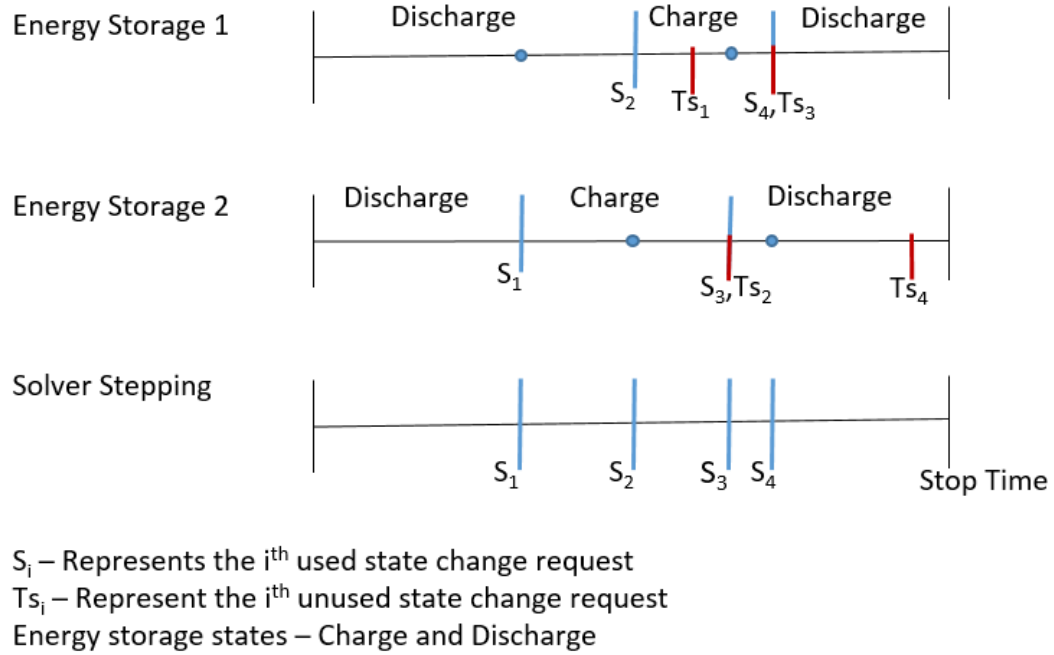


Figure 2.2 Stepping event example

Figure 2.2 illustrates the different calls made by each energy storage and the chosen stepping times. Initially, both energy storages are providing power to the load, and both inform the solver about the point in time that they need to change mode, either for having depleted their energy storage, which for this example, means that the energy storage should move into charging mode, or because it has reached full capacity, which for this example means that the energy storage should change its mode to the discharge. For S_1 , step 1, the energy storage 2, ES_2 , requests the shorter amount of time, and so that is the chosen time to step, while the prospective time point of ES_1 was a failed attempt, T_{S_1} . Due to ES_2 changing its mode to charging at S_1 , the ES_1 had to

provide extra power to the load, causing its mode change time to occur earlier, while the ES_2 charging time, T_{s2} , was discarded. S_3 was then set to the point at which ES_2 is fully charged and the attempted stepping point of ES_1 at T_{s3} was discarded. The next step, S_4 , is then chosen from ES_1 for being the next earliest set time, while ES_2 time at T_{s4} is discarded. Since the Mission Segment time ends before the stepping times set by both energy storages, both times are discarded and the final step is set to end at the mission segment end time.

2.4 ADDITION OF TOOLS FOR DEFINING MODE TRANSITIONS

With our power flow solver and modified nodal analyses solver now able to incorporate time, it was possible to implement the ability to create a tool that could provide the user with ability to define mode transitions. Because VTB has a database containing control blocks using a signal solver [15], our first approach was to use block diagrams. It soon became clear that for the size and complexity of the systems used by ESRDC, control blocks were not enough, due to the rate of growth of the number of control blocks when trying to build a more complex control system.

Due to this shortcoming of the block diagram approach, we instead created an approach where we are able to add software instructions directly into an equipment using the Python language. The idea of using Python was due to the fact that it was very simple to interface instructions written in Python within our framework. The addition of Python instructions provides one more option for users to define mode transitions in a systems. Because the written code is simulated together with a specific model, and it difficult to determine the solver call order between components, it is not advised to add

instructions that should immediately impact a different component, since, depending on the call order, the instructions may not be applied. This limitation restricts the Python approach to only rely on information provided by the same model for which it will be executed or information that does not change as the simulation proceeds. For example, the attributes associated with the limit value that some equipment has in the amount of power or current.

In order to utilize the Python code, the user must first change the attribute “Python Script” to true and double click the equipment. Figure 2.3 shows the interface window that allows the user to insert the intended instructions. The code inserted under OnSimulationStart is only accessed at the beginning of each mission segment and is mostly used for initializations, while the code inserted under SignalStep is accessed by the model before each step.

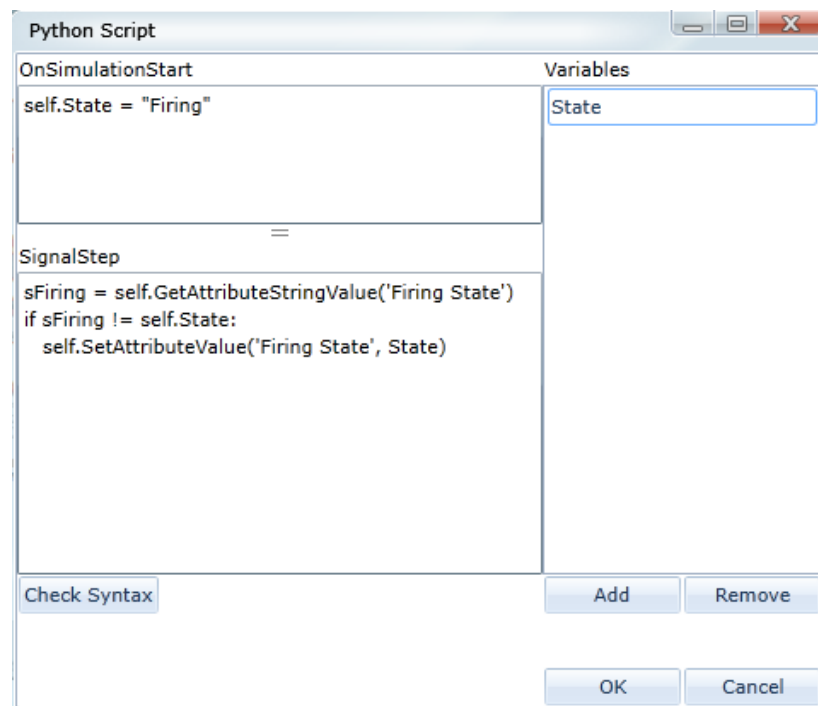


Figure 2.3 Python script tool

The Python script code in Figure 2.3 controls the firing mode of the Railgun seen in Figure 2.4. The railgun has three operating modes, firing, idle and offline. In this example, the Railgun is attempting to continuously fire for as long as enough power is provided to it. Because the electrical model used to represent the Railgun changes its mode to idle after each projectile fired, it is necessary to provide additional instructions in order to fire multiple projectiles in a single segment. In the Python script, we start by initializing the modes of the weapon to firing, which is done at OnSimulationStart. In the SignalStep section, the mode attribute value is read into the sFiring variable and it is determined whether the mode is set to firing. If it is not, we change the mode of the weapon back to firing. This script could be used to determine metrics such as the maximum rate of fire, continuous rate of fire, or the number of shots that could be fired using the current Generator.

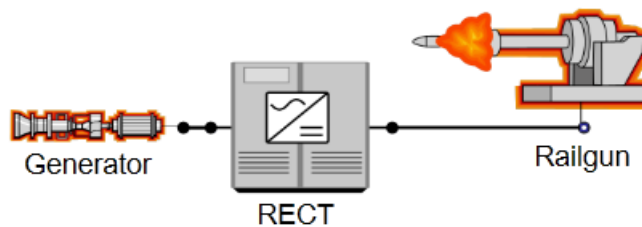


Figure 2.4 Python code system example

CHAPTER 3

COMPONENT DEVELOPMENT

In order to take advantage of the fact that the solver is now able to track time and take into consideration operating point changes, most of the component models need to be altered; the exceptions are the models that do not have any parameters that can be changed during the mission, such as the cable, shaft, gear box, and pipes. With those changes, the component model is now able to provide information regarding time dependent metrics, to handle discrete state changes, and to allow communications between the different discipline models of an equipment.

3.1 DISCRETE STATE AND TIME DEPENDENT MODIFICATIONS

Operating point changes were incorporated in the solver with the help of a controller class, enabling software models to use time when calculating time-dependent metrics. In the case of an operating point change, the solver is requested to run a second time. In that additional run, it will consider the component's new discrete state (3.1). An example can be seen with the behavior of a breaker, because the system can only know the amount of current passing through the breaker after it has been solved. One cannot obtain the answer of the system after the breaker has opened unless the system is solved a second time with the breaker tripped. The ability to define transition modes using Python code made it necessary to expand this behavior to not only all

equipment that can change its operating point, but also to some other attributes that could change the models operating state. The gate valve is such an attribute. We can change the value of the 'Valve Level' attribute in order to control flow. Instructions could be given to increase the flow to try to reduce the temperature in a certain branch.

The electrical percent charging power SOC curve is an example of an attribute with which we use the relationship between two metrics to determine its operating point. In this attribute, the relationship between power and state of charge (SOC) is used to determine the amount of power consumed by the energy storage, considering the amount of charge it currently has. Figure 3.1 show the electrical percent charging power SOC curve attribute. For more about the definitions of those attributes, see APPENDIX B.

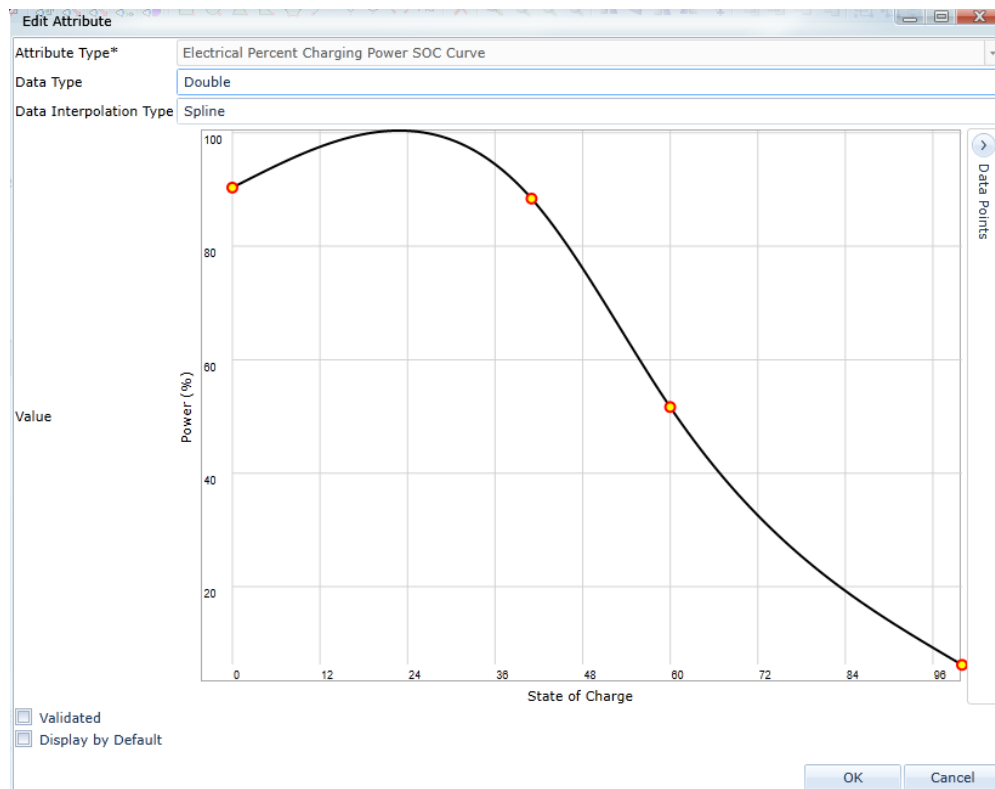


Figure 3.1 Electrical percent charging power SOC curve attribute

Next, we will take a closer look at the components requiring the most changes: the energy storage, the railgun, and the tank. Those merit specific focus because they are the only components that currently set requests of time-dependent events.

3.1.1 Energy storage

In order to better represent the energy storage energy cycles, multiple changes were needed.

With the help of equations (3.1) to (3.4), we are able to determine the amount of time left before the energy storage is depleted or becomes fully charged. The calculated time is then compared with the last time step, and if inferior, it becomes our new time step.

To calculate energy storage charging time,

$$E_c = P_c * (t_c - t_p) * E + E_p \quad (3.1)$$

$$T_{Step} = \left(\frac{E_{sc} - E_c}{P_c * E} \right) \quad (3.2)$$

To calculate energy storage discharging time,

$$E_c = P_i * (t_c - t_p) * E + E_p \quad (3.3)$$

$$T_{Step} = \left(\frac{E_c}{P_i * E} \right) \quad (3.4)$$

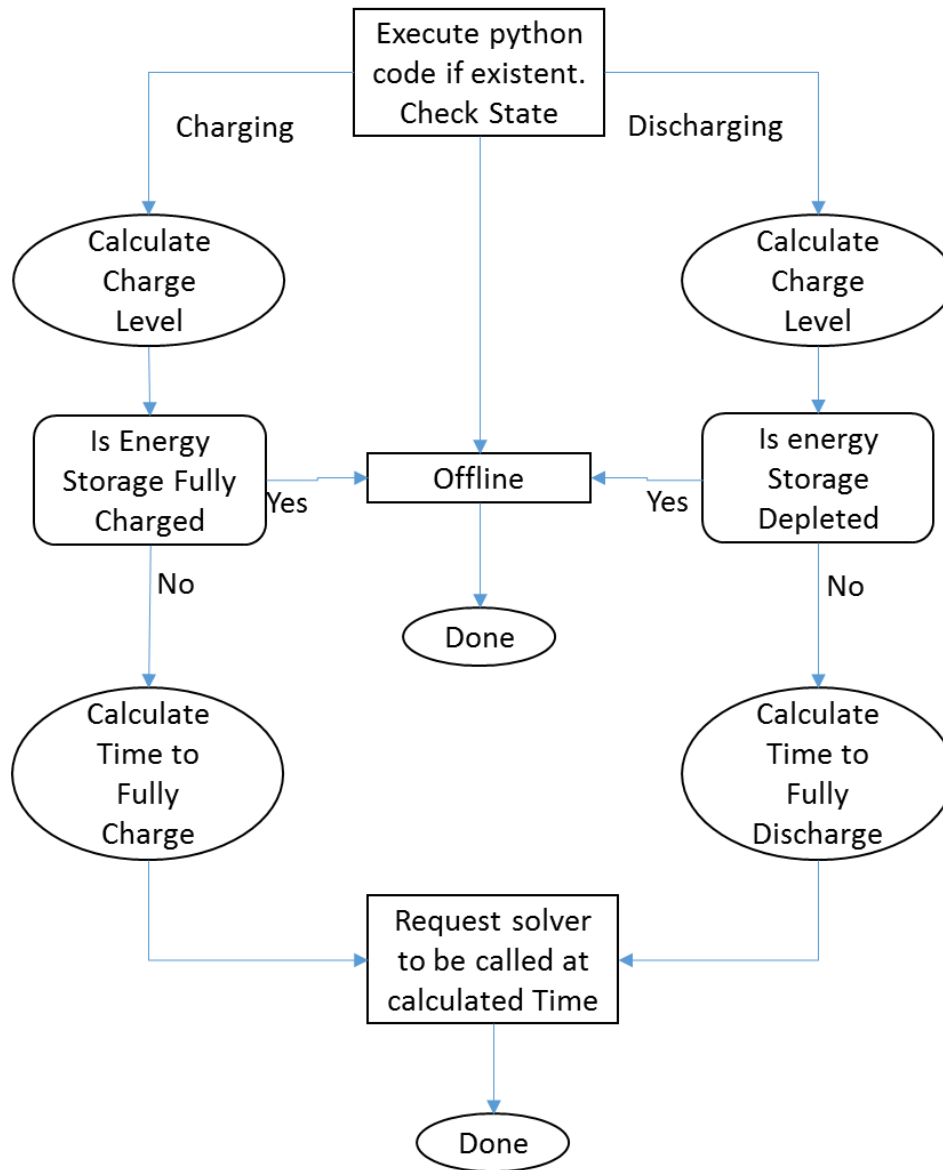


Figure 3.2 Energy storage electrical model flow-chart

Figure 3.2 is a flowchart illustrating how the energy storage electrical model determines if the energy storage is required to change mode and when a change of mode might be needed. The Python code is called at the beginning, so that changes can be made at the commencement of each step.

3.1.2 Railgun

The railgun model is a model of a single shot energy gun with an internal energy storage. Because it has an internal energy storage, it has equivalent equations to those of the energy storage to calculate the time necessary to charge the gun. In addition to charging time, the railgun has an attribute named cool down time that ensures that the equipment had enough time to cool between shots.

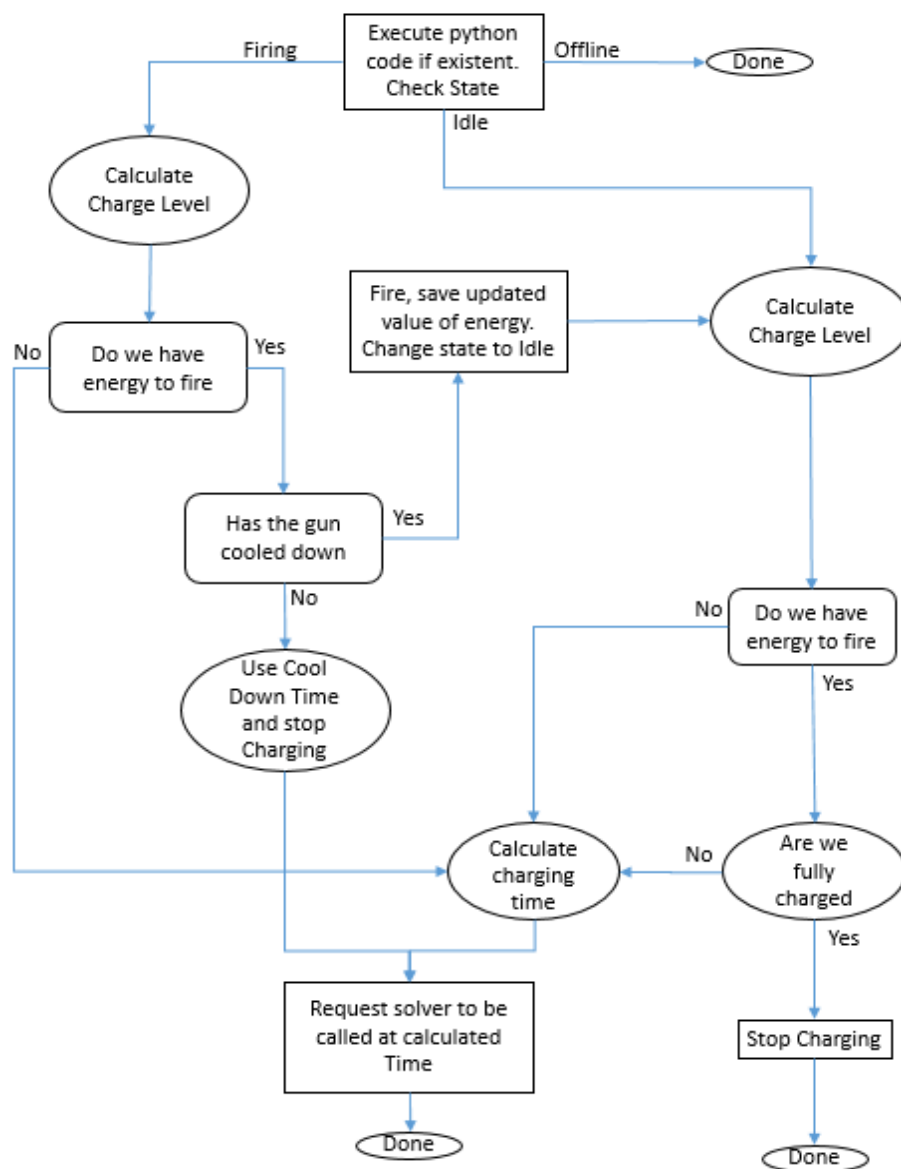


Figure 3.3 Railgun electrical model flow-chart

The flow chart of Figure 3.3 displays how the state and time to step are determined in the electrical model of a railgun.

3.1.3 Tank

The tank's model, consists of one inlet and one outlet. The tank is responsible for setting the initial temperature of the fluid and the type of fluid used. Improvements were made to the tank in order to determine how much fluid has been used, and whether it contains enough fuel for a predetermined mission. In order to model tanks that able to be connected in series, we added the ability for the tank to be filling and draining simultaneously. In order to implement the unlimited fuel option, we gave the tank the ability of instantaneously refueling. Equations (3.5) to (3.7) were used to calculate the time necessary to empty or fill the tank.

$$T_{AddFuel} = M_{FRate} * (t_c - t_p) \quad (3.5)$$

$$T_{TFuel} = T_{TFuel} - T_{AddFuel} \quad (3.6)$$

$$T_{Step} = \frac{T_{TFuel}}{M_{FRate}} \quad (3.7)$$

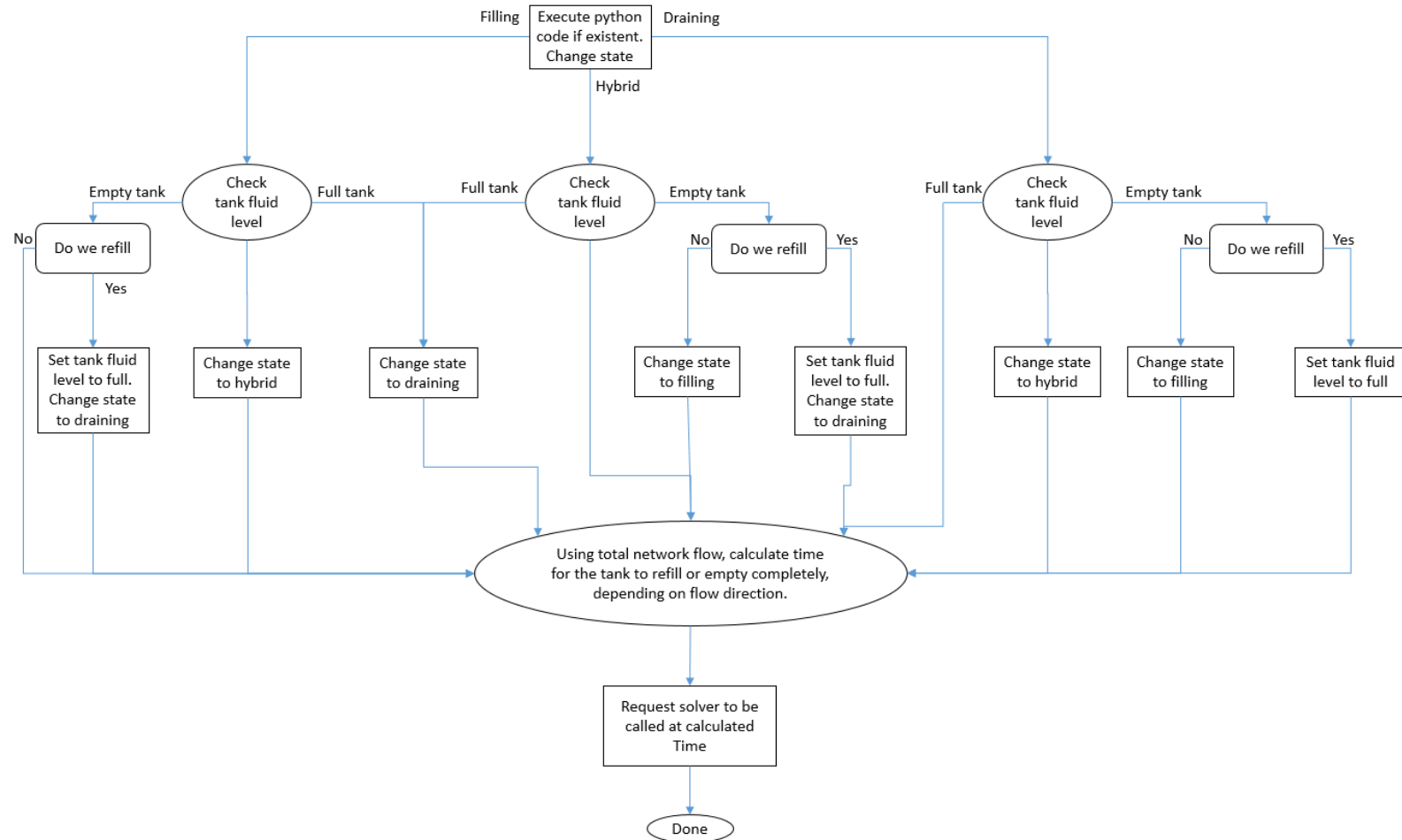


Figure 3.4 Tank model flow-chart

Figure 3.4, shows what questions the model representing the tank asks in order to determine if the tank is required to change mode and when the next event time will occur.

3.2 COMMUNICATION ACROSS DISCIPLINES

In order to make more realistic simulations where interdependencies between disciplines are accounted for, the different representations of each equipment had to be linked with each other. To be able to share information from different models of a single equipment a few modifications to the engines were necessary. In the electrical and machinery disciplines, every load now saves the power consumed, while every source saves the power provided, and in the cooling disciplines, the electrical dependent models save the value of electrical power that they require to function. Those attributes are used to determine a variety of quantities, such as the amount of fuel an engine requires, the amount of heat that is required to be dissipated, and the electrical power needed.

In case any of the attributes change, where the value is shared between models of an equipment, a flag is thrown to inform the mission analyzer that the dependencies of the equipment have not been satisfied.

Because the name of the equipment is unique across disciplines, it is easy to recognize each model being linked to a single equipment. The system shown, with Figure 3.5 representing the electrical discipline and Figure 3.6 representing the machinery discipline, was created to help understand such cross discipline communication. When both disciplines are simulated, a warning was thrown in the

machinery discipline by the motor. The exception 'Insufficient electrical power provided' warned the user that the motor did not receive enough electrical power, when the electrical discipline ran, to be able to provide the power requested by the motor. This occurred because the motor did not have the mechanical power required to turn the generator, due to not receiving any electrical power.

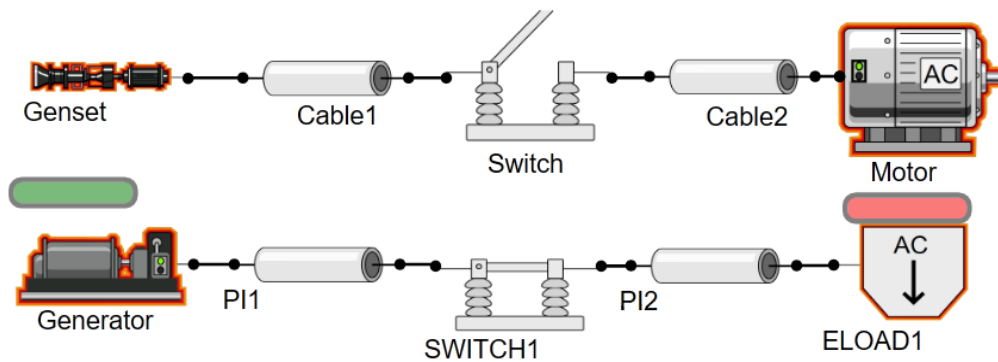


Figure 3.5 Electrical discipline

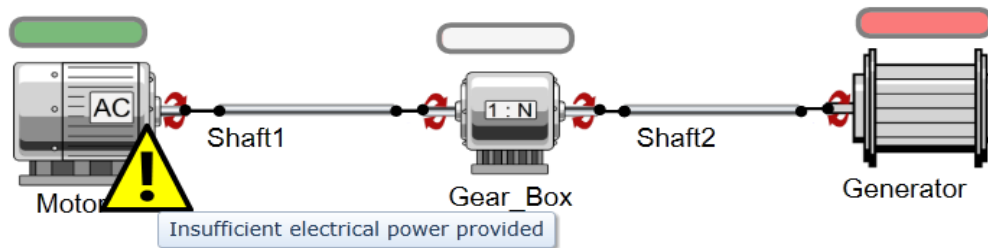


Figure 3.6 machinery discipline

CHAPTER 4

MISSION ANALYSER

The mission analyzer is a tool created to simulate and provide results for a mission, taking into account the pre-determined alignments for every discipline. With the help of the mission simulator, we can also use the mission analyzer to simulate any number of mission segments, providing a report on the success of the mission as well as certain metrics such as fuel consumed.

The mission analyzer has two modes of operation, it can either run the current design alignments or a mission created using the mission simulator. The current design is the system alignment that is currently created at the user's project design. The ability to simulate the current design allows the user to test the performance of a layout before deciding to use it in a mission. When solving the current design, the user is required to provide a time and duration of the mission as seen in Figure 4.5, whereas when using the mission analyzer tool to run a mission, those metrics are part of the mission segment.

The mission analyzer provides a variety of messages to the user to inform them about changes of mode, warning advice, and information about failure events, such as the ship running out of fuel. Those messages are labeled with three distinct levels, information, warning and failure errors. The message type information is used to

provide information about events the user might know about such as a weapon firing or an energy storage changing mode from charge to discharge. Warnings are messages with the objective of alerting the user about a problem in the system, where the problem should not be major enough to compromise the mission, such as a weapon not being able to fire. A failure error is used to inform the user about a problem existent in the system which compromises the success of the mission. When in the presence of a failure error, the simulation is interrupted, the mission is aborted, and an alert about the cause of the error is provided to the user as we see in Figure 4.5. The user is provided with options to change any information level from any of the three level to another.

4.1 CONVERGENCE ALGORITHM

In order solve the interdependencies equipment of different disciplines, a convergence method capable of establishing the optimal order for solving each of the disciplines was needed. An algorithm designed to determine the installation order of software packages having dependencies on other packages in a Linux system was studied and adapted for our own use [16]. The chart from Figure 4.1 describes how the decision is made regarding which discipline needs to be solved next. An example of how the interdependencies are chosen can be seen in APPENDIX C.

In order to implement this algorithm, every equipment having models in different disciplines were given the capacity of informing the mission analyzer tool about whether or not their dependencies have been satisfied. Section 3.2 provides more information on such communication across disciplines.

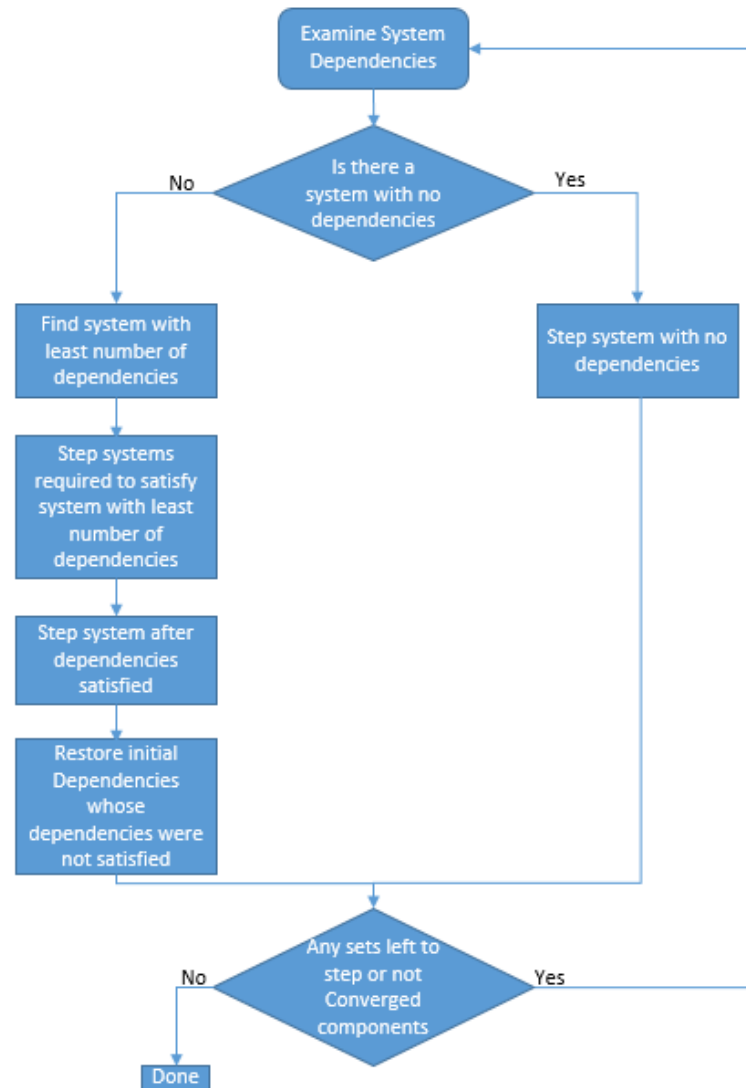


Figure 4.1 Resolving solver dependencies

4.2 MISSION ANALYZER WALK-THROUGH

The user interface when first accessing the mission analyzer is seen in Figure 4.2, where an option between simulating a mission or the existing ship design configuration is given. This interface also provides the user with a list of possible missions to choose. When a mission is selected, a list of its segments is displayed.

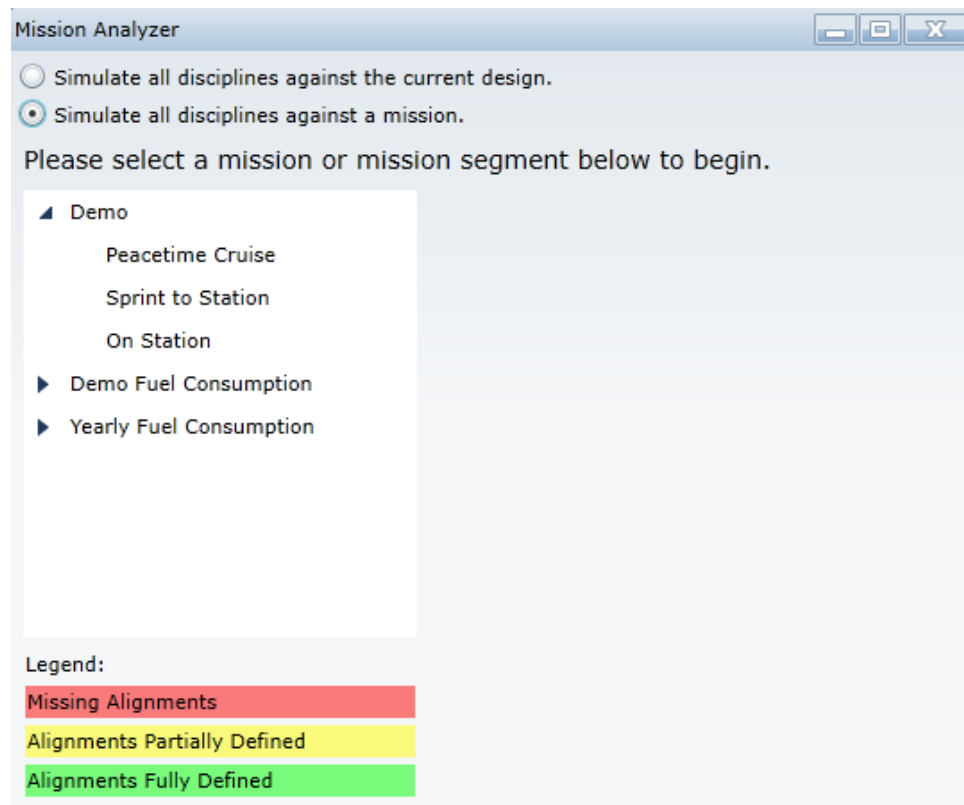


Figure 4.2 Mission Analyzer principal page

If the option to simulate all disciplines against the current design is chosen, the tool will bring the user to a new window, Figure 4.3, where the user can view the operating state of each discipline. At this stage, the tool labels disciplines that are loaded and have every equipment placed in the system as green, and disciplines that either have not been loaded or have unplaced equipment as yellow.

The user is able to check for details to know the reason that a discipline alignment is labelled yellow, and if there is something that needs to be changed, the user can select Open Designer. This will allow the user to have direct access to the desired design (Figure 4.4) and have a chance to modify any operating state. The user can also ignore the warning and simply move into the simulation tab.

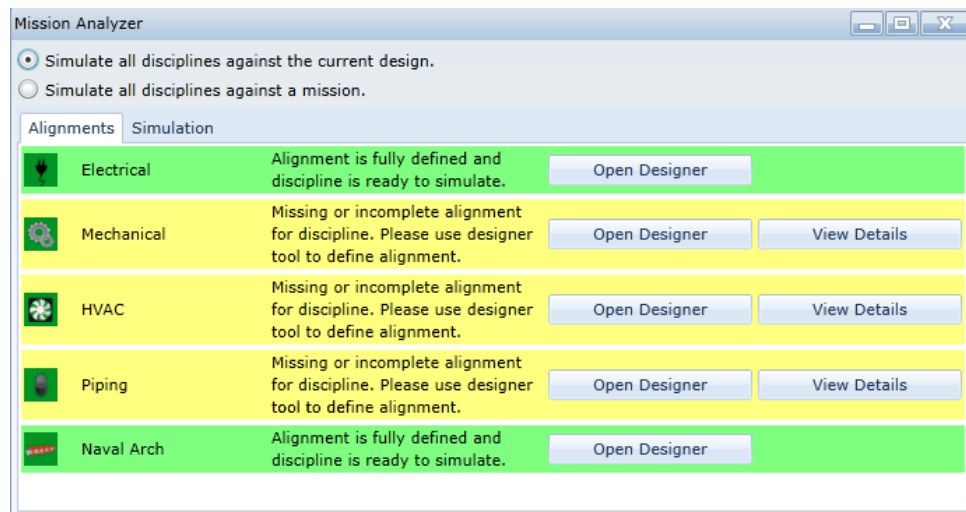


Figure 4.3 Alignments setup

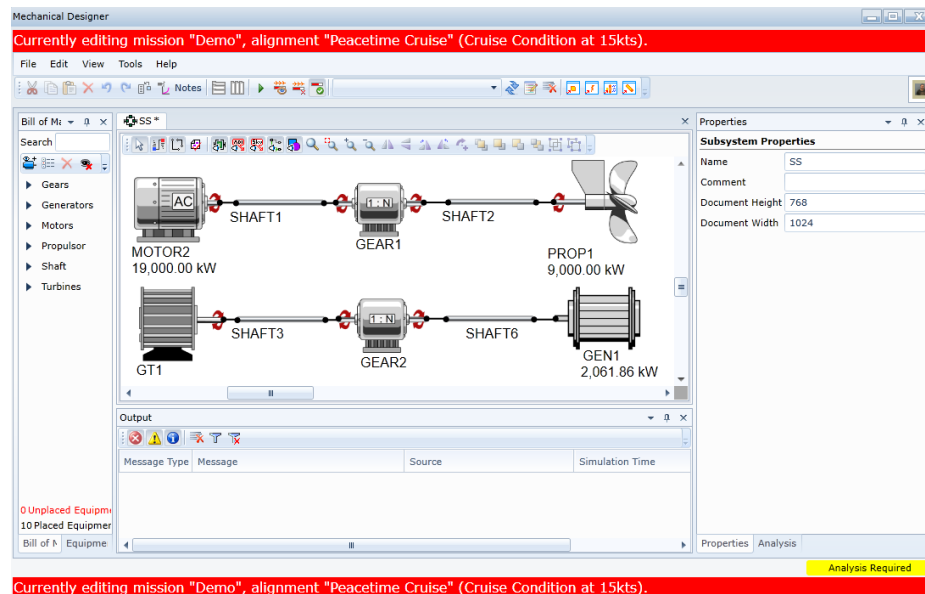


Figure 4.4 Setting an alignment

Figure 4.5 shows the graphical interface of the mission analyzer that allows the user to run the simulation of the existing alignments without the need of defining a mission. In this window, the user is expected to provide the amount of time the design is expected to run, and the speed at which the ship should be travelling. In order to run the mission, the ► symbol must be pressed, and the symbol to the right is used to open the interface seen in Figure 4.6. When the system has finished running a list of messages regarding information results, warnings, and possible errors is provided. The message levels are distinguished by color code: black for information, yellow for warnings, and red for failure errors.

If the user wishes different model behavior during certain simulation events for a particular mission, he is able to edit the level of assignment. In Figure 4.6, we are changing the message level to information when the provided power by a generator is greater than its rated power. The user could want such behavior so that he could simulate the mission even if he had placed a generator that is not powerful enough to provide the power required to run the current electrical system. This can either be done to every equipment under the generator category, to that equipment type “-Generators liquid cooled,” or we could further expand and edit the change to a very specific equipment of that type. In the top right corner of the window, the tool displays the values of time simulated, fuel consumed, and distance traveled.

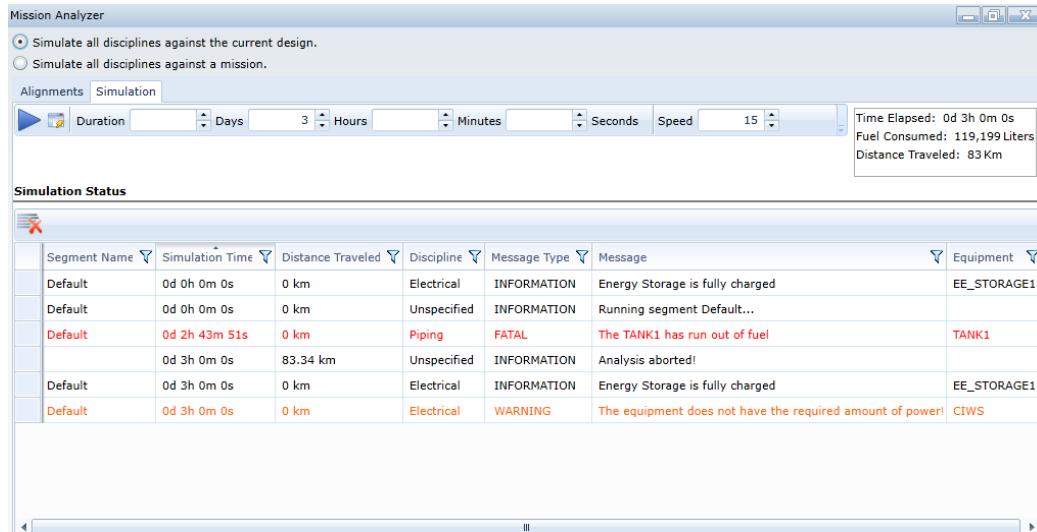


Figure 4.5 Output display window, current configuration

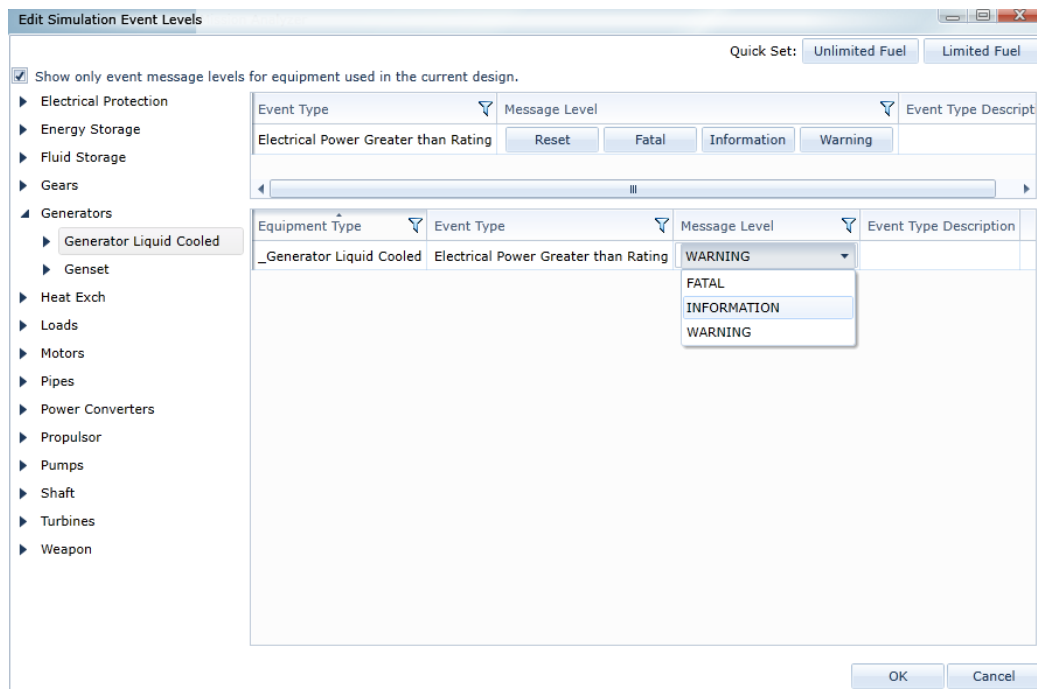


Figure 4.6 Changing message warning level

4.3 MISSION EXPLORER WALK-THROUGH

The mission explorer is a tool used to define missions and mission segments. The mission explorer can be accessed for simulation for the user-defined ship designs. An

example of the mission explorer user interface can be seen in Figure 4.7. In this example, we have four missions created, Default, Demo, Demo Fuel Consumed, and Year Fuel Consumption. For each of those missions, we have created mission segments that can be accessed by expanding under the mission of interest. In this example, we are examining the Demo mission, which has 3 mission segments. When selecting a segment, we can define the duration of the segment and the ship speed, or we can use the map to select the desired distance and derive the speed from that.

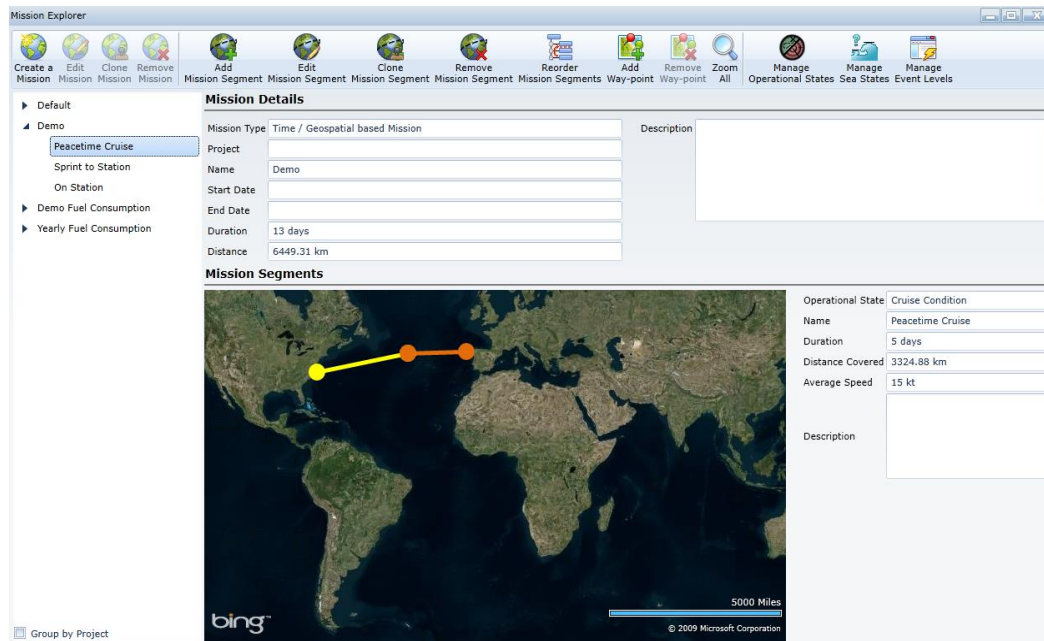


Figure 4.7 Mission setup

Using the duration and speed provided by the mission explorer, the mission analyzer can now be used to define each alignment and compute the operating points.

In Figure 4.8, we can see the mission analyzer in the mode of simulating a mission. When looking at missions, the user can simply load the current design, or manually edit the operating point of each equipment to the corresponding value for the

segment. If a segment does not have information about the operating point of every equipment for an alignment, the segment and mission will be underlined with a red color. If there are one or more alignments with unplaced equipment, they will be underlined yellow and if all equipment is placed and information on operating point is loaded, they are underlined green.

The Simulation tab (Figure 4.9), is very similar to the one used to display the results from simulating all disciplines against the current design (Figure 4.5).

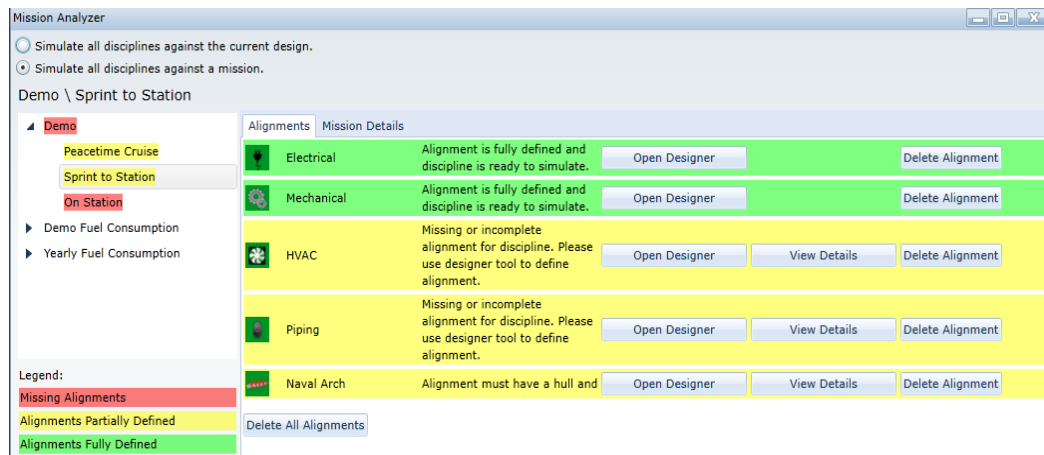


Figure 4.8 Segment setup

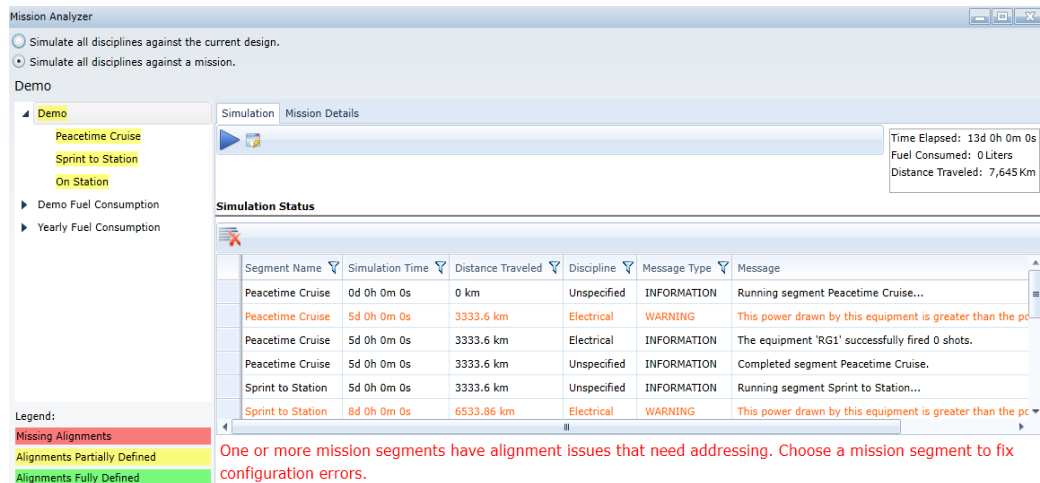


Figure 4.9 Full mission simulation output display window

4.4 MISSION EXAMPLE

In this example, we will be looking at the system configuration defined in Figure 4.10 through Figure 4.13. The system design starts at the Naval arc in which the hull, radars, weapons systems, and propeller are selected. For this example, we chose a USV, a single propeller, a bow mounted sonar, a phased array radar, and a CIWS weapon (Figure 4.10).

Responding to this design specification, in the machinery discipline, we constructed a small system where an AC motor provides the power to the propeller (Figure 4.11).

In the electrical discipline two gensets supply power to a 6.9KV DC bus, from which the loads take the necessary power, and an energy source provides the power to the weapon. Because of the varying voltages and current type in use, a few power converters were added (Figure 4.12).

In the thermal fluid discipline, we created a fresh water loop cooled down by a heat exchanger that dissipates some of the heat into the sea with the help of a pump. The thermal fluid discipline also has a fuel tank that provides F76 fluid to both gensets (Figure 4.13).

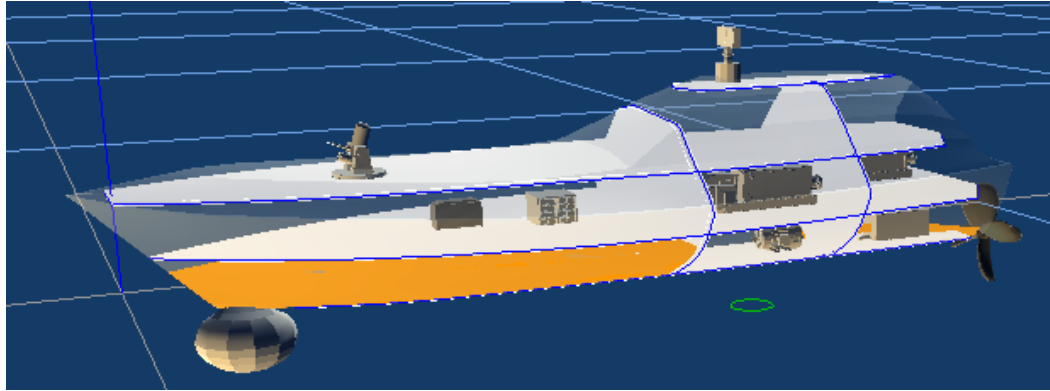


Figure 4.10 Naval arc design ship representation

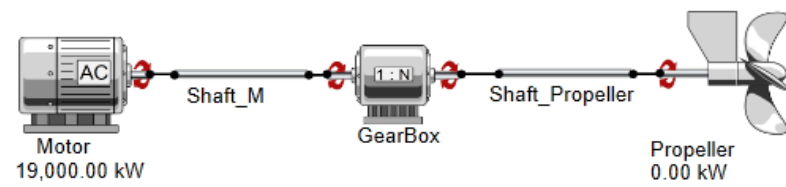


Figure 4.11 Propulsion system schematic

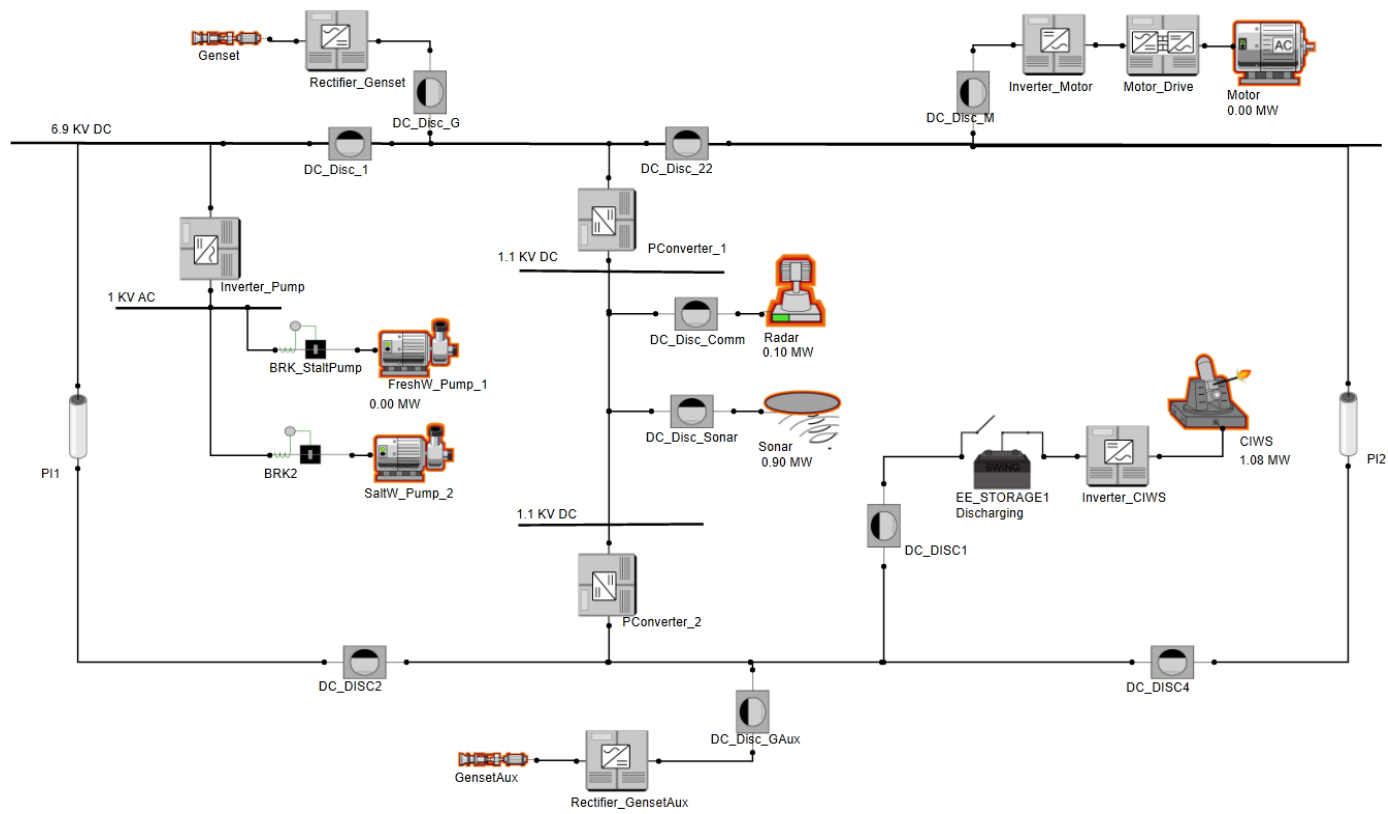


Figure 4.12 Electrical system schematic

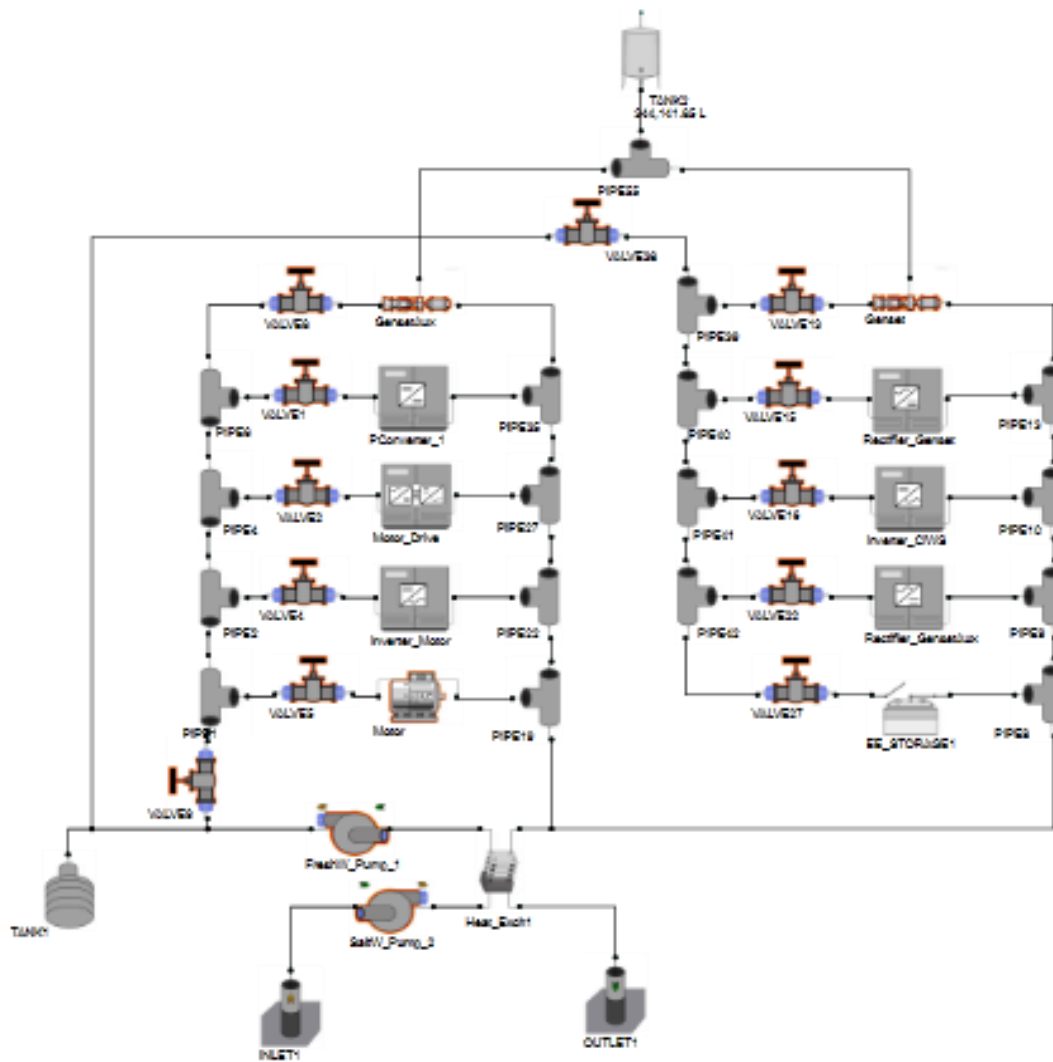


Figure 4.13 Thermal fluid system schematic

Once the system is completed, we want to test it using the mission Demo seen in Figure 4.7. This mission contains three mission segments, Peacetime Cruise, Sprint to Station, and On Station. The Peacetime Cruise segment takes 10 days at a speed of 15 knots, the Sprint to Station segment takes 5 days at a speed of 24 knots, and the On Station segment takes 15 days at a speed of 2 knots. In order to run the mission, we need to first set the operating point of each component at each individual mission

segment. The bow sonar will be online for the duration of the mission. The radar, modeled with three possible power outputs, will be at its maximum of 0.9MW during the Sprint to Station segment, at medium power of 0.5MW during the Peacetime Cruise segment, and minimum power of 0.1MW during the On Station segment. For this mission, the weapon does not need to fire.

Because it is a very long mission, we will use the unlimited fuel option in order to learn about when the ship would require refueling.

Simulation

Mission Details

Time Elapsed: 30d 0h 0m 0s

Fuel Consumed: 24,230,522 Liters

Distance Traveled: 13,334 Km

Simulation Status

Segment Name	Simulation Time	Distance Traveled	Discipline	Message Type	Message	Equipment
Peacetime Cruise	10d 0h 0m 0s	6667.2 km	Electrical	INFORMATION	Energy Storage is fully charged	EE_STORAGE1
Sprint to Station	10d 0h 0m 0s	6667.2 km	Electrical	INFORMATION	Energy Storage is fully charged	EE_STORAGE1
Sprint to Station	15d 0h 0m 0s	12000.96 km	Electrical	INFORMATION	Energy Storage is fully charged	EE_STORAGE1
On Station	15d 0h 0m 0s	12000.96 km	Electrical	INFORMATION	Energy Storage is fully charged	EE_STORAGE1
On Station	30d 0h 0m 0s	13334.4 km	Electrical	INFORMATION	Energy Storage is fully charged	EE_STORAGE1
Sprint to Station	15d 0h 0m 0s	12000.96 km	Electrical	WARNING	This power drawn by this equipment	Inverter_Motor
Peacetime Cruise	1d 5h 16m 13s	813.13 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	2d 10h 32m 27s	1626.27 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	3d 15h 48m 40s	2439.4 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	4d 21h 4m 54s	3252.53 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	6d 2h 21m 7s	4065.66 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	7d 7h 37m 21s	4878.8 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	8d 12h 53m 35s	5691.93 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Peacetime Cruise	9d 18h 9m 48s	6505.06 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Sprint to Station	10d 22h 11m 7s	7653.3 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Sprint to Station	11d 20h 22m 15s	8639.4 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Sprint to Station	12d 18h 33m 23s	9625.5 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Sprint to Station	13d 16h 44m 31s	10611.61 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
Sprint to Station	14d 14h 55m 39s	11597.71 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
On Station	19d 4h 42m 38s	12373.99 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
On Station	23d 9h 25m 17s	12747.03 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1
On Station	27d 14h 7m 56s	13120.06 km	Piping	INFORMATION	The TANK1 needs to be refueled	TANK1

Figure 4.14 Mission results

Figure 4.14 shows the results from each segment as well as of the full mission fuel consumed, distance traveled, and duration. It also gave us information about events such as the tank needing to be refueled, and problems that occur during the mission such as the fact that we need an inverter with a higher power capacity to be connected with the motor in order to move the ship at 24knots.

Looking into the information provided by the tank, we can find the relationship between the fuel consumed and power consumed by propeller and radar. Table 4.1 shows the relationship existent between the power utilized by the radar and the speed of the ship with time needed to deplete the fuel tank.

Table 4.1 Power and fuel consumption

Mission Segment	Radar (MW)	Speed (Knots)	Tank (hours)
On station	.1	2	100.1
Peacetime cruise	.5	15	29.3
Sprint to station	.9	24	22.1

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In this thesis, I presented three principal contributions to the field of ship design. They are the development of a controller class that allows steady state solvers to provide time-dependent information, the incorporation of operating point changes and time-dependent metrics at the component level, and the adaptation of a software installation ordering algorithm to develop a method for efficiently solving a system while satisfying the interdependencies between disciplines

Steady state solvers are able to provide quick solutions to system simulations that are not dependent of time, but struggle when system components experience localized operating point changes. By developing a controller class responsible for keeping track of time and relaying the necessary time-dependent information to the solvers, we are able to determine solutions to time intervals of interest. In this way, we are able to handle time intervals where operating point changes occur, while still retaining the ability of the steady state solvers to quickly handle the long periods of time where no discrete state changes are occurring. Essentially, this allows the length of a mission to be partitioned into segments that can be solved quickly and accurately.

With the solvers now able to utilize time-dependent information in order to track operating point changes within the system, we allow equipment to provide time dependent information and to undergo discrete state changes. By relaying such information to the controller class, the equipment is able to communicate across disciplines. This sharing of information allows for the solvers to take into account the interdependencies between disciplines when providing solutions.

We gave every equipment having models in multiple disciplines the capability of informing our mission analyzer tool if their dependencies have been satisfied. This allowed us to implement an algorithm that determines the order for how the disciplines are solved such that their interdependencies are taken into account with minimal solver calls, providing accuracy and efficiency in our simulation.

5.2 FUTURE WORK

In order to propagate changes that occur between disciplines during a mission segment, we will need to run all of the disciplines simultaneously, under the same time step, exchanging information between each other at every step. This will improve the accuracy of the results and allow the user to better predict how some events that happen in one discipline can affect another discipline.

A better method of implementing commands at a higher level is something we should strive to create. This should be a tool with a simple interface with every component that would allow for implementation of control systems and easy exportation of those same instructions into other similar ship designs.

We would also like to create a tool that allows the comparisons between multiple ship designs against a mission or segment. This tool should use the values calculated by the mission analyzer and, based on the metrics of interest, compare any number of designs in the project.

REFERENCES

- [1] G. Lipari, "Earliest Deadline First," Scuola Superiore Sant'Anna, Pisa-Italy, 2005.
- [2] C.-W. Ho, P. A. Brennan and A. E. Ruehli, "The Modified Nodal Approach to Network Analysis," *IEEE Transactions on Circuits and Systems*, vol. 22, no. 6, pp. 504-509, 1975.
- [3] Association for Computing Machinery, "Modeling and simulation glossary," [Online]. Available: <http://www.acm-sigsim-mskr.org/glossary.htm>.
- [4] B. Bras and F. Mistree, "Designing Design Processes in Decision-Based Concurrent Engineering," *SAE Transactions, Journal of Materials & Manufacturing*, vol. 100, pp. 451-458, 1991.
- [5] R. A. Dougal and B. Langland, "Catching It Early," *Marine Technology*, pp. 63-69, January 2016.
- [6] J. Chalfant, B. Langland, S. Abdelwahed, C. Chrysostomidis, R. Dougal, A. Dubey, T. E. Mezyani, J. Herbst, T. Kiehne, J. Ordonez, S. Pish, S. Srivastava and E. Zivi, "A Collaborative Early-Stage Ship Design," ESRDC Technical Report, 2012.
- [7] M. Engelhardt, "SPICE Differentiation," *ELECTRONICS WORLD*, vol. 121, no. 1946, pp. 16-21, 2015.
- [8] F. Milano, "An Open Source Power System Analysis Toolbox," *IEEE TRANSACTIONS ON POWER SYSTEMS*, vol. 20, no. 3, pp. 1199-1206, 2005.
- [9] H. Anton and C. Rorres, in *Elementary Linear Algebra: Applications Version*, John Wiley & Sons, 2010, pp. 477-486.
- [10] D. Coppersmith and S. Winograd, "Matrix multiplication via arithmetic progressions.," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, New York, 1987.
- [11] Y. Levron, J. M. Guerrero and Y. Beck, "Optimal Power Flow in Microgrids with Energy Storage," *IEEE Transactions on Power Systems*, vol. 28, no. 3, pp. 3226-3234, 2013.
- [12] S. A. Nasar and F. C. Trutt, in *Electric Power Systems*, vol. 28, CRC Press, 1998, pp. 72-87.

- [13] D. I. Sun, B. Ashley, B. Brewer, A. Hughes and W. F. Tinney, "OPTIMAL POWER FLOW BY NEWTON APPROACH," *IEEE Transaction on Power Apparatus and Systems*, Vols. PAS-103, no. 10, pp. 2864-2880, October 1984.
- [14] X. Wu, H. Fuguroa and A. Monti, "Testing of Digital Controllers Using Real-Time Hardware in the Loop Simulation," *Power Electronics Specialists Conference*, 2004. PESC 04. 2004 IEEE 35th Annual. Vol. 5. IEEE, 2004.
- [15] D. Burrows, "Modelling and Resolving Software Dependencies," *Conferenza Italiana sul Software Libero (CONFSL 2010)*, 2015.
- [16] C. Dufour, J. Mahseredjian and J. Bélanger, "A Combined State-Space Nodal Method for the Simulation of Power System Transients," *Power Delivery, IEEE Transactions*, vol. 26, no. 2, pp. 928-935, 2011.

APPENDIX A: DIFFERENCES BETWEEN POWER LOAD AND MNA

In order to examine the equations discussed in Section 2.1 , we need to first understand the difference in number of iterations between linear convergence and quadratic convergence when using Newton-Raphson. Figure A.1 illustrates the relationship between the number of iterations needed and how far the method is from the correct solution. The Y-axis, representing the distance between the calculated value and the convergence value, grows exponentially. In the example given by Figure A.1, we start at the same distance from the convergence value with regards to error. The quadratic system takes 5 iterations to converge while the linear system takes 25. From the graph, we determine that the relationship can be represented by X iterations for linear convergence, and \sqrt{X} for quadratic.

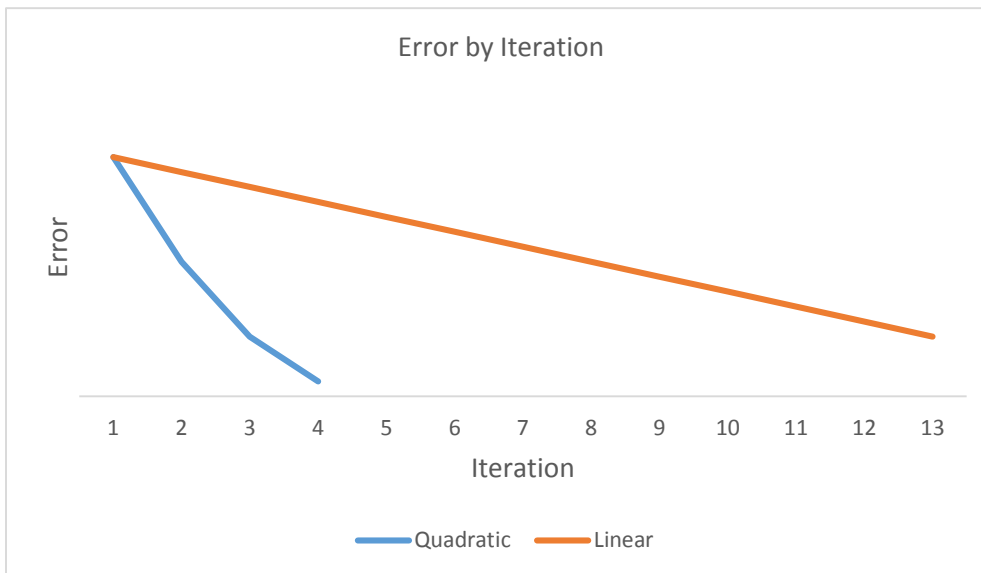


Figure A.1 Difference between Newton-Raphson linear and quadratic convergence

To refer to the number of iterations we used K_{MNA} for MNA method and K_{lf} for load flow, with exception being when the system lacks nonlinear equations. In that case, the MNA requires more than one computation before it finds the steady state solution of the system, and we use m_{MNA} to refer to the number of computations. In Figure A.2, we see a representation of how the solution varies at each computation in MNA until a steady state solution has been reached.

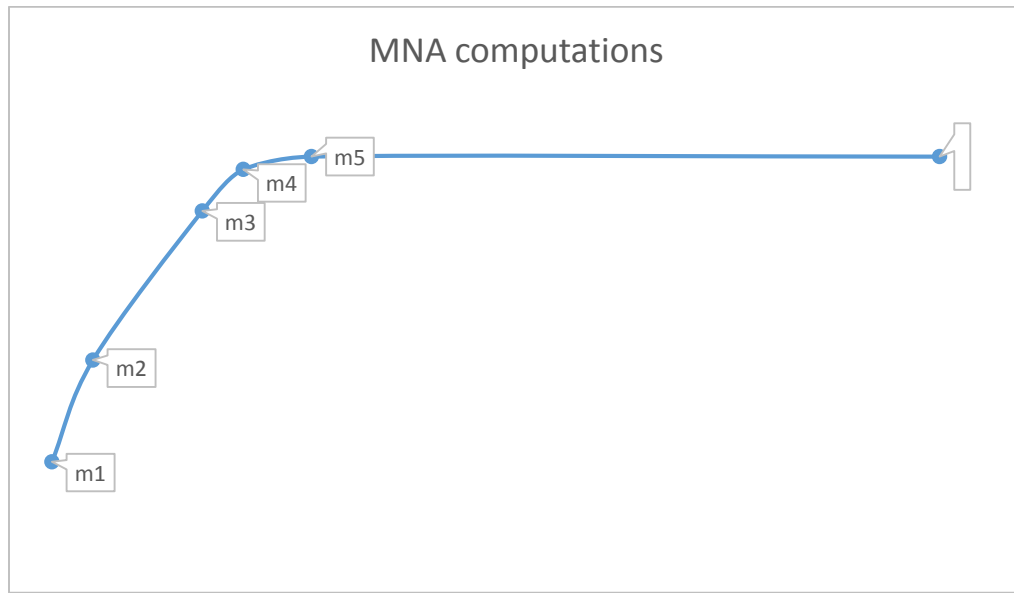


Figure A.2 Representation of MNA computing into steady state

Looking at Section 2.1 we analyze three cases where MNA is non-linear.

In the first case represented by equation (2.1), both methods, MNA and power-load, achieve the same convergence, being linear or quadratic $m_{MNA} \cdot k_{MNA} \cong 5 \cdot k_{lf}$.

In the second case, MNA is linear but power-flow is quadratic, so K_{MNA} grows exponentially faster than K_{lf} therefore we have $m_{MNA} \cdot \frac{k_{MNA}^2}{k_{lf}} \cong 5$, and considering our initial assumption that $K_{MNA} \approx K_{lf}$, we get equation (2.2), $m_{MNA} \cdot k_{MNA} \cong 5$.

In the third case, MNA is quadratic and power-load is linear, for this case K_{lf} grows exponentially faster than K_{MNA} , $m_{MNA} \cdot \frac{k_{lf}^2}{k_{MNA}} \cong 5$, and we get equation (2.3),

$$m_{MNA} \cong 5 \cdot k_{lf}$$

APPENDIX B: MULTI INPUT ATTRIBUTES

In order for our simulation to correctly simulate the behavior of certain metrics such as efficiency, fuel consumed, and charging power, we introduced an attribute type that allowed the input of multiple data points. Those attributes can use a variety of interpolation methods to connect the given points. Figure B.3 shows us how the spline interpolation methods to connect the given points. Figure B.3 shows us how the spline function is being used to determine the specific fuel consumption versus power curve.

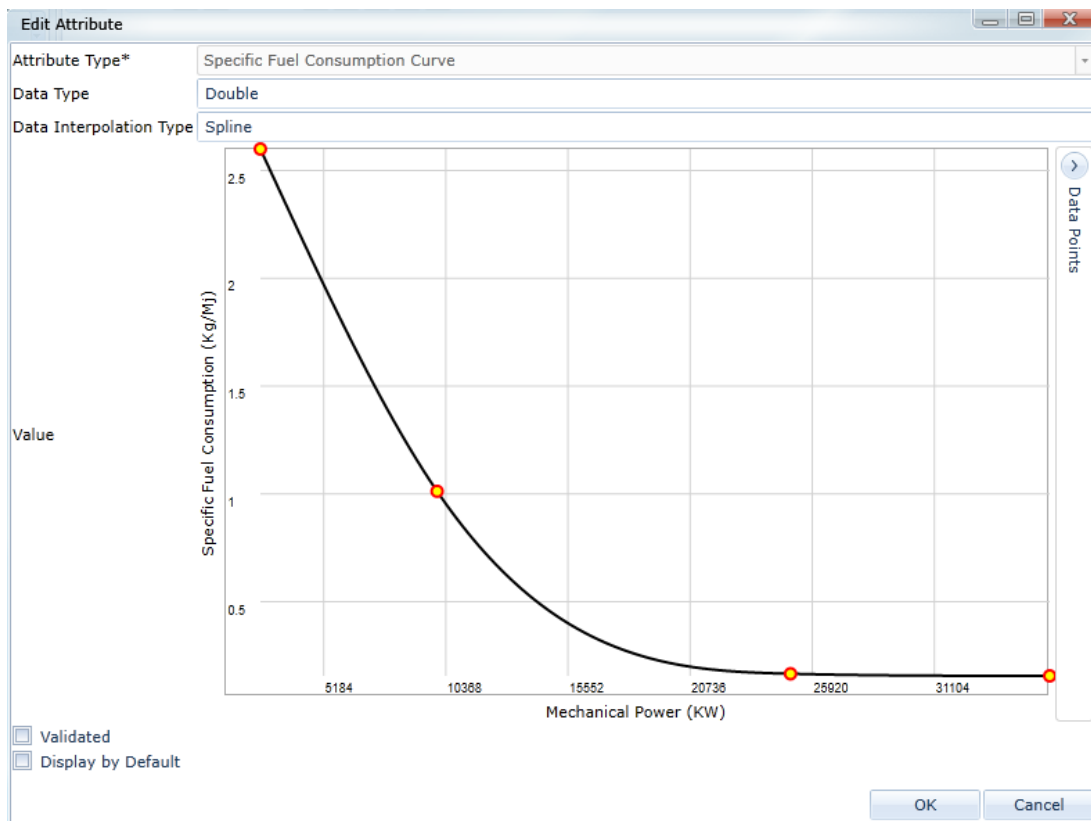


Figure B.3 Spline curve of specific fuel consumption versus mechanical power

APPENDIX C: INTER-DISCIPLINE CONVERGENCY GENERALIZED EXAMPLE

We shall examine a case in where all components get powered, and we have the following dependencies. In C_x^y the subscript x, represents the discipline where the component belongs to, and superscript y represents the discipline the component can potentially affect. The different discipline tools in S3D are currently composed of the following set of components:

$$Electrical = \{ C_e^m, C_e^p, C_e^h \}$$

$$Mechanical = \{ C_m^e, C_m^p, C_m^h \}$$

$$Piping = \{ C_p^e \}$$

$$HVAC = \{ C_h^e \}$$

Therefore, each discipline contains the following dependencies on components from other disciplines.

$$Electrical \rightarrow \{ C_m^e, C_p^e, C_h^e \}$$

$$Mechanical \rightarrow \{ C_e^m \}$$

$$Piping \rightarrow \{ C_e^p, C_m^p \}$$

$$HVAC \rightarrow \{ C_e^h, C_m^h \}$$

The stepping-order could be dynamically determined using the algorithm described in the flow-chart above, but for the purposes of this implementation, the stepping-order solution is constant. In the future, as additional components are added which may cause additional dependencies, this algorithm may be implemented to

dynamically determine the best order. The full steps to determine the current stepping order based on the existing S3D dependencies described above are:

Since the machinery disciplines has the least dependencies we run the electrical discipline first to satisfy them. We mark the machinery set with an asterisk because its dependencies are not actually satisfied by the electrical discipline, since its dependencies were not satisfied and the result is guaranteed to not have converged.

$$E \rightarrow \{C_m^e, C_p^e, C_h^e\}$$

$$M^* \rightarrow \{\emptyset\}$$

$$P^* \rightarrow \{C_m^p\}$$

$$H^* \rightarrow \{C_m^h\}$$

The machinery discipline has no dependencies, so we run the machinery discipline, and restore its dependencies on the electrical discipline, because that dependency was not fully resolved during this iteration.

$$E \rightarrow \{C_p^e, C_h^e\}$$

$$M \rightarrow \{C_e^m\}$$

$$P^{**} \rightarrow \{\emptyset\}$$

$$H^{**} \rightarrow \{\emptyset\}$$

The HVAC and the piping have no dependencies, so we run both disciplines, and restore the dependencies which were not fully resolved at the time they were run.

$$E \rightarrow \{\emptyset\}$$

$$M \rightarrow \{C_e^m\}$$

$$P \rightarrow \{C_e^p, C_m^p\}$$

$$H \rightarrow \{C_e^h, C_m^h\}$$

The Electrical discipline has no dependencies, so we run the electrical discipline.
Its dependencies do not need to be restored because at the time each of its dependencies were run, they were marked as having no dependencies of their own.

$$E \rightarrow \{\emptyset\} : done$$

$$M \rightarrow \{\emptyset\}$$

$$P \rightarrow \{C_m^p\}$$

$$H \rightarrow \{C_m^h\}$$

The machinery discipline has no dependencies, so we run the machinery discipline.

$$E \rightarrow \{\emptyset\} : done$$

$$M \rightarrow \{\emptyset\} : done$$

$$P \rightarrow \{\emptyset\}$$

$$H \rightarrow \{\emptyset\}$$

The HVAC and the piping have no dependencies, so we run both disciplines.

$$E \rightarrow \{\emptyset\} : done$$

$$M \rightarrow \{\emptyset\} : done$$

$$P \rightarrow \{\emptyset\} : done$$

$$H \rightarrow \{\emptyset\} : done$$

* Dependency was satisfied by running a system whose dependencies were not satisfied.