NTNU – Trondheim
Norwegian University of
Science and Technology

Diesel-Electric Generator Load Optimization

Andreas Carlsen

Master of Science in Cybernetics and Robotics
Submission date: May 2014
Supervisor: Tor Engebret Onshus, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This assignment has been written for the company Metso Automation AS with the task of developing a load optimization module for diesel-electric generators in the marine industry. This module will delivered as part of an ongoing project.

The optimization module has been developed in Java and integrated with the Metso DNA system using the engineering tools within this system. The overall safety of the system has been assured by implementing several safety measures.

I would like to thank Metso Automation for providing me with required information/data as well as an office. A special thanks to Lars Svaasand, Stein Østby and Jens-Petter Nyland for their guidance and to Professor Tor Onshus for regular telephone meetings and guidance during the project.

Assignment text

A ship with diesel-electric propulsion has a several marine diesel generators creating power on a high-voltage switchboard. The specific fuel oil consumption (SFOC) of a generator is dependent on the load. In this assignment the system has four generators; two big generators and two smaller generators. The two generator types have a different relationship between SFOC and load. This assignment will address the following points:

- The optimum set point of each diesel generator must be calculated in real time based on the actual load of the switchboard. Additionally, the optimum diesel generator running combination needs to be calculated.
- Safety of the power production must be maintained. Items to consider include:
    1. Any sudden change of electric load occurs will instantaneously be taken by the diesel engine. The power management system will then start to distribute the load according to the control strategy. The power management shall never allow a diesel engine to be loaded below a minimum value or above a maximum value.
    2. Optimization mode should only be available in “normal conditions”. Any deviation like a sudden change in propulsion load shall automatically cause a switch to sea or maneuver mode.

Table of contents

List of figures

Abbreviations

SFOC	-	Specific Fuel Oil Consumption
PMS	-	Power Management System
OPS	-	Operator Station
ACN	-	Application and Control Node
IBC	-	Internal Bus Controller
IO	-	Input/Output
ALS	-	Alarm Station
FBC	-	Field Bus Controller
GUI	-	Graphic User Interface
DG	-	Diesel-electric Generator

Chapter 1

Summary and conclusion

On a ships power plant the diesel-electric generators produce the electricity. Each of the generators has its unique specific fuel oil consumption (SFOC) profile depending on the load. Up until now the generators have been run with balanced load sharing, meaning that all running generators have the same load.

In this thesis an unbalanced load sharing module has been implemented. The module takes advantage of the SFOC profiles, thus optimizing fuel efficiency. This means that each generator can have a different load depending on what is economically beneficial. Since the demand for this optimization is fairly new in the marine industry it proved difficult to find any related work. It was therefore decided to design the optimization algorithm from scratch.

The optimization module was made in Java and implemented into the Metso DNA system via an interface created by the

Metso DNA engineering tools. Optimization was added as a separate mode to the power plant. However, since the optimization mode should have all of the same safety features as another mode, called sea mode, the already built in safety features of this mode was utilized. This was done by building the optimization mode as an add-on to the sea mode in the logic. However, to the user it will appear as two separate modes.

The results of the algorithm were very positive, theoretically saving up to around 200 kg of fuel per hour. Not every situation was as optimal, but the optimized unbalanced load sharing always performed better than or as well as the balanced load sharing. By testing some typical scenarios an average saving of 25,83 kg of fuel per hour (in optimization mode) was found which could mean yearly savings of up to USD 99 434.

Chapter 2

Introduction

On ships the production of electricity is done by a combination of diesel-electric generators (hereafter called generators). The generators produce mechanical energy which is converted to electrical energy that is supplied to the switchboard. Up until now the load has been equally shared amongst the running generators, but now the shipyards are requesting unbalanced load sharing to optimize fuel efficiency.

The SFOC of the generators varies with the load of the generator. Earlier this relationship was quite flat, but with the new engine standards (see Chapter 3) the SFOC curves have become steeper. This leaves room for optimization.

The goal of this thesis has been to minimize fuel consumption by choosing the most optimal load for each generator. The load set points are not set directly by the Metso DNA system, instead it gives out increase/decrease commands to a regulator.

One very important thing to consider when doing an optimization like this is safety. Until now the ship's power plant has had three different running modes in Metso DNA; Harbor, Sea or Maneuver mode. The optimization will be added as a separate mode, leaving four modes in total. Harbor mode is used when the ship is at harbor. Maneuver mode is generally used when the ship is maneuvering, causing the load on the engines to change rapidly. Sea mode is normally used for steady sailing and the optimization mode will also generally be used in this situation. The only thing that separates Sea mode and Optimization mode is that the Optimization mode will have unbalanced load sharing. Since the Sea mode already has built in safety mechanisms this points towards the possibility of utilizing these mechanisms in the optimization mode as well.

The safety mechanisms in Metso DNA mainly consist of making sure there are enough running generators to cover the power demand. Shutdowns/start blocks of generators are handled by the generators own control system. Metso DNA's task is to start up a new generator if a shutdown/start block occurs.

Figure 1 shows the basic overview of the system. The interfaces from the external data and the Metso DNA system

have not handled in detail in this thesis. The focus of the thesis has been on the “Load optimization” and “Safety functions” box. Only a basic overview of Metso DNA will be given.

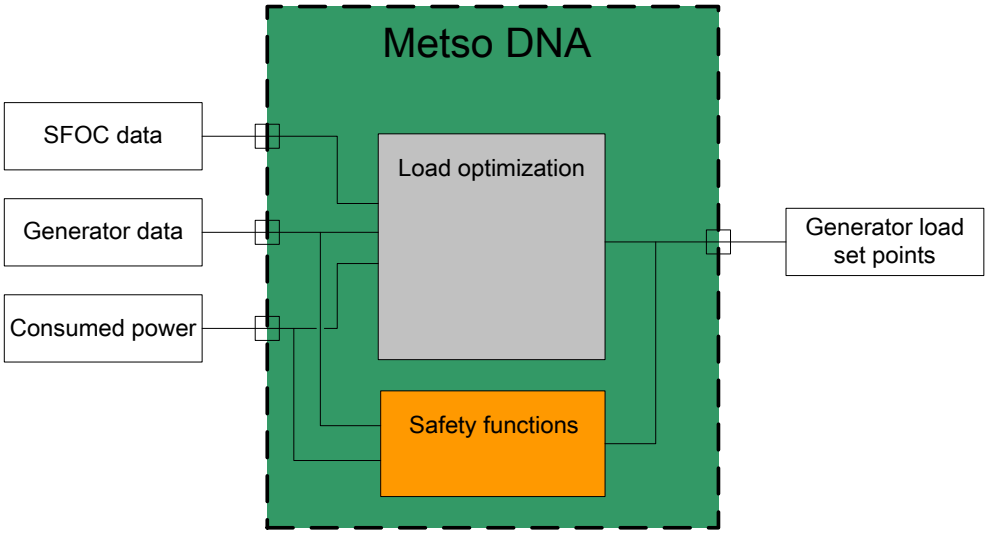

Figure 1 Basic overview of the system

Another consideration is run time as the optimization program has been given a time slot of 200 ms in every cycle of the Metso DNA system. This means that the optimization program needs to read the input, calculate the set points and set the output to the correct values in less than 200 ms.

Chapter 3

Background

In the past the generators have been run with balanced load distribution. That is every generator has the same load. The SFOC profile of the generators was much flatter than it is today and therefore there was not very much to gain by optimizing the load distribution. However, due to the Tier II and Tier III emission demands [1], the engine manufacturers have done modifications to their engines causing SFOC profiles to change as well [2]. Now the profile is steeper and there is more to gain by optimizing the load distribution.

The customer in this project is currently building a new ship. Metso Automation is delivering several systems for this ship, one being the power management system (PMS). The PMS includes load dependent start/stop of the engines as well as the safety mechanisms that start up a new engine if e.g. a shutdown occurs.

For this project the customer has requested optimization of the load distribution as part of the PMS. The scope of this thesis has been the design of this optimization module.

Chapter 4

Theory

In this chapter the theoretical aspects of the optimization is explained.

4.1 Mathematical model

To get an overview of the problem at hand a mathematical model was created. The model represents the optimization problem with the constraints. The problem can be stated as

$$\text{Minimize} \sum_{i=1}^{4} y_i(x_i) \frac{x_i}{100} w_i \alpha_i$$

Equation 1 Minimization problem

Given that

$$\sum_{i=1}^{4} \frac{x_i}{100} w_i \alpha_i \geq D$$

$$\sum_{i=1}^{4} \frac{U_i}{100} w_i \alpha_i \geq D + S$$

$$x_i \geq L_i$$

$$x_i \leq U_i$$

$$x, y, L, U, w, D, S \geq 0$$

$$\alpha = \{0,1\}$$

Equation 2 Constraints

Where the variables are defined as the following:

w_i = Maximum effect of generator i

y_i = Vector containing SFOC vs load of generator i

x_i = Load of generator i (in percent)

L_i = Lower load limit for generator i

U_i = Upper load limit for generator i

α_i = Binary variable that described whether or not a generator is running

D = Demand on the board

S = Safety band

4.2 SFOC curves

The SFOC curves describe the relationship between the SFOC and the load of a specific engine. The specific oil consumption is given as the consumption of fuel, in grams, per kWh (g/kWh).

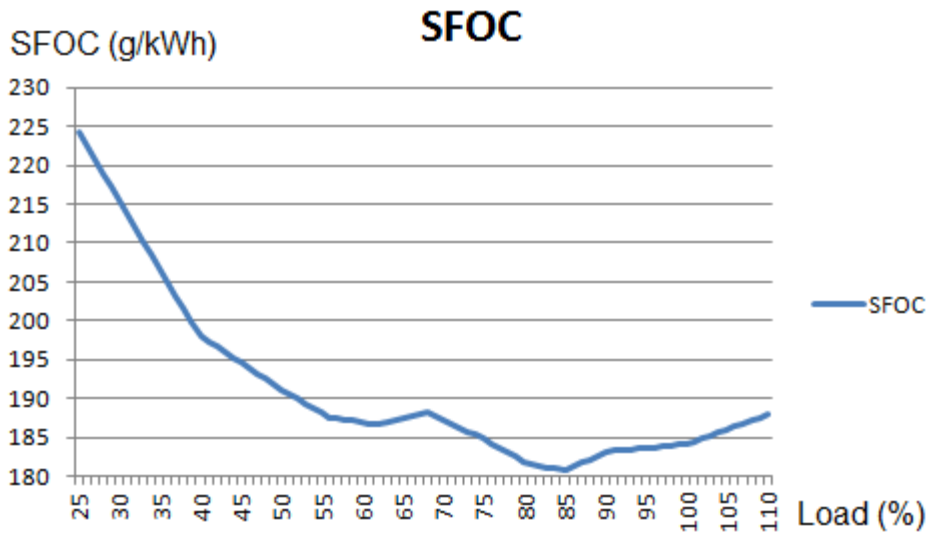

Figure 2 Example of an SFOC curve

An example of an SFOC curve is given in Figure 2. As we can see by this curve the optimal load for this generator is 85%. At this point the generator will use the least amount of fuel per kWh.

Of course, if we only have one generator running there is no room for optimization, but if we have several running generators we might be able to save fuel by giving the generators different load set points. E.g. it is quite obvious from Figure 2 that if we have two generators of equal size that shares the same SFOC profile it will be beneficial have the generators running at 85% and 62% instead of both running at 73.5%.

The initial SFOC curves are created from confidential FAT data, however the SFOC curve for a specific generator will change during its lifetime. It is therefore necessary to update the SFOC curves on a regular basis. If the curves are not updated the optimization algorithm may not find the optimal set points. The curves are updated according to the strategy explained in chapter 6.2.

4.3 Optimization algorithm

The only time restriction is that the algorithm must finish in its assigned time frame of 200 ms. As this is a quite long time frame a simple algorithm was constructed. If the simple algorithm performed within the timeframe there would not be any need for any further modification of the algorithm itself. First of all the resolution of the algorithm also needed to be decided.

4.3.1 The resolution of the algorithm

Since the resolution of the input (SFOC curves) is very low there is no point in having a very high resolution on the output. It was therefore decided that a resolution of 1% would be acceptable. The algorithm should then give set points to the n-1 running generators and the last generator would take the remaining load. The algorithm will still take the load of the remaining engine into account when deciding the optimal load set points.

4.3.2 Simple algorithm

The first and simplest algorithm that was tested simply checked every load combination of the running generators within the low and high limits. The most optimal load combination was then given as an output.

A separate algorithm for each number of running generators was made. If two generators was running then the algorithm consisted of two for loops, if three generators was running then the algorithm consisted of three for loops and so on. The pseudo code for the main program is presented in Figure 3.

```
procedure calculateOptimalLoad
if 1 running generator then
      run OptimalLoad1Gen(active_gens);
else if 2 running generator then
      run OptimalLoad2Gen(active_gens);
else if 3 running generator then
      run OptimalLoad3Gen(active_gens);
else if 4 running generator then
      run OptimalLoad4Gen(active_gens);
```

Figure 3 Pseudo code for the calculateOptimalLoad method of the simple algorithm

After the program determined the number of running generators the associated algorithm would be run. The pseudo code for the algorithm given four generators is given in Figure 4.

```
procedure OptimalLoad4Gens
for i:=gen1 low limit to i:=gen1 high limit do
 for j:=gen2 low limit to j:=gen2 high limit do
  for k:=gen3 low limit to k:=gen3 high limit do
   for l:=gen4 low limit to l:=gen4 high limit do
    set temp_load to (i,j,k,l);
    if temp_load gives necessary production then
     set temp_OC to calculated oil consumption;
     if temp_OC is better than the best oil consumption so far then
      set best_OC to temp_OC;
```

Figure 4 Pseudo code for the OptimalLoad4Gens method of the simple algorithm

The simple algorithm did not perform within the 200 ms time frame and therefore modifications were needed.

The complete code for the simple algorithm can be found in Appendix A.

4.3.3 Modified algorithm

In the modified algorithm the running generator with the lowest priority is not checked in every point, rather it is just set to take the remaining load. The program was also compressed so that it did not have a separate method for each number of running generators.

```
procedure calculateOptimalLoad
for i:=(gen1 low limit)*(gen1 bin) to i:=(gen1 high limit))*(gen1 bin) do
 for j:=(gen2 low limit)*(gen2 bin) to j:=(gen2 high limit))*(gen2 bin) do
  for k:=(gen3 low limit)*(gen3 bin) to k:=(gen3 high limit))*(gen3 bin) do
   if 4 active generators then
    set temp_load to (i,j,k,remaining load);
   if 3 active generators then
    set temp_load to (i,j,remaining load,0);
    if remaining load is within limits then
     set k to gen3 high limit;
   if 2 active generators then
    set temp_load to (i,remaining load,0,0);
    if remaining load is within limits then
     set j to gen2 high limit;
   if 1 active generators then
    set temp_load to (remaining load,0,0,0);
    if remaining load is within limits then
     set i to gen1 high limit;
   if remaining load was within limits then
    set temp_OC to calculated oil consumption;
    if temp_OC is better than the best oil consumption so far then
     set best_OC to temp_OC;
```

Figure 5 Pseudo code for the calculateOptimalLoad method of the modified algorithm

The bin variables in the algorithm in Figure 5 are 1 if the generator is running and 0 if the generator is not running. The algorithm is still quite simple, however it is less time consuming than the simple algorithm and performed within the 200 ms time frame.

The program is in a form that the Metso system can handle. If four generators is running this program cannot set the set points of all four generators. It will set the set point of three of the generators and the last one will take the remaining load.

The complete code for the modified algorithm can be found in Appendix B.

Chapter 5

Metso DNA

In this chapter we will take a closer look on how the Metso DNA system works. As much of this is confidential we will only be scratching the surface. However, since a large part of this assignment is interfacing the Metso DNA system, we need to get a basic understanding of how this is done.

5.1 Basic overview

Figure 6 gives a basic overview of the Metso DNA System. The input signals are processed on card level in the IO rack. Analog signals get a digital value and a fault bit while binary signals get a digital value, a timestamp and a fault bit. The signals are then collected by the internal bus controller (IBC) and sorted into tables of data that the IBC updates.

The application and control node (ACN) is a computer where the IO is processed in different Metso modules. The ACN has a

field bus controller (FBC) that communicates with the IBC collecting the signals. The signals are then processed in the ACN. This processing results in monitoring signals, alarm signals or control signals. The FBC have a defined sample rate and the shortest sample rate used on the PMS is 200 ms. In this assignment the goal was to stay within this time frame as this is the sample rate generally used for control functions.

On the other end of the system there is an operator station (OPS) which will gather signals from the ACN and display the values in a user interface. If the user modifies a value the OPS will update the value in the ACN, which in turn will send the value to the IBC which will update the given outputs on the IO rack.

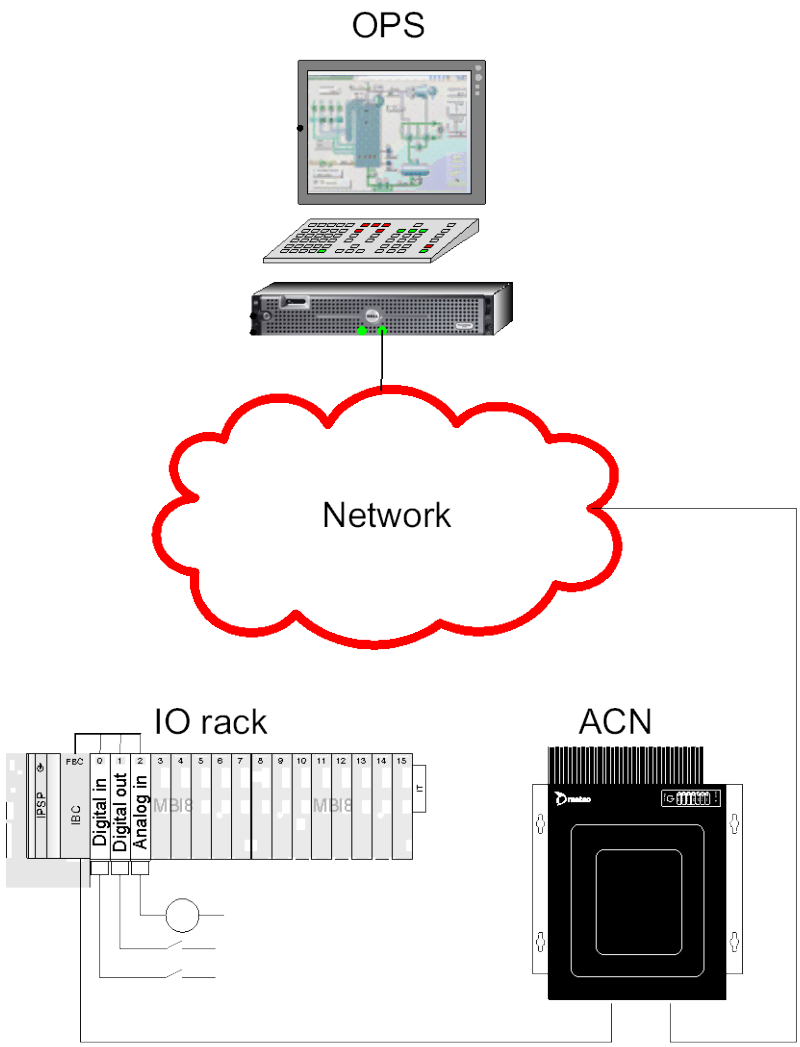

Figure 6 Basic set up of the Metso DNA system

The ACN and the OPS gathers the signals from the network ring. This network ring will look something like Figure 7 using a token passing Ethernet protocol[3]. As we can see from Figure 7 other stations can also be connected to the network, like the Alarm station (ALS) shown in the figure. All components are connected to the network via switches which are not shown in the figure.

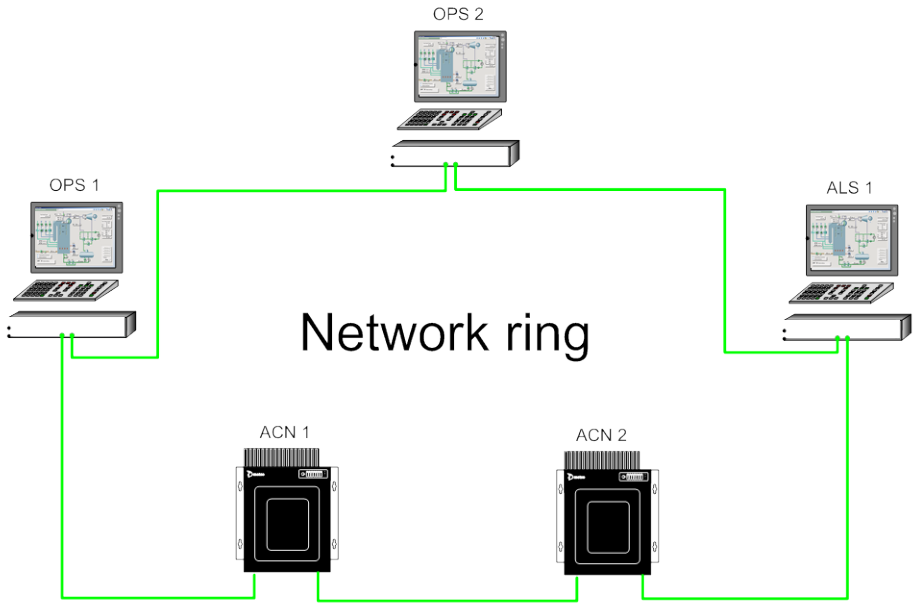

Figure 7 Metso DNA network

5.1.1 Redundancy

Metso DNA uses redundancy in every layer. This means that the set up in Figure 6 will not be satisfactory for the system. The set up must therefore be modified according to Figure 8.

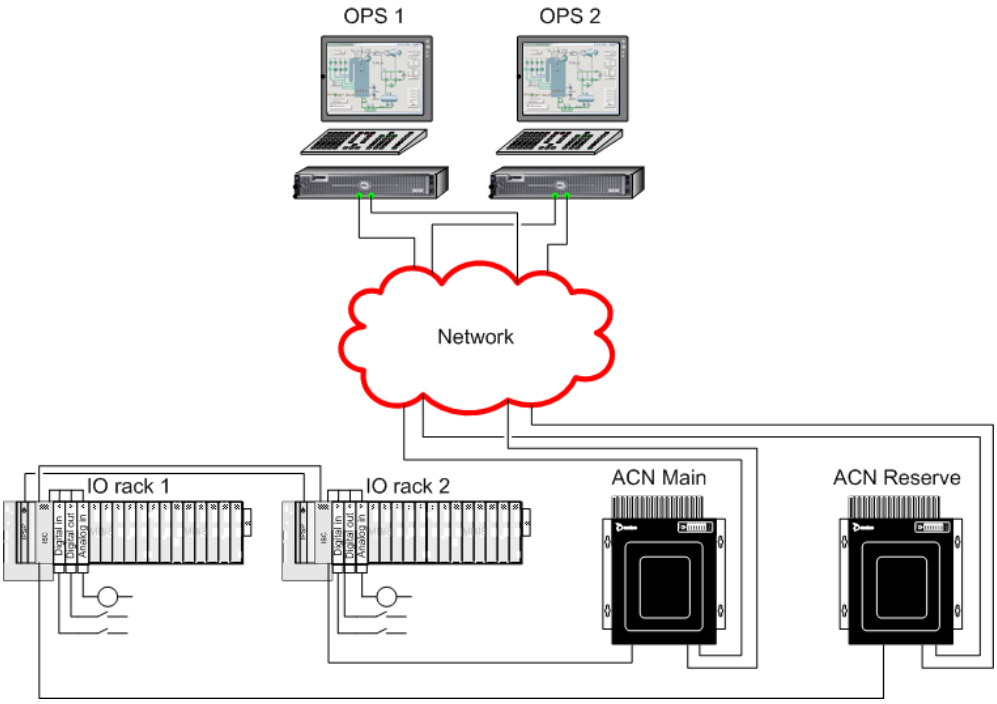

Figure 8 Redundancy in the Metso DNA system

ACN Main and ACN Reserve need to run redundant modules so that the system can handle the loss of an ACN and still remain functional. Each ACN have two different connections

to the network, each going to a separate switch. This set up allows any module or connection to fail and still remain functional. If two of the same component fails the system will no longer function.

5.2 FbCad

In the Metso DNA system modules are made using function block programming. The programming environment is called FbCad. The function block diagrams created in FbCad control loops related to controlling and monitoring the process controlled by Metso DNA [4].

Figure 9 shows an example of a module created in FbCad. This example has one input from an analog input card (1), two digital inputs from another module (2) and one output to a digital output card (4). The function of the module (3) is that it sets the output high if, and only if, the analog input is greater than a constant value and one of the digital inputs is high. The number on each of the function block is the execution order of the block. In this case the module will first OR the two digital inputs, then compare the analog input to a constant (42,0) and finally AND the result of the two previous blocks.

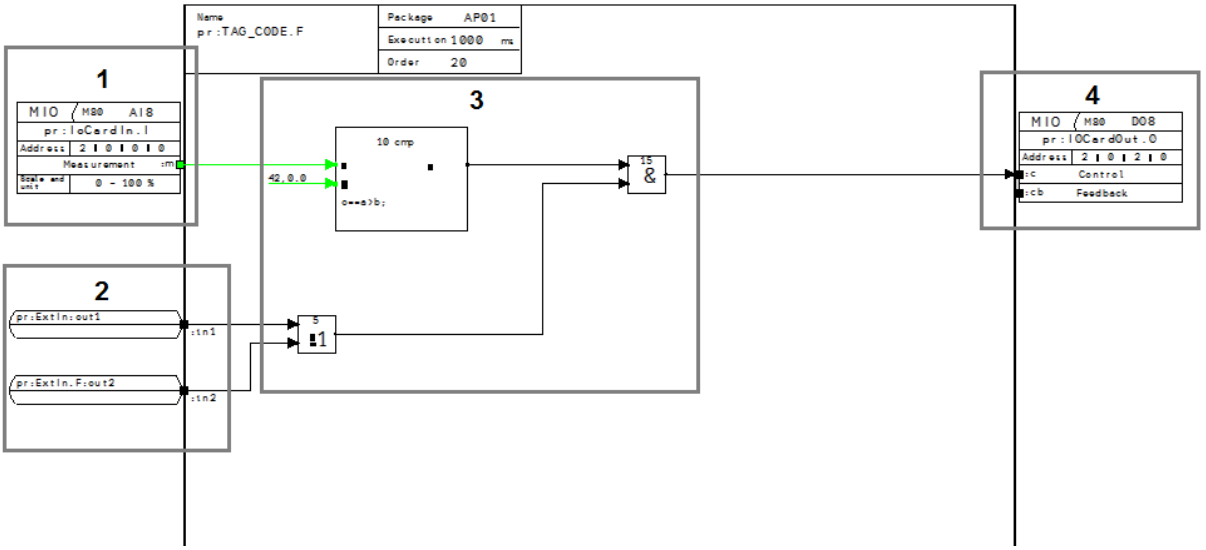

Figure 9 Basic example of a module made in FbCad

This gives us a basic idea of how FbCad works. Further information about FbCad can be found in [4] and a complete list of function blocks can be found in [5].

5.3 Java prog2 block

In FbCad there is also an option to create Java code which can be integrated into a function block. This increases the flexibility of the system allowing the engineers to develop more complex programs in Java.

5.3.1 Enabling Java on Metso DNA

First of all Java must be enabled in the Metso DNA system. A guide to doing this is given in [6]. Java is not automatically set up after installation of Metso DNA so this procedure must be done manually on each process station running Java blocks.

5.3.2 Adding a new prog2 block

When FbCad is open one can add a new prog2 block by simply pressing “Fblocks3” and choosing “prog…”. A new dialog box will appear and in this dialog box choose the prog2 block [7].

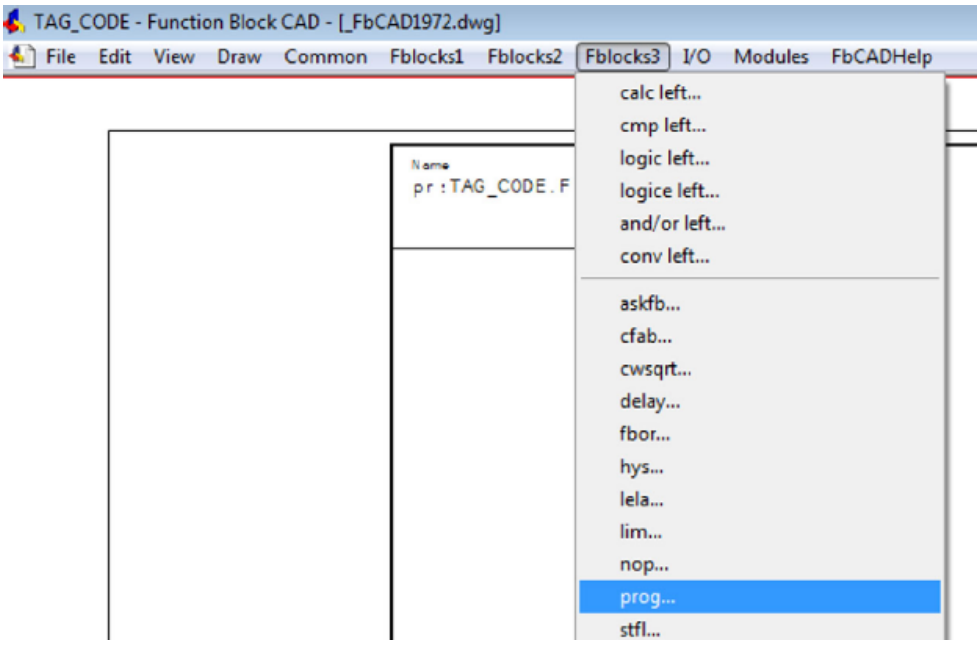

Figure 10 Adding a new prog2 block

After adding the prog2 block to the desired location in the diagram the dialog in Figure 11 will appear. In this dialog the engineer should set the execution order of the prog2 block, the program name and the name of the java program class whose methods are to be called from the block. Inputs and outputs can be added by typing the variable name into the "Tag" field, choosing input (direction left) or output (direction right) and setting the appropriate data type in the "DNAType" field. A complete list of supported data types can be found in [8].

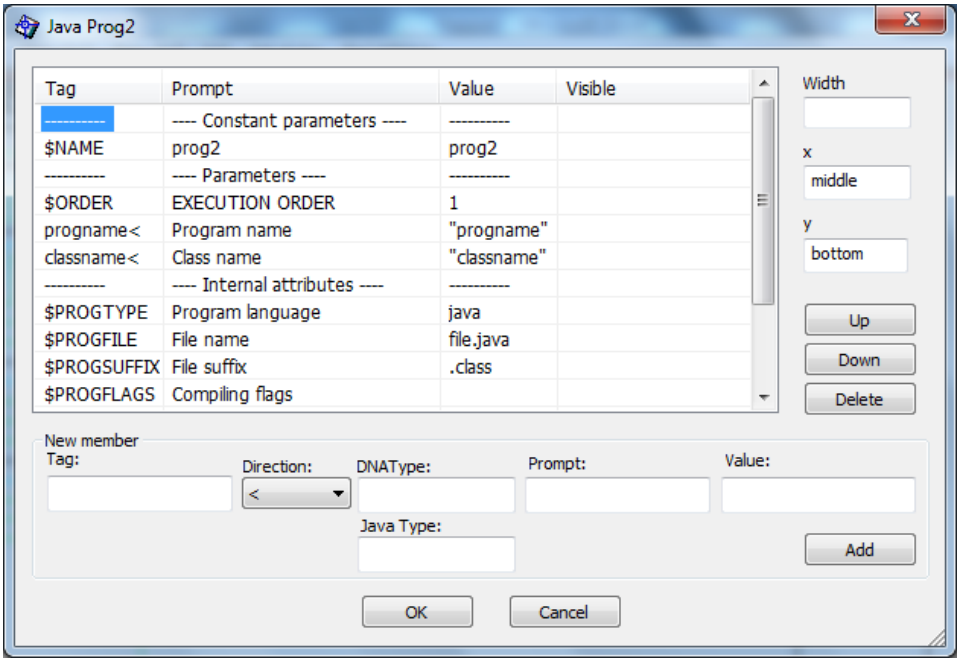

Figure 11 prog2 dialog

5.3.3 Java skeleton

Once the prog2 block has been added in FbCad it is possible to create the Java program skeleton that contains the function that the prog2-block calls upon on execution.

Creating this skeleton is done by entering the command “genjava” in the CAD command prompt, selecting the prog2 symbol using the mouse and entering the name of the generated source file. The file name should be in the format

<classname>.java. The generated file will be located in the folder *dna**Data**EA**fbmod* [9].

Figure 12 shows the functions included in the Java skeleton. unbalLoadOpt() is the constructor. Here the member variables are connected to the input/output of the prog2 block. checkInitialTableConnections() checks that the parameter tables are properly connected, while checkTableConnections() checks the integrity of the parameter tables. init() is called once for each function block, when it is loaded into the application, and can be used for initialization code. run() is called when the execution block is executed and it in turn calls upon the runMe() method which is where the program logic should be located. moduleUpdate() is called upon if the engineer updated the function block diagram on the application server. Finally the exit() method is called upon if the method is unloaded from the application server and should contain clean up code. The complete generated file can be found in Appendix C.

```
public <classname>() throws AITableReferenceException
private final void checkInitialTableConnections() throws AITableReferenceException
private final void checkTableConnections() throws AITableReferenceException
public void init()
public void run()
private void runMe()
public void moduleupdate ()
public void exit()
```

Figure 12 Functions included in the Java skeleton

5.4 Updating the SFOC curves

In a project like this there are many different companies delivering different subsystems for the ship. In this specific project another company is responsible for updating the SFOC data. They are gathering the necessary information in real time and update the SFOC data accordingly.

It has been agreed that this company will send updated SFOC data via serial line to the Metso DNA system. 12 points will be sent for each generator. It will then be in the scope of my program to generate the rest of the points by a simple linearization between each of the given points.

5.5 Metso DNA simulator

Since the installation of the equipment onboard the ship will be after the delivery of this thesis there has been no opportunity to test the optimization module on a real environment. However Metso has developed a simulator that has proven to be very well suited for testing the PMS.

In this thesis Metso DNA C2013 PMS simulator was used and the results referred to in this thesis have been created using this simulated environment. Figure 13 shows the user interface of this simulator. The pink fields are functions that were not in use during testing.

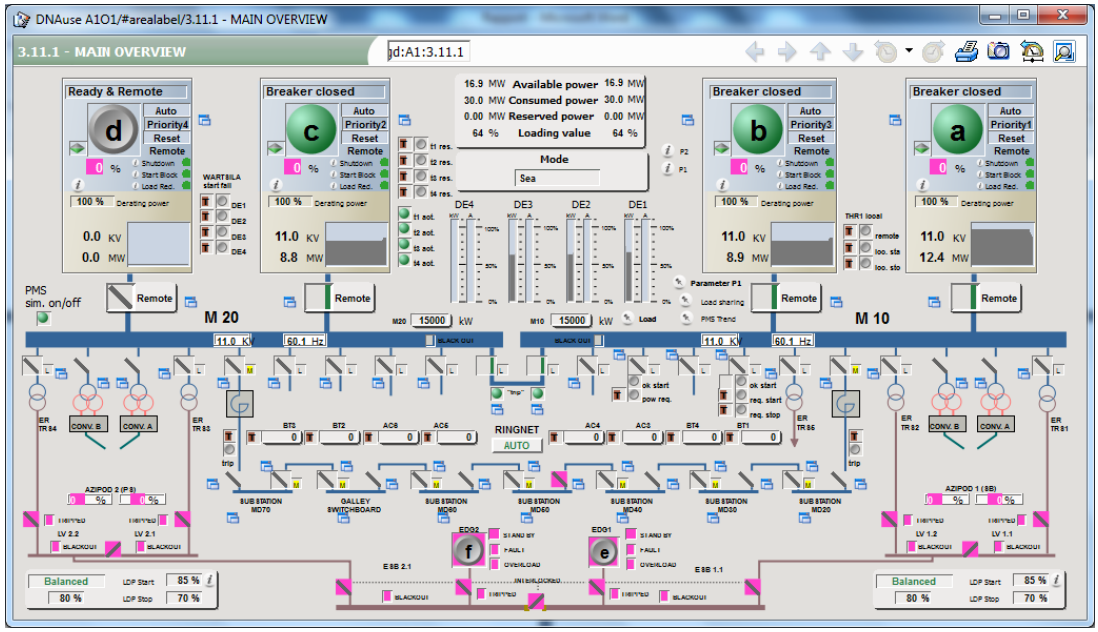

Figure 13 Metso DNA simulator user interface

Chapter 6

Method and material

6.1 Material

In this assignment the following software was used:

- Eclipse IDE for Java developers, version: Juno service release 1
- Java, version: 1.6
- Metso DNA C2013

6.2 SFOC curves

The SFOC curves are generated by a set of points. The curves are initialized with 12 points and then updated after some time. A point is given as x = load and y = SFOC. The x-values of the points are always the same, but the y-values will change over time. The curves are generated by using a simple linearization between the points (see Appendix D).

The 12 points that are used to initialize the SFOC curves are taken from the SFOC data of the engines. This data is confidential and thus not included in this report.

With the system running the points will be updated every month. The new points are sent, by serial line, from an external system to the Metso system. At the receiving end of the data we use a standard Metso serial line interface. The 12 points are then given as input to the optimization program together with a bit that tells the program that new SFOC data is available. The interface between the optimization program and the Metso system is explained in chapter 6.5.

6.3 Optimization algorithm

The optimization algorithm requires input from the Metso DNA system. For each generator the program require a binary variable telling the program whether or not the generator is running, high and low limits, priority and nominal power. The program also needs to know the consumed power/power demand.

The time consumption of the algorithm was tested by giving the algorithm different input and printing the run time. This showed a worst case scenario with a run time of 90 ms. This is well within the time slot of 200 ms so it should perform good enough to be integrated with Metso DNA. If the run time of the program increases beyond 200 ms when the program is integrated into Metso DNA we would get an error message in the error log of Metso DNA. This would acquire further modification of the algorithm.

6.4 A simple simulator

Before making an interface to the Metso system a simple simulator was created to check if the optimization algorithm was working. This simulator was also useful for checking how much fuel was saved by using the optimization algorithm.

6.4.1 The user interface

The user interface of the simple simulator was created with Swing (java) and is very straight forward. The left column shows the load distribution and fuel consumption using the optimization algorithm, while the left column shows the load distribution and fuel consumption using balanced load sharing. The user gives the input (the power on the board) by adjusting a slider. This GUI is very useful for checking the potential savings in fuel consumption. The complete code for the simple simulator can be found in Appendix E.

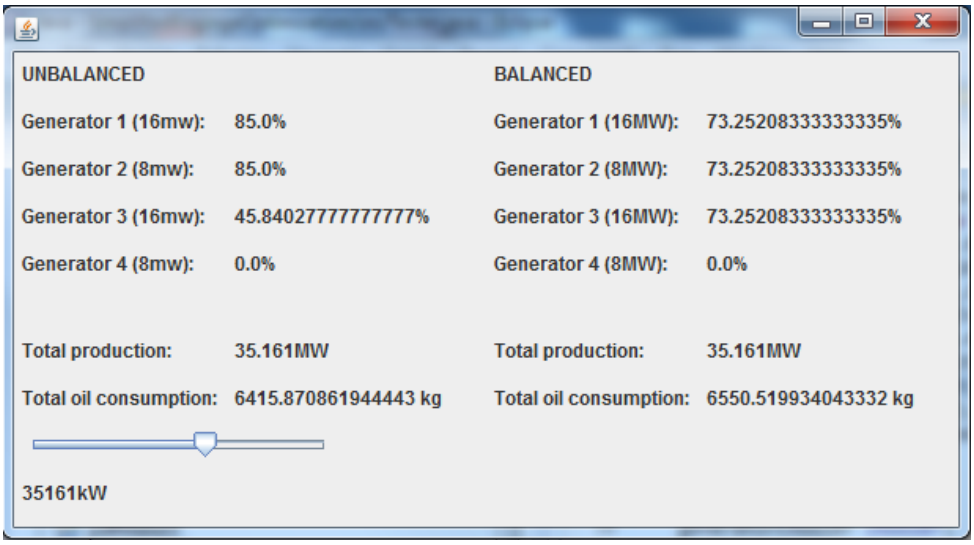

Figure 14 User interface of the simple simulator

6.4.2 How it works

In the “Unbalanced” column of the GUI the load set points are set according to the optimization algorithm as explained in chapter 4.3.3. In the “Balanced” column every generator takes the same set point according to the code in Appendix F.

In this simple simulator a simplified algorithm for starting/stopping generators was made. This algorithm starts a new generator when all of the running generators reach their high limit and stops a generator when it is possible to run one generator less without reaching the high limit (Java code can

be found in Appendix G). The limits can be adjusted directly in the code so that it is possible to test different scenarios.

The fuel consumption is calculated by the following formula:

$$\sum_{i=1}^{4} y_i(x_i)\frac{x_i}{100}w_i$$

Equation 3 Fuel consumption

6.5 Interfacing the Metso DNA system

To interface Metso DNA a new FbCad module was created. A prog2 block was then added to the module according to chapter 5.3.1 and 5.3.2 (See Figure 15).

The interface between the optimization program and Metso DNA is the Java skeleton explained in chapter 5.3.3. This means that the optimization program logic was integrated in the Java skeleton. The optimization algorithm was placed in the runMe() method while initialization code and clean up code was placed in the init() and exit() methods respectively. The complete code can be found in Appendix H.

After incorporating the optimization program logic in the skeleton file a .jar-file was generated using the built in compiler in Eclipse. The .jar-file should be located at *dna**CA**pcs**program*. After this was done the system files were edited according to [10] so that the Metso DNA system knows where the files are located.

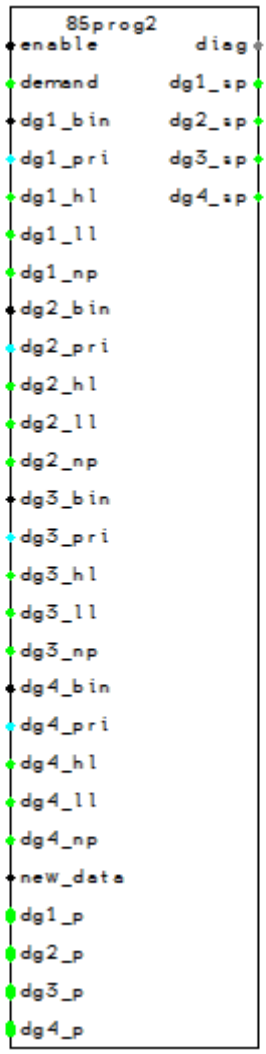

Figure 15 Optimization prog2 block

6.6 Implementing the module in Metso DNA

The prog2 block was now ready for use. It will function as a standalone application with no other logic included in the same function block diagram. The function block was connected to its respective inputs and outputs as seen in Figure 16.

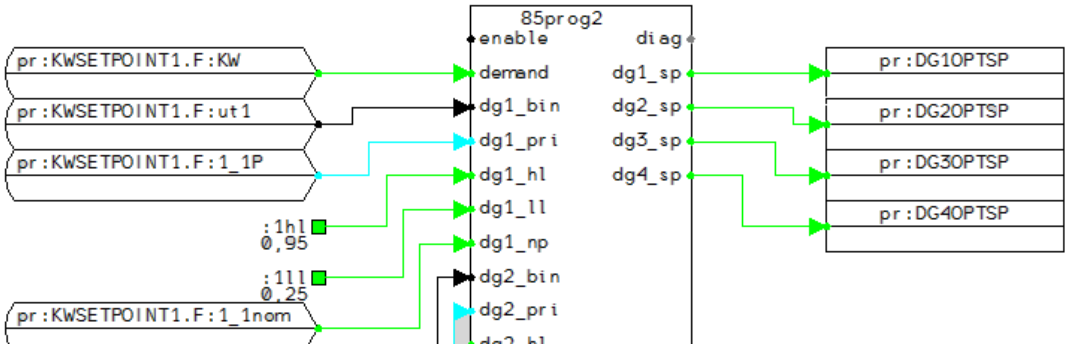

Figure 16 Section of the connections for the prog2 block

6.6.1 Inputs and outputs

The inputs come from confidential program logic so the origin of these signals will not be shown in this thesis. However, as Figure 16 shows, the inputs are the power demand as well as the different parameters for the generators (binary

running/not running, high limit, low limit and nominal power). Also there are inputs for updating the SFOC curves shown as “dg1_p”, “dg2_p”, “dg3_p” and “dg4_p” as well as the “new data” binary input in Figure 15.

The outputs are written to a standalone module that only contains these data. This enables the engineer to easily read this value in another module.

6.6.2 Implementing the module as a separate mode

Since the unbalanced load optimization was to be implemented as a separate mode the mode selection logic in Metso DNA had to be modified. Figure 17 shows the old mode selection logic located in the MODECONFIG module. The “MODE” variable is an *int* representing the selected mode. The compare block sets the respective mode active in the logic.

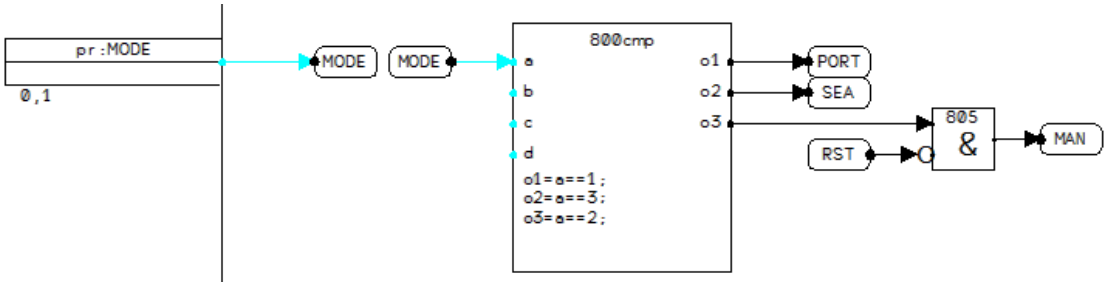

Figure 17 Old mode selection logic

The new mode was added to this logic as mode number 4 according to Figure 18. The compare block was modified so that it included the “OPT” mode and also it will set the “SEA” mode active when “MODE” is set to 4 (optimization mode).

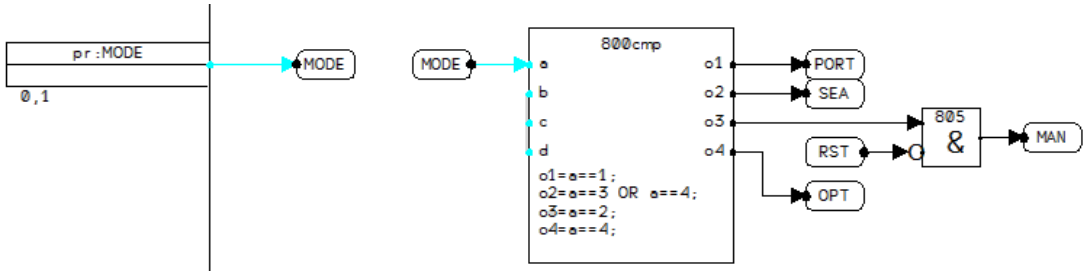

Figure 18 New mode selection logic

The reason why sea mode is set active when optimization mode is active is that sea mode contains all of the necessary

safety functions needed in optimization mode. The only thing that should separate these two modes is the set points given to the generators under normal conditions.

The optimizing mode should only be available when the board is closed. That is when the breakers marked in Figure 19 are closed. This means that an automatic jump to sea mode should occur if one or both breakers are open.

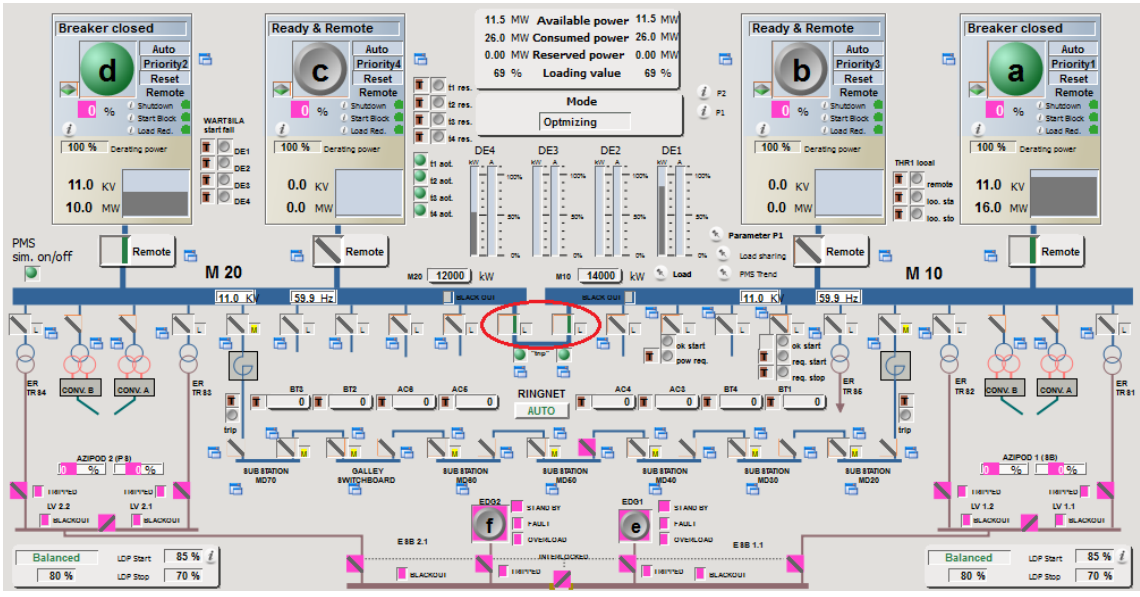

Figure 19 Electrical board breakers

The automatic mode switch was implemented as shown in Figure 20. The first OR block says that if any of the switches

are open (set to 0) the output of the block is set to 1. Next this signal goes into an AND block together with the OPT bit. If any of the switches are open and the OPT bit is set to 1 the output of this block will be 1, otherwise 0. This signal then goes into a conditional block (ccos) which copies the constant value 3 (sea mode) into the MODE variable if the conditional signal is 1.

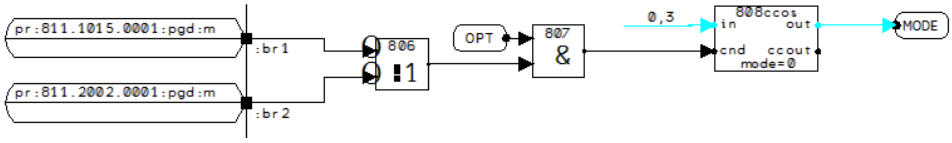

Figure 20 Automatic mode switch

6.6.3 Maintaining safety

Since the optimization mode has been built as an add-on to the sea mode it maintains all of the same safety functions that exist in sea mode. As mentioned earlier the only difference between sea mode and optimization mode is the unbalanced load sharing. This eliminates the need for changing the mode automatically when a fault (other than open breakers) occurs. It was therefore decided by Metso, in cooperation with the customer, that there should not be an automatic mode change in the case of a fault.

6.6.4 Sending the set points to the generators

The logic that actually creates the signals to the generators is confidential. However, the logic that decides whether or not the set points should be set by the optimization module can be shown. All of the logic in this chapter is located in the KWSETPOINT module.

First of all a bit called biX, where X is the number of the generator, is set (Figure 21). biX is 1 if the generator is running (bX) and is not unloading (uX). Unloading means that the generator is preparing to shut down and the load should be distributed amongst the other running generators.

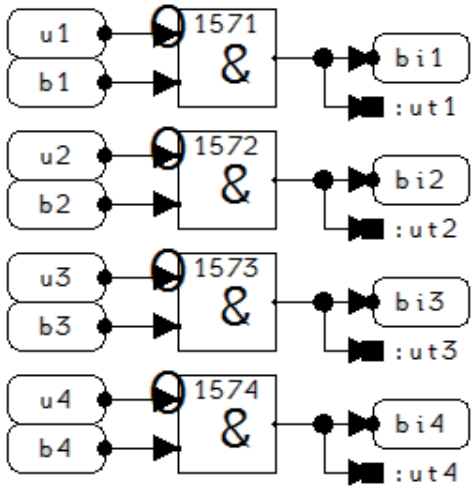

Figure 21 Running/not running logic

Next the lowest priority of the running generators is found (Figure 22). The diss blocks copies the priority of the generator (dXp) if the generator is running (biX is 1) and a constant with the value 0 if the generator isn't running (biX is0). The dvss block copies the largest input to the output. The result is that the ldg variable contains the lowest priority of the running generators. It is important to note that a high priority means a low value of the dXp variable while a low priority means a high value. 1 is the highest possible priority, while 4 is the lowest.

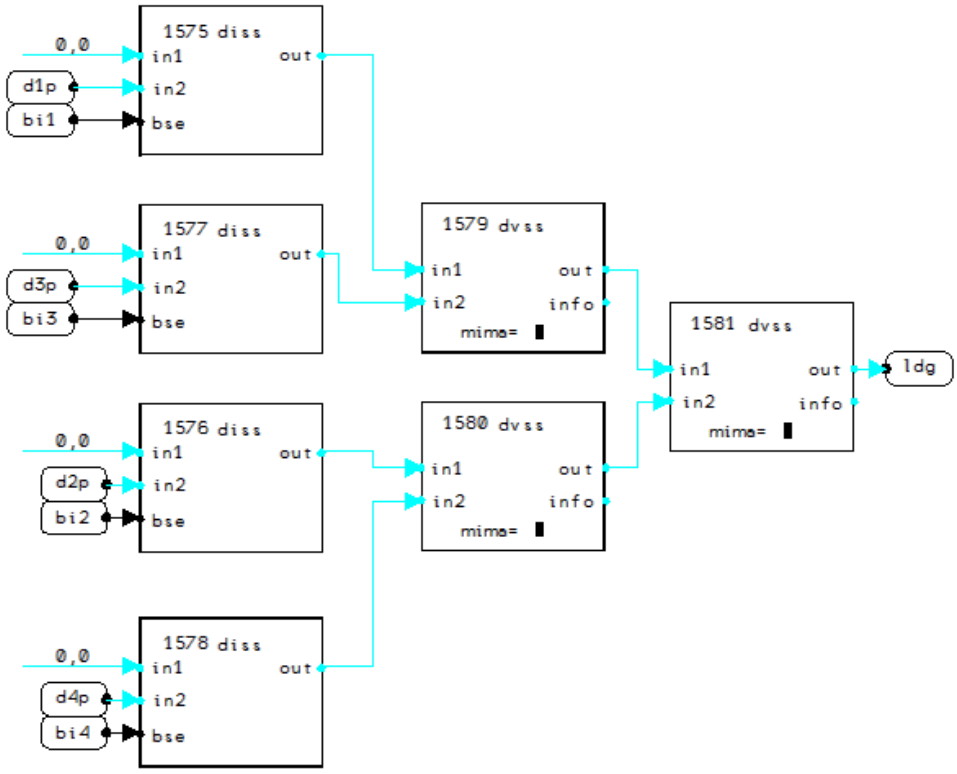

Figure 22 Finding the lowest priority of the running generators

Figure 23 shows the logic stating whether or not generator X should get its set point from the optimization logic or not. dXu is set to 1 if the ldg variable is greater than the priority of the generator (dXp) and the generator is running. This is because the running generator with the lowest priority should not be set directly by the optimization module; rather it should just take the remaining load.

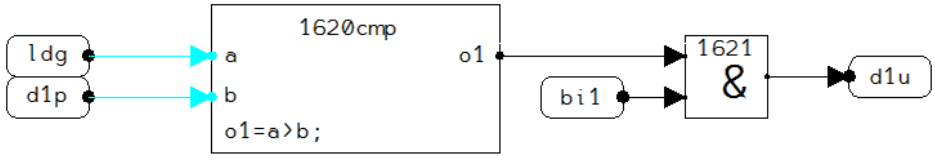

Figure 23 Priority logic

Next the optimization mode bit is read into the local variable optm (Figure 24) and finally the logic deciding whether or not a generator should take its set point from the optimization module is presented in Figure 25. If the program is in optimization mode (optm=1) and the dXu bit is 1 the generator should get its set point from the optimization program. The logic in the black box is then executed taking the set point from the optimization module. If the result of the

AND block is zero the cng 2 block will create a jump forward to the “1745 label” bypassing any optimization logic.

Figure 24 Reading the optimization mode bit

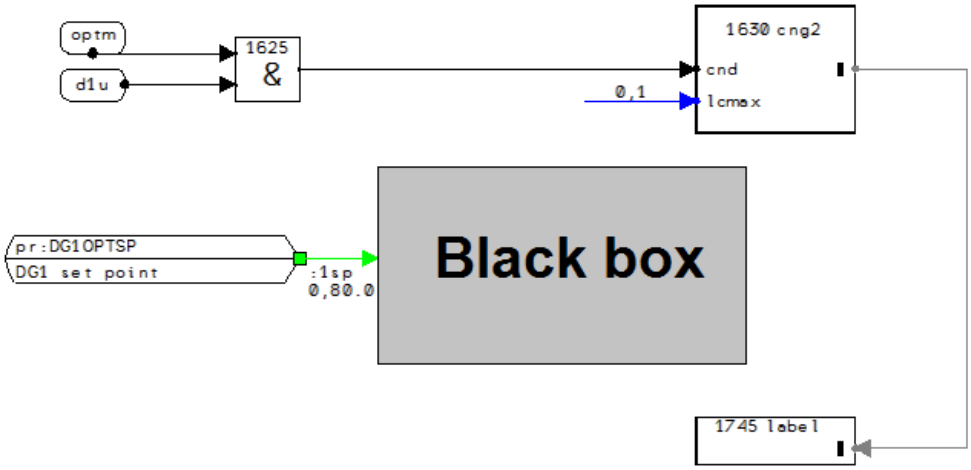

Figure 25 Bypass optimization or not

Chapter 7

Results

Since this system has only been tested in a simulated environment the results are theoretical. We will not know the true results until we test the system on a real ship. However, the simulated environment is well suited for giving us an idea on how well the system will perform.

7.1 Implementation

The implementation of the algorithm was very successful. The Metso system is a versatile system, easy to work with, and the implementation of Java programs was straight forward as well. As shown in Figure 26, unbalanced set point were successfully set by the program and transferred to the Metso System. Figure 27 shows that the set points are also correct according to the simple simulator designed to test the algorithm.

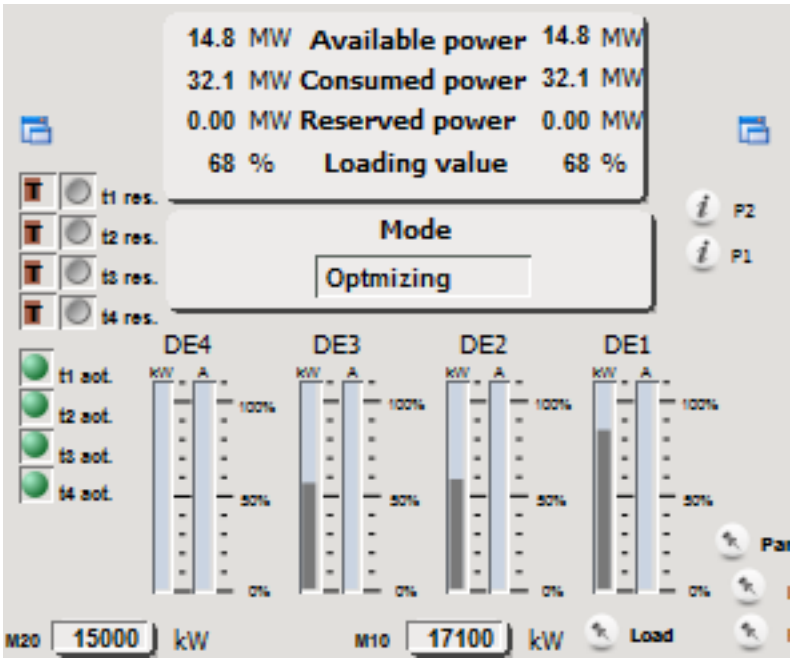

Figure 26 Optimization module running on Metso DNA

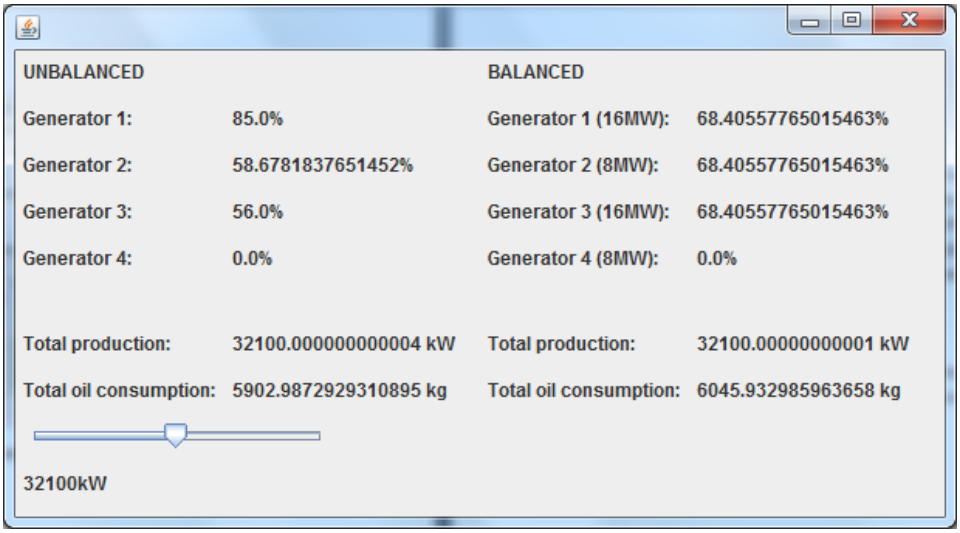

Figure 27 Simple simulator showing the same set points as in Metso DNA

No error messages were recorded in the Metso DNA error logs during testing. This means that the algorithm performed within its time slot in every test scenario. The program was left running overnight on several occasions to make sure that the program was performing consequently without errors.

7.2 Savings

The simple simulator was used to check the potential savings of the optimized unbalanced system compared to the balanced system. Potential saving of up to around 200 kg of fuel per hour, when four generators are running at an average load of 67,85% (see Figure 28), is a very good result. Not every situation showed this level of savings (see Figure 29), but for every situation tested the optimization gave results that was better than or as good as if balanced load sharing had been used.

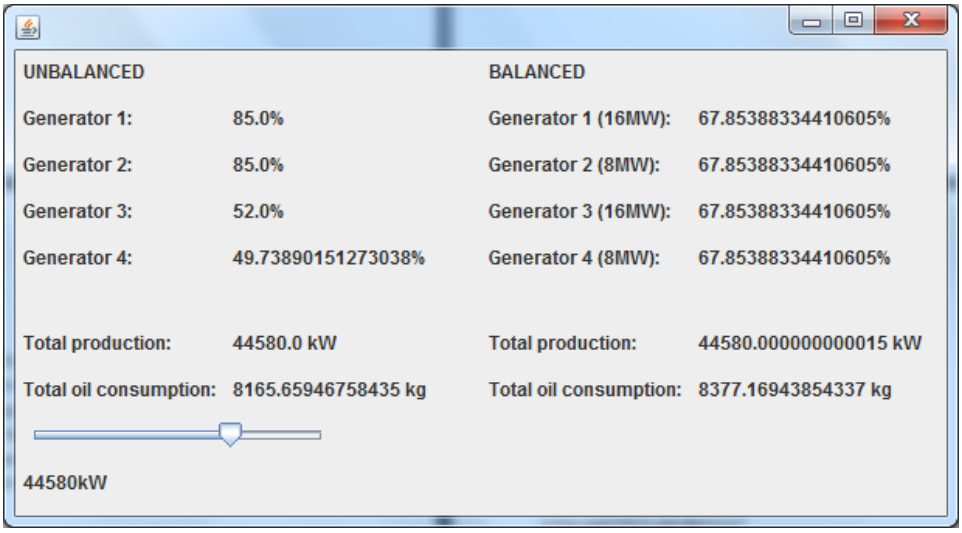

Figure 28 Fuel savings of up to around 200 kg every hour

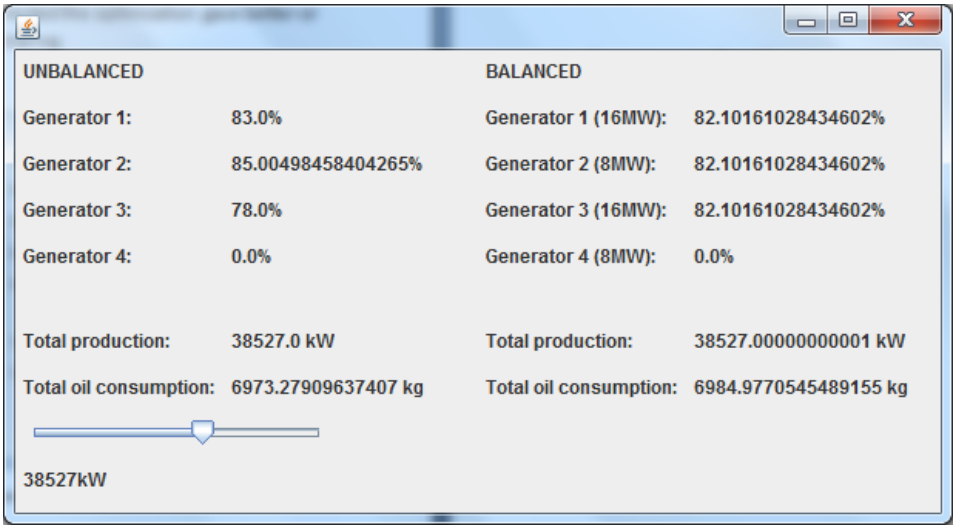

Figure 29 Another situation showing less savings

To get a better picture of how the actual savings will be, several different typical situations were tested. The three main scenarios are:

1. High speed: 4 generators running with an average load of 85%.
2. Normal speed: 3 generators running with an average load of 80-84%. Tested at 80% and 84% load.
3. Half speed: 2 generators running with an average load of 60-80%. Tested at 60%, 70% and 80% load.

If we postulate that while the ship is in optimization mode it is going high speed 5% of the time, normal speed 75% of the time and half speed 20% of the time (based on experience

within the Metso team[1]) this amounts to an average saving of 25,83 kg of fuel per hour (individual test results can be found in Appendix I). With today's fuel oil prices of around 610 USD per ton [11] this is equal to USD 15,76 of savings per hour.

A typical 7 day cruise is at port 8 hours for five of the days. The rest of the time it is at sea. If we assume that, when the ship is at sea, the power plant will be in maneuvering mode for 1 hour each day and the rest of the time in optimization mode (based on experience within the Metso team[1]) this amounts to a yearly saving of USD 99 434.

[1] Experience is based on how much the ship is usually in sea mode. For the purpose of this thesis we assume that the power plant will run in optimization mode all of the time it has previously ran in sea mode.

7.3 Load dependent start/stop

The Metso system has a built in strategy for starting and stopping generators. It was necessary to test that the load dependent start/stop strategies still works in optimization mode. In this test scenario the generators are called DG1, DG2, DG3 and DG4. DG1 had priority 1, DG2 had priority 3, DG3 had priority 2 and DG4 had priority 4.

7.3.1 Load dependent start

In this test DG1 and DG3 was running with an initial power demand of 27,5MW. The power demand was increased to 29MW which should cause a start up of DG3.

Figure 30 shows the load of DG1 (blue) and DG3 (orange) increasing as the power demand increases. This causes a start up of DG2 (green) which after a while begins synchronizing and the generators stabilize on their new set points.

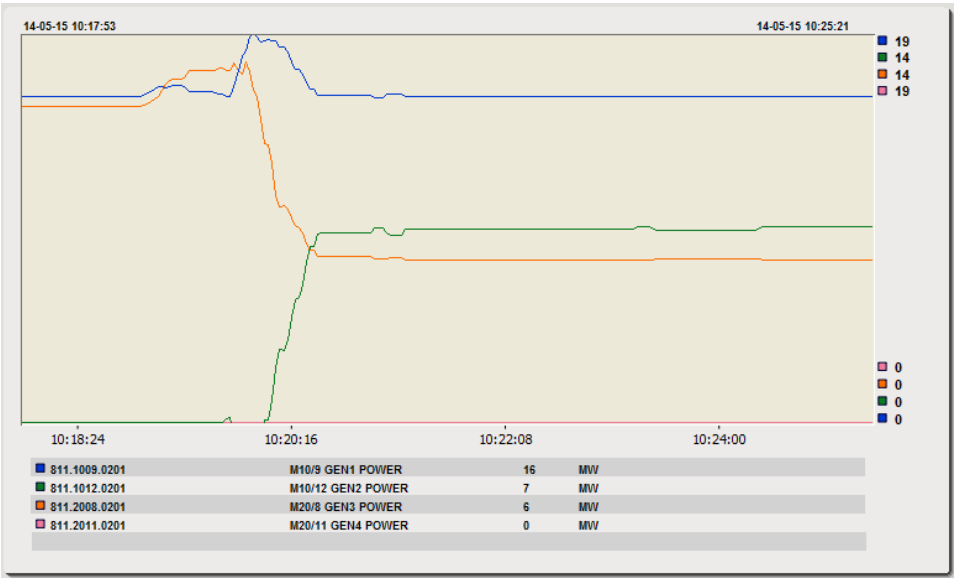

Figure 30 Load dependent start

7.3.2 Load dependent stop

In this test DG1, DG2 and DG3 was running with an initial power demand of 23,5MW. The power demand was decreased to 22MW which should cause DG2 to unload and shut down. Normally the stop delay is several minutes, but for the purposes of this test it was set to 45 seconds.

Figure 31 shows the load of DG1, DG2 and DG3 decreasing as the power demand decreases. This starts the stop delay

counter and after 45 seconds DG2 starts unloading and shuts down. DG1 and DG 3 stabilize on their new set points.

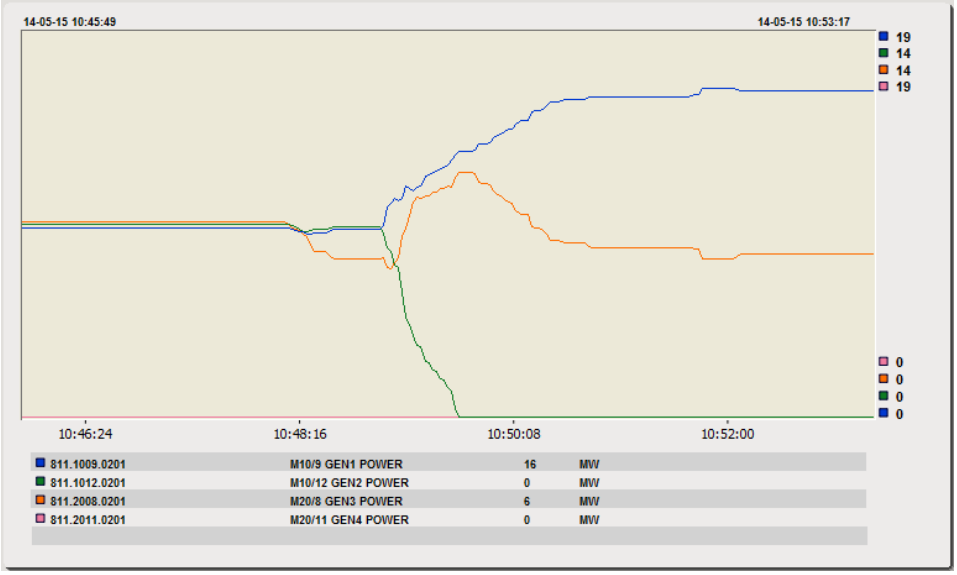

Figure 31 Load dependent stop

7.4 Safety

The safety functions were tested by simulating the different fault scenarios in the simulator. In the test situations three generators were initially ran (DG1, DG2 and DG3) with an average load of 60% (28 MW). DG1 had priority 1, DG2 had priority 3, DG3 had priority 2 and DG4 had priority 4.

7.4.1 Shutdown

Shutdown of a generator can happen if an unexpected error occurs. If this occurs the load from this generator will instantly be distributed amongst the remaining generator. A safety mechanism that shuts down the thrusters if the remaining generators are in overload exists. It is not a part of the Metso DNA system; rather it is built in to the generators control logic. It was therefore only possible to test a scenario where the remaining generators did not overload.

Figure 32 shows how the system reacted when we simulated a shutdown of DG3 (orange). DG1 (blue) and DG2 (green) instantly take the load. DG4 (pink) instantly starts up, but it takes a while before it is ready to synchronize with the other

generators. After a while it starts synchronizing and the generators stabilize on their new load set points.

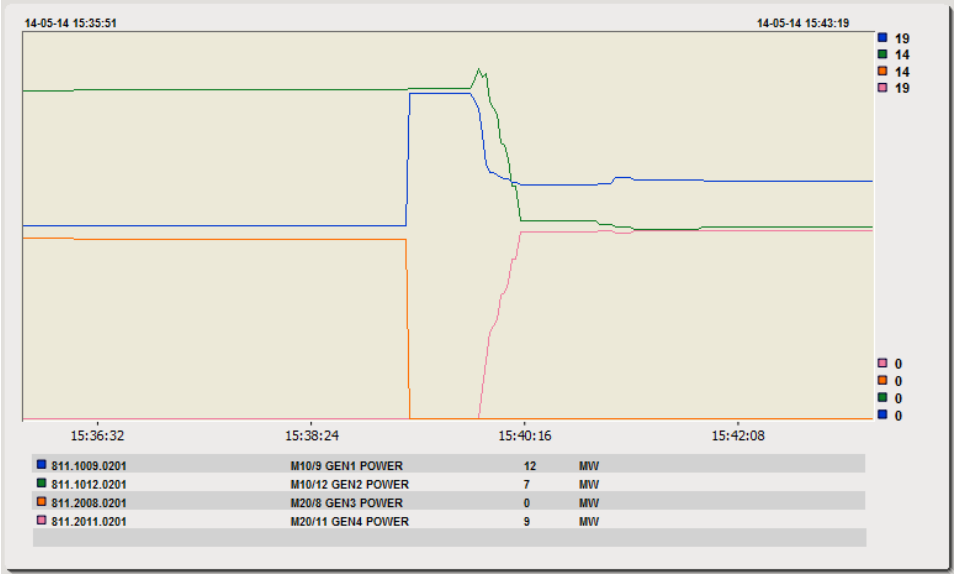

Figure 32 Shutdown of DG3

7.4.2 Load reduction

A load reduction request can be sent to a generator for a number of reasons, the main reason being that the engine is overheating. A load reduction request is generally sent to a generator in cases where it is no longer desirable to have the generator running, but there is no need for an immediate shutdown. When a load reduction request is sent another

generator starts up and when the new generator is starting to synchronize the generator causing the load reduction request will start unloading.

In Figure 33 a load reduction request was sent to DG3. When DG3 starts unloading DG4 starts synchronizing. DG1 and DG2 also get new load set points according to the new optimal situation. After a while the generators stabilize on their new set points.

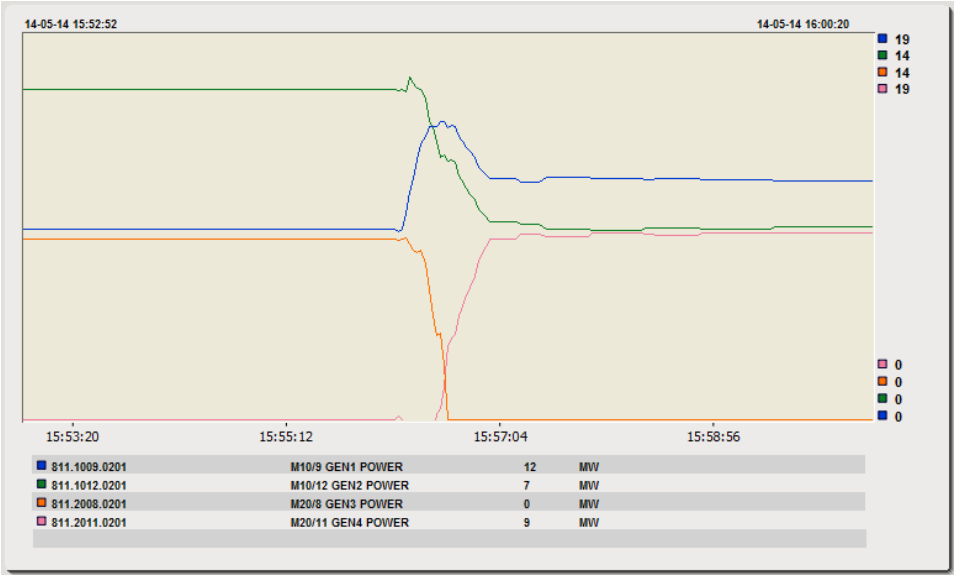

Figure 33 Load reduction of DG3

7.4.3 Start failure

A start failure means that the generator that is supposed to start, according to its priority, fails to start. This can happen if the start command fails to reach the generator, or if something is wrong with the generator itself. When a start failure occur the generator with the next highest priority should start.

In this case DG1 and DG3 were running with a power demand of 12,5MW while increasing the power demand to 14MW. A start up failure was then simulated on DG2 which should cause DG4 to start up.

Figure 34 shows this situation. As the power demand is increased from 12,5MW to 14MW we see that the load on DG1 and DG3 increases. This load increase should have started up DG2, however since DG2 has a start failure this does not happen. After a while DG4 starts up instead according to the start up failure procedure.

Figure 34 Start failure of DG2

7.4.4 Safety summary

All safety functions were tested successfully. This means that the optimization module was successfully implemented without compromising the safety of the system.

Chapter 8

Discussion

The algorithm itself is solid since every legal set point combination, with a resolution of 1%, is tested. There is a point to be made about increasing the resolution. However, since the resolution of the input is so low it is no way of knowing if we would actually achieve a more optimal situation or a less optimal situation. The linearization between the input points helps to create an image of how the SFOC curves look like, but there will be minor variations between the points which the linearization will not show.

The savings proved to be very significant in certain situations and not so significant in other situations. The average saving however was 25,83 kg of fuel per hour (for the tested scenarios). This is a significant amount of fuel and the potential savings are high, but as important is the positive environmental effect. Less fuel used means reduced emissions, and is both good for the environment and the environmental profile of the customer.

The results presented in this thesis are still only theoretical since the optimization module has only been tested in a simulated environment. The simulator has been refined by Metso over the years and has proven to be reliable. There is however a chance that problems will occur on the real system that the simulator has not taken into account. The system therefore needs to go through extensive testing on a real system for the final approval.

Since the resolution of the input is not very high the accuracy of the algorithm will not be 100%. If the resolution of the input was higher the accuracy of the algorithm would be better as well. If the resolution of the input was higher than the current output resolution this means that the algorithm could be modified so that the resolution of the output would be better than it is now. However, this would mean that the algorithm would have to look through a larger set of points which means that the run time would increase. If the run time is above 200 ms the algorithm would render useless given that the Metso DNA system is not modified.

Another challenge for this algorithm would be if the number of generators increased. This would also result in larger data sets to look through, which again would increase the run time.

This means that the algorithm, as it sits, is not very expandable without doing any changes to the Metso DNA system. However, since the regulators in this system are fairly slow there is in fact no reason why the 200 ms time slot could not be expanded. This would mean some minor changes in the Metso DNA system, but these changes would not be hard to make. It is however important to make sure that the timeframe is not too large since this is a real time system, and there would be a need to analyze how large this time frame could be without interfering with functionality.

Chapter 9

Further work

In the future there might be a need to expand the algorithm to include a larger number of generators. Since this will increase the run time of the algorithm the time slot in the Metso DNA system may have to be increased. It is however a point to be made that this may not be a sustainable solution. A better solution would be to look towards optimizing or changing the algorithm itself.

If the algorithm is to be changed it should be changed in such a way that it would also handle a better resolution on both the input and the output. It might therefore be necessary to develop a “smarter” algorithm. A natural step forward would be to look towards nonlinear programming. The framework developed in this thesis could still be used. Only the algorithm itself would have to be replaced.

References

[1] I. M. Organization, "Chapter III Requirements for control of emissions from ships," in *Report of the marine environment protection committee on tis fifty-eight session, Annex 13*, 2008, pp. 14-27.

[2] G. Hellén, "Wärtsilä Product Development to comply with IMO Tier III (NOx)," 26 May 2010. [Online]. Available: http://meeting.helcom.fi/c/document_library/get_file?p_l_id=18827&folderId=1123321&name=DLFE-41646.pdf. [Accessed 27 May 2014].

[3] Metso, Metso DNA architecture and Ethernet Networks v1.3, Metso, 2014.

[4] M. Automation, Function Block CAD Manual, rev. 6 ed., vol. 2013, Tampere, 2013.

[5] M. Automation, Function Blocks, Tampere, 2013.

[6] M. Automation, "5 Runtime Environment," in *Prog2 Function Block Manual*, 7 ed., vol. 2013, Tampere, 2013, pp. 11-14.

[7] M. Automation, "4.3.1 Adding a new prog2 block," in *Prog2 Function Block Manual*, 7 ed., vol. 2013, Tampere,

2013, pp. 6-7.

[8] M. Automation, "4.3.6 Supported Data Types," in *Prog2 Function Block Manual*, 7 ed., vol. 2013, Tampere, 2013, p. 10.

[9] M. Automation, "4.3.4 Java program skeleton," in *Prog2 Function Block Manual*, 7 ed., vol. 2013, Tampere, 2013, p. 9.

[10] M. Automation, "7.9.2 Java program in a jar package," in *Prog2 Function Block Manual*, 7 ed., vol. 2013, Tampere, 2013, p. 30.

[11] "Bunker Index," 15 May 2014. [Online]. Available: http://www.bunkerindex.com/. [Accessed 15 May 2014].

Appendixes

Appendix A	-	Simple Algorithm
Appendix B	-	Modified Algorithm
Appendix C	-	Java Skeleton
Appendix D	-	SFOC Vector generator
Appendix E	-	Simple Simulator
Appendix F	-	Equal Load
Appendix G	-	Simplified start/stop
Appendix H	-	prog2 interface
Appendix I	-	Fuel saving tests
Appendix J	-	Javadoc

Appendixes are located on the CD included in the booklet.