Fuel Tankering

- Flygprestanda AB

LUNDS UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Department of Computer Engineering

Bachelor thesis:
Martin Lindgren
Jonathan Brynhagen

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund 2012

Abstract

One of the biggest expenses for airline companies is the fuel for the aircrafts; preceded only by labor costs it is important for an airline to optimize its use of fuel[1]. One option to save money on fuel is by doing something called Fuel Tankering. Essentially, fuel tankering is a way to lower the fuel cost by refueling at departures where the fuel price is lower than at the destination of the aircraft.

It is difficult for a pilot to predict whether or not he can make a profit by tankering and if he can, how much extra fuel he should take onboard to gain the biggest profit possible.

To be able to calculate whether it is possible to make some profit on fuel tankering there is a lot of variables that needs to be taken into account, such as fuel tankering amount, flight distances, velocities, winds, altitude, local fuel prices, payload and more. All these variables are available in Flygprestandas flight planning system FOCS so our task was to integrate a fuel tankering solution into FOCS that retrieves accurate fuel consumption data with different amount of extra fuel loaded and uses this to present a tankering table in the tripkit for the pilot to see how much fuel the pilot should tanker to gain the most profit (if there is any profit).

The solution is integrated and fully working but certain enhancements can still be made. There are still some variables that are missing that will slightly affect the profit results, such as CO2 emission tax and different currencies. When these variables are added at a later stage our design allows these to be integrated without much difficulty.

Keywords: Tankering, flight planning, calculations, tripkit, profit

Sammanfattning

En av de största utgifterna för flygbolagen är bränslet för flygplanen; kostnaden föregås endast av arbetskostnader och därför är det viktigt för flygbolagen att minska utgifterna för bränsle[1]. Ett alternativ för att spara pengar på bränsle är att använda sig utav något som heter fuel tankering. Fuel tankering är en metod för att minska bränslekostnaderna genom att tanka på extra bränsle vid avgångens flygplats där bränslekostnaden är lägre än vid flygplanets destination.

Det är svårt för piloten att förutse om han kan spara pengar genom att fylla på extra bränsle eller inte, och om det går att spara pengar, hur mycket ska piloten då tanka på för att spara så mycket pengar som möjligt.

För att kunna beräkna eventuell vinst finns det många variabler som måste tas i beaktning, såsom mängden extra bränsle, ruttlängd, hastighet, vindar, höjd, bränslepriser, flygplanets vikt, m.m. Alla dessa variabler finns noggrant uträknat i Flygprestandas flygplaneringssystem FOCS. Vår uppgift var att integrera en fuel tankering lösning i FOCS som hämtar dessa data och därefter gör ett antal beräkningar så att vi kan presentera en lista i tripkiten för piloten som gör att han kan se hur mycket han ska tanka på extra för att spara så mycket pengar som möjligt (om där är någon vinst).

Denna lösning är integrerad och fungerar felfritt men där finns ett par förbättringar som kan göras. Där finns ett par variabler som saknas som kan komma att påverka resultatet något, såsom koldioxidskatt och valutakonvertering. När dessa väl läggs in så tillåter vår design det till att implementeras utan några stora svårigheter.

Nyckelord: Tankering, flygplanering, beräkningar, tripkit, vinst

Foreword

We would like to thank Flygprestanda for giving us the opportunity to do our thesis work for them. During our time there we have had help from Vadim Feldman and Lars Andersson together with the developers at Flygprestanda which we are very thankful for and without your assistance this thesis would not have been possible. We would also like to thank our examiner Mats Lilja for helping us with writing the thesis.

List of contents

1 Introduction 1
  1.1 Background 1
    1.1.1 Flygprestanda AB 1
    1.1.2 FOCS 1
  1.2 Problem Description 2
  1.3 Technical background 3
    1.3.1 FOCS 3
    1.3.2 Database 3
    1.3.3 Java and Eclipse 3
    1.3.4 Subversion (SVN) 3
    1.3.5 Apache Maven 3
  1.4 Goals 3

2 Project Model 4
  2.1 Time Plan 4
    2.1.1 Preparation and Investigation 4
    2.1.2 Programming & Testing 5
    2.1.3 Thesis 5
  2.2 Project Model 5

3 Investigation 7
  3.1 Current Fuel Tankering Solutions 7
  3.2 Real-world usage of Fuel Tankering 7
  3.3 Risks with fuel tankering 7
    3.3.1 Cold soaked wings 7
  3.4 Calculations 8
    3.4.1 Initial pre-study 8
  3.5 What data is required 11
    3.5.1 Terminal Charges 11
    3.5.2 Overflight Fees 11
    3.5.3 Future Planned Routes 11
    3.5.4 Weather 11
    3.5.5 Route 12

4 Implementation 13
  4.1 Prototype 1 13
    4.1.1 Proposed solution 13
    4.1.2 Implementation 13
    4.1.3 Problems 14
    4.1.4 Result 15
  4.2 Prototype 2 16
    4.2.1 Proposed solution 16
    4.2.2 Bug fixing & Optimization 16

4.2.3 Enhancements 18
4.2.4 Problems 20
4.2.5 Result 22
4.3 Prototype 3 22
4.3.1 Proposed solution 22
4.3.2 Bug fixing 23
4.3.3 Enhancements 24
4.3.4 Result 24
5 Conclusion 25
6 Further enhancements 26
6.1 Currency conversion 26
6.2 CO2 emission tax 26
6.3 Three or more flight legs 27
6.4 Mask fuel-prices in tripkit 27
6.5 Optimization 27
7 Terminology 29
8 References 30
9 Appendix 31
9.1 Prototype 2 31
9.2 Prototype 3 32

1 Introduction

This report describes the different parts of the actual thesis work done at Flygprestanda to integrate a fuel tankering solution into their flight planning system FOCS. It contains different parts of the project phases such as investigation, and implementation where it is explained in detail methods used, problems encountered, proposed solutions, and phases during this project.

The purpose of this thesis is to first investigate whether or not you can make a profit or loss by fuel tankering, and if there is profit, implement a solution that will make it easy for the pilot to see how much money he can gain/lose if he tankered a specified amount of extra fuel.

Due to respect of Flygprestanda AB, this report does not contain any actual code from either the existing system (FOCS) or our solution. Our solution is explained on such a level where no sensitive information is revealed. Some names of the classes have been renamed. However, our result is not affected by this and will be visible in this report.

1.1 Background

When we asked our Programme Manager Christin Lindholm about recommendations for companies to do our thesis work with, Flygprestanda AB was one of them. We heard about students that had done their thesis work here in the past and that the collaboration was successful so we contacted the company via e-mail. They quickly replied with enthusiasm which made us excited to come for a visit so a meeting was arranged the following week. On the meeting a basic problem description was given with an introduction of the company. We thought it sounded interesting so we decided to do our thesis there. Another meeting was booked where some ethical rules and company policies were discussed.

1.1.1 Flygprestanda AB

Flygprestanda AB [2] makes software and databases for hundreds of airline companies all over the world. Their services include both software development and technical-engineering calculations with focus on delivering all information needed to perform a commercial flight from one place to another. Flygprestanda AB has around 50 employees and has their head office in Malmö, Sweden and has another office in the United States. The company was founded 1969.

1.1.2 FOCS

FOCS is a system developed by Flygprestanda AB to do flight planning and other flight related calculations. It allows the user to, with a few clicks, find the best route possible taking into account all parameters required to get the best/most economical route possible. Furthermore it analyzes NOTAM data and other sources of information to keep track of obstacles on the way.

1.2 Problem Description

To earn money, an airline company needs to have their aircrafts up in the air as much as possible. The problem is that fuel is very expensive and it keeps getting more expensive every day. There are a lot of things you can do to lower the fuel consumption, for example:

- Aircrafts fuel efficiency
- Reducing the payload (fuel also weighs, so you only refuel the amount of fuel that is needed for the trip and for the alternate departure airport)
- Optimizing the flight routes
- Regular aircraft maintenance
- Fuel tankering

Fuel tankering is a way to lower the fuel cost by buying extra fuel in other countries where the fuel is cheaper. You may think that if the fuel is cheaper at the arrival country you can just refuel as much as possible. Unfortunately, it is not that easy.

For example, if you fly between Malmö and Stockholm, you will have to fuel up around two tons of fuel (this is of course, actually depends on the route and the aircraft) and the more fuel you have, the more your airplane weighs. The more your airplane weighs, the more fuel is needed for the engines.

So the problem is how the pilots should know how much extra fuel they should buy based on if the fuel price is cheaper at the departure than at the arrival to get the most money out of it.

Our task is to investigate and implement an aircraft fuel tankering module into FOCS. The users of the system should be able to make flight planning and routing decisions based on the information from the fuel tankering module.

1.3 Technical background

1.3.1 FOCS

FOCS consists of a client and can’t be run without a FOCS server. The client is where the user make flight plans and such and the server is where the calculations are made and where all the data is stored.

1.3.2 Database

The database that is used is MySQL and will be administrated using phpMyAdmin, which is a web administration tool for MySQL databases. The database communicates with the server in order to retrieve and save data. The database contains information about aircrafts, data about the actual flight plans, route segments, etc.

1.3.3 Java and Eclipse

FOCS is developed in Java so this will be the language we use. Eclipse will be our IDE with a few plugins in order to integrate revision control and project dependencies into Eclipse. These are Subclipse and Maven.

1.3.4 Subversion (SVN)

In order to get the latest branch from FOCS we use SVN as the revision control system. SVN is a tool used for retrieving earlier and current versions of documents and source code. It is also used for tracking changes between versions.

1.3.5 Apache Maven

For build automation and project dependencies within FOCS it is required to use Apache Maven. Maven is a tool used for Java that automatically builds projects into a distributable unit.

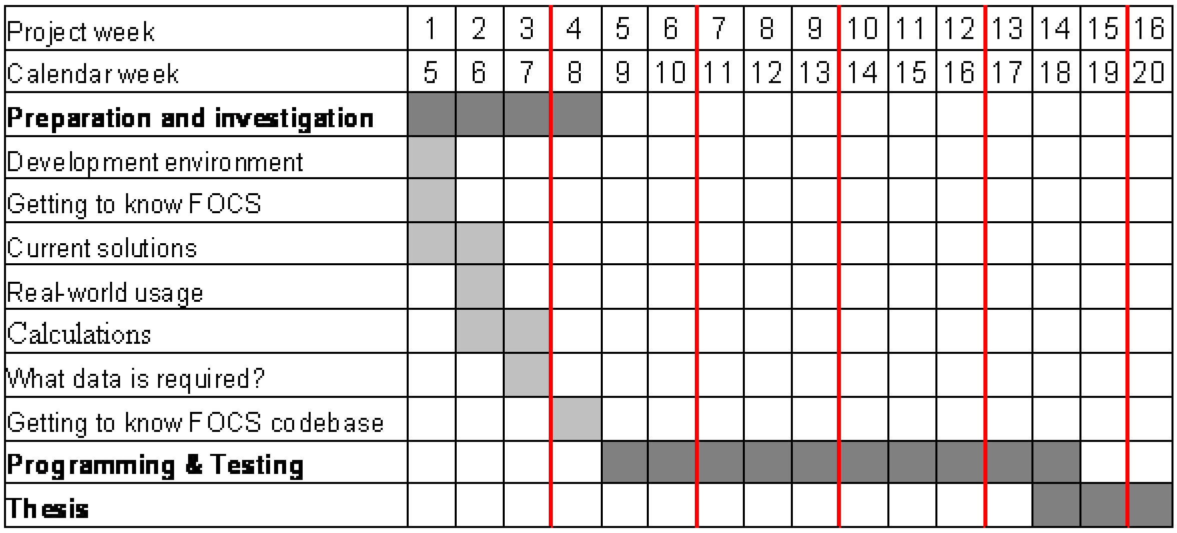
1.4 Goals

The main goal of the project is to integrate a fully working solution with the current system FOCS to give an as accurate result as possible with minimum amount of user interaction. To reach this goal an investigation must be made to determine whether or not fuel tankering can be used to make a profit for the airlines. An examination on what the current fuel tankering systems do shall be made so it can be determined what can be made better.

The solution should be as modular as possible and be written in the coding standards specified by Flygprestanda.

2 Project Model

2.1 Time Plan



Project week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Calendar week	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Preparation and investigation	█	█	█	█												
Development environment	░															
Getting to know FOCS	░															
Current solutions	░	░														
Real-world usage		░														
Calculations		░	░													
What data is required?			░													
Getting to know FOCS codebase				░												
Programming & Testing					█	█	█	█	█	█	█	█	█	█		
Thesis														█	█	█

Figure 1 – The initial time plan

2.1.1 Preparation and Investigation

Development environment
A development environment has to be set up. Eclipse, Maven and subversion are all necessary tools that has to be installed and configured.

Getting to know FOCS
FOCS is the program that will be integrated with our solution so it is very important to know what FOCS is and how the users interact with it and how it should be presented with our solution in the GUI.

Current solutions
Before starting to think about the problem it can be helpful to study existing solutions, in order to gain some insight into the current status of the field.

Real-world usage
How useful can our solution be? Do most airline companies use any fuel tankering method? Here an investigation will be made about the usage of fuel tankering around the world.

Calculations
A series of calculations must be made in order to get an accurate result. This part is dedicated to coming up with some ideas for getting the most accurate result possible.

What data is required?

Before implementing an investigation should be made on what data is needed in order to get an accurate output. Deciding what data that is most important and how they depend on each other is also necessary.

Getting to know the FOCS codebase

Before development can begin it is necessary to familiarise ourselves with the current codebase so that we may know where our module fits in and how the system is designed.

2.1.2 Programming & Testing

This is where the actual work for Flygprestanda is done and where the implementation of our solution into FOCS will be made. Testing will be made on the solution iteratively to see if it works properly.

2.1.3 Thesis

Work on the thesis will be done throughout the course of the project; this is more of a reserved space for any thesis work remaining after the project is completed.

2.2 Project Model

The project model used will be a light version of Kanban where the implementation phase will be split up into four sprints. A Kanban board will be set up at our office that will use three phases; *backlog*, *in progress*, and *done*. At the beginning of each sprint, it will be discussed on what should be focused on at this prototype, and what things that should be done in this prototype. At the end of each prototype, it will be discussed what needs more focus and what problems were encountered during the sprint.

At the end of these sprints the prototype shall be launched and demonstrated. Marked in red on the Time Plan are weeks where the investigation, prototypes and implementation shall be finished.

Deadline one: A formula should have been made to see how much gain or loss fuel tankering would give in a given case. It should also be made clear what fuel tankering systems are out there today, and how the solution may differ from others. At this point there should be a general idea of what data is required.

Deadline two, three, and four: By now a prototype shall be ready for demonstration. It may not be fully implemented, but for each sprint this prototype will be more and more integrated into FOCS. Every new prototype should have more in-data to rely on to get an even more accurate output.

Deadline five: Here the prototype will have been developed into a fully working solution. It should be fully integrated into FOCS and it shall give an accurate output with small interaction from the user.

All work will be done Monday through Thursdays at the Flygprestanda office except for a bit of the thesis work. All equipment for development is supplied by Flygprestanda. The system shall be developed in Java programming language using Eclipse as development environment.

3 Investigation

3.1 Current Fuel Tankering Solutions

There are systems out there today that deal with Fuel Tankering. Most of them are inaccurate and the rest are not simple enough to use. What is unique about our implementation is that it will require close to no interaction from the user as all the necessary data is already stored within FOCS.

3.2 Real-world usage of Fuel Tankering

Fuel Tankering is being done by many airlines today but most of them lack a proper tool and instead go by general figures or speculations that are not always accurate.

In some cases, the pilots are calculating on it on their own with a simplified formula that is called the 3-5% rule (Discussed in [7]). This rule says that for every hour travelled you burn between 3-5% (dependent on the aircraft) of the extra fuel tankered.

In some other cases the pilots have a fuel tankering table that consists of trip distance and a break-even price ratio. The pilot will take the fuel price where he departed from (where the fuel is cheaper), and divide it with the break-even price ratio for the given distance. If the result is higher than the fuel price at arrival (where tankering is intended to) then fuel tankering is to be considered.

Today there are also fuel tankering solutions that will do all the calculation for you which are more accurate than the previous methods. These methods require the user to enter data such as aircraft, trip distance, and route.

3.3 Risks with fuel tankering

3.3.1 Cold soaked wings

Cold soaked wings [3] is a phenomena where ice will form on the wings even though the air temperature can be well above zero degrees.

Most of the aircrafts today are equipped with fuel tanks in their wings. The problem is that if you are flying on a high altitude for a long period of time where the air temperature is below zero, the temperature of the fuel in the wings can get to below zero, which will also make the wing surface temperature below zero. When descending, if the wings come in contact with liquid water, such as condensation or rain, the wings will begin to freeze.

This effect can have serious consequences, because it can reduce the speed to such a degree that the aircraft cannot even reach the minimum speed for take-off, or maintain flight. If you get cold soaked wings, you will need to de-ice your aircraft. De-icing means that you are warming up the fuel in the wings to above zero to prevent ice from forming on the wings. This can take a long time and can get the aircraft to be delayed for next flight, and money is lost.

3.4 Calculations

3.4.1 Initial pre-study

The first investigation made was to see whether or not you could even make any profit by fuel tankering.

FOCS was started and a specific route was entered with the aircraft Embrear E135. Then it could be seen how much fuel the airplane would spend on the trip without any extra fuel on it. It was then added an extra 500 kg fuel to the aircraft to see how much fuel it would have spent extra compared to the first case.

Calculations were made that ended up with a simple inequality which gives the condition for using fuel tankering:

$$P_{dep}\left(F + F_B(F)\right) < P_{des} \cdot F$$

P_{dep} stands for fuel price at the airport you are departing from
P_{des} stands for fuel price at destination,
F stands for extra fuel tankered. This is the variable that was used as an input.
F_B is a function of for extra fuel burned (from carrying F). This number is calculated using many variables, such as the weight of the airplane with F loaded onboard, the route, weather, etc. FOCS does this very accurately already so this data is used. This is the output from FOCS.

As F increase so will F_B and if the prices differ greatly then the cost of this is quite negligible.

If the left hand side is less than the right, you will gain money by tankering extra fuel. The formula was applied to the numbers returned from FOCS and it was made clear that if the price difference was big enough, money could be earned.

The next investigation was to determine how the gain/loss changed depending on the route length. It was also necessary to determine if the function between extra fuel loaded and money gained/lost was linear or not.

FOCS was launched and a specific route was entered with the aircraft E135.

Notes were taken for every 100 kg fuel tankered to see how more the aircraft would spend in terms of fuel. Extra fuel added was increased by 100 kg until MTOM (Maximum Takeoff Mass) was reached.
The data saved was tankered fuel, trip fuel, total fuel, ATOM (Actual Takeoff Mass) and ALM (Actual Landing Mass).

A diagram was made to see if the function was linear or not. This was done for three different routes with different length to determine if there is any relation between the distances regarding the gain/loss. To ensure that the results are consistent we turned the weather calculations off for these diagrams.

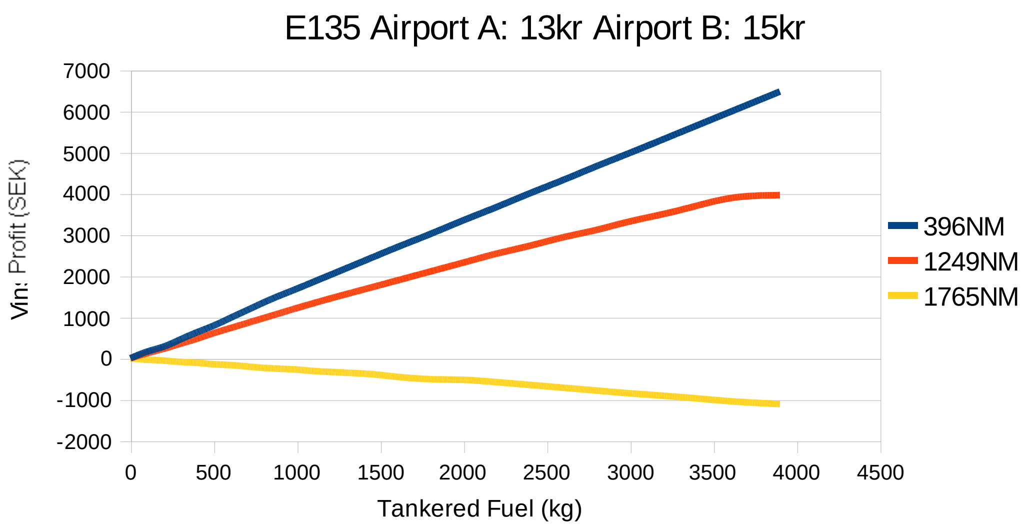


Figure 2 – Profit diagram for the aircraft E135 with different routes

In figure 2, 3 and 4, the x-axis shows how much fuel is tankered, and the y-axis shows how much money you will gain/loss. The diagram shows that there is a relation between the distance and the money gained/lost. The longer the distance the more expensive it is to carry the extra fuel, and eventually the price for the extra fuel burned exceeds the money saved by fuel tankering. In making the diagrams an empty airplane is assumed, loaded only with the required fuel for the flight and a pilot.

In figure 3, the same test was made with the aircraft Embraer E145 and the results turned out quite similar.

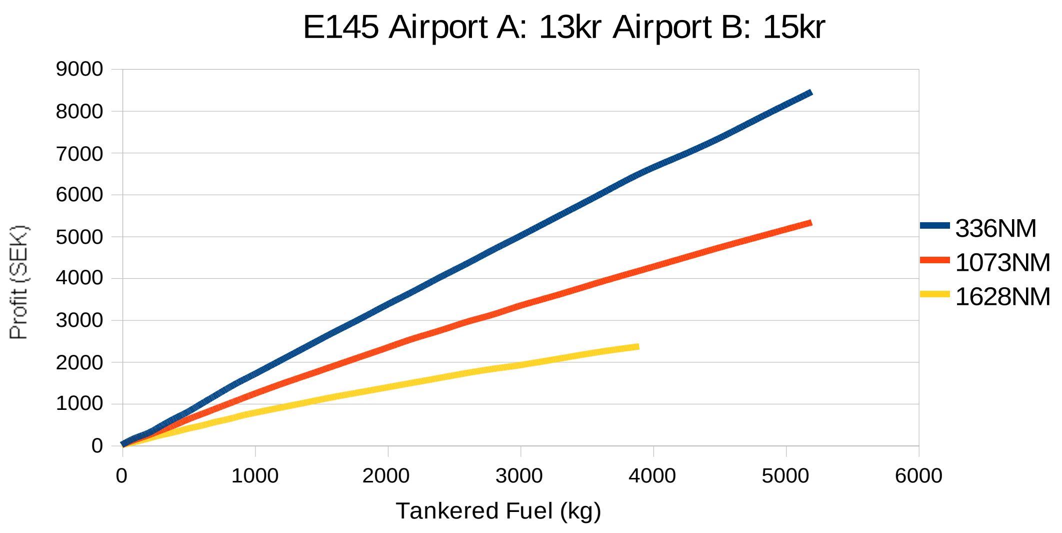


Figure 3 – Profit diagram for the aircraft E145 with different routes (NM = Nautical miles)

The results show that the function is more or less linear for short distances but the rate of the gain decrease as distances increase. For amounts of fuel that is to be considered realistic for this airplane the function can be treated as linear.

In figure 4, the same test was made with a much larger airplane that can carry a lot more fuel. The results point in the same direction where shorter distances are linear but for longer distances the fuel cost depends more non-linearly on the tankered fuel as the aircraft gets heavier.

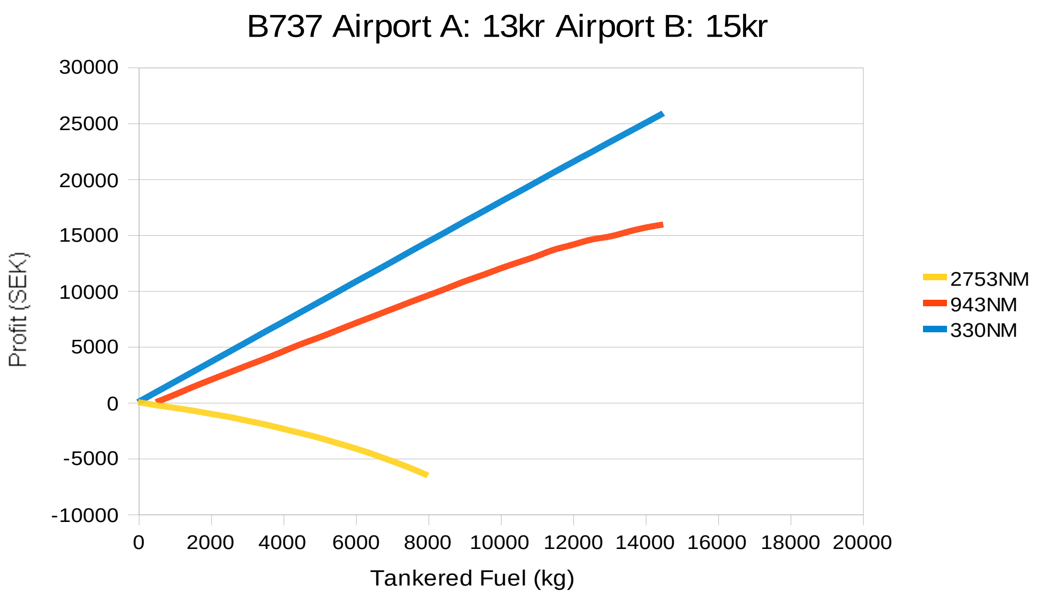


Figure 4 – Profit diagram for the aircraft Boeing 737 with different routes

3.5 What data is required

3.5.1 Terminal Charges

Terminal charges are fees paid by airlines provided by the airports. Whenever an aircraft is going to use any service provided by the airport such as: (Taken from [4])

- Use of the runway (landing charges)
- Use of the airport infrastructure (parking and boarding bridge charges)
- Use of the terminal building (passenger charges)
- Airport security (security charges)
- Protection of the environment (noise charges)
- Air traffic control (en route navigation and terminal charges)
- Other air navigation services (meteorological and aeronautical information services)

The airline is going to need to pay a terminal charge. The terminal charge varies from airport to airport, and no data can be found regarding the change of charges depending on the aircraft weight, so these are not going to be factored into the calculations.

3.5.2 Overflight Fees

A factor that was considered to take into account was overflight fees. An overflight fee is a charge that must be paid in order to use a certain airspace and it is based on distance traveled, airplane weight and unit rate of charge. While tankering certainly increases airplane weight, it is not the actual take-off weight being measured, instead they use something called Maximum Take-off Weight (MTOW for short) [5]. MTOW is the maximum allowed weight for an airplane to take off and so tankering extra fuel does not affect this.

3.5.3 Future Planned Routes

Having additional destinations planned is something that can definitely be taken into consideration to make extra profit with fuel tankering. For example: you are taking off from Airport A with a fuel price of 13kr/kg. Your destination is Airport B where the fuel costs 15kr/kg. After landing at B you are scheduled to fly to Airport C where the fuel is only 10kr/kg. Under certain conditions the best option would be to fuel up as much fuel needed at Airport A to take you to Airport C and then tanker again there but this is not always true.

3.5.4 Weather

Weather is a big factor in calculating fuel consumption. Wind, temperature, atmospheric pressure are all some of the parameters that need to be accounted for. This is all done in the current version of FOCS so this will not have to be included in our calculations, however as mentioned above when carrying fuel in low temperature areas icing on the wings might occur and it costs time and money to remove it. This is too situational to include in our calculations though and the decision has to be made on a case to case basis.

3.5.5 Route

The route you fly is obviously the biggest impact on fuel consumption. Altitude, velocity and distance are some of the parameters here. These are, again, calculations already being done in FOCS and thus will not have to be made by us. Tankering a bunch of fuel might impact the route selected however and this might have to be taken into consideration.

4 Implementation

This chapter will be describing the implementation phase of the different prototypes for this project. Each prototype will be described as an ongoing process divided into sections of proposed solution, implementation, problems and ends with a reflection on the results of each prototype.

4.1 Prototype 1

4.1.1 Proposed solution

The first idea on how to solve this problem in the first prototype is to re-calculate the performance for the same trip by adding extra fuel between the performance-calculation iterations. A search is needed to determine where within the source code the fuel calculations are made. A new class will probably be created that will contain the data needed for fuel tankering calculations between the results in order to get the most optimized results.

The goal for this prototype is to get a decently accurate result that later can be improved and optimized further in the following prototypes.

4.1.2 Implementation

The first thing that had to be done was to find how the essential data could be retrieved to perform calculations, and where the solution should be integrated. The class `PerformanceData` contained information regarding the actual flight (ATOM, ALM, and such). The `Aircraft` class contained useful information about the MTOM, MLM (Maximum Landing Mass).

In order to determine the place where the performance calculations actually were done, a search was made. The class `FlightPlanningClass` calls on a method that does performance calculations and then returns a `PerformanceData` object where all the results for the actual flight are stored. These performance calculations are implemented in another project which FOCS uses to retrieve performance values. It seems appropriate to implement the solution in `FlightplanningClass` because it was noticed that it should be possible to change the needed data for the flight before the calculations are executed here.

The next step was to try to calculate the same trip but with extra fuel. It took some time to find a way to change the data in `PerformanceData`, but at last the result gave different trip fuel burns, even though the execution time was very long due to our implementation.

The different trip fuel burns had to be saved before making calculations on them. For this reason a class called `FuelTankeringHelper` was created. Within this class all the necessary calculations are made as well.

In `FlightPlanningClass` a method was implemented for capturing all the data and doing calculations on them. In pseudo-code, it looks like this:

```
While the aircraft is not too heavy and fuel capacity is not
exceeded
{
           Increase the extra fuel tankered
           Clear the performance for the previous flight

           Calculate new Performance and check that ATOM,ALM,Fuel
                      capacity isnt too high

           Calculate tankering profits

           if the current profit < largest profit  for the flight
                                 break
}
Do some final calculations
Print out the values
```

The process can be illustrated in a block diagram in the following figure.

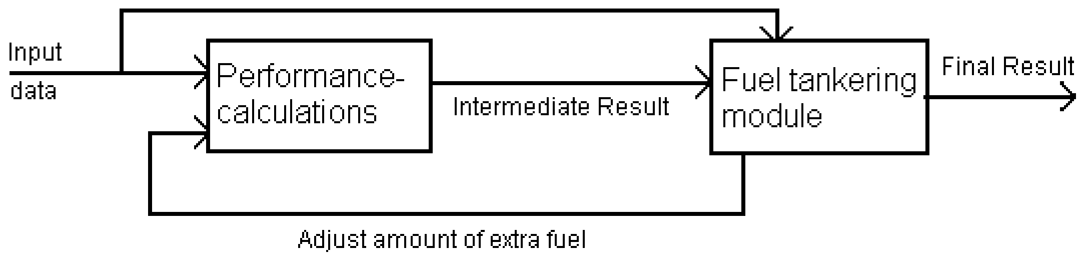


Figure 5 – A block diagram illustrating the fuel tankering process

The first block returns performance calculations for the flight (fuel used, weights, time etc.). This data is then handled by our fuel tankering module which determines whether the performance calculation should iterate again with more extra fuel. When the iterations are done, the tankering module should return a fuel tankering result.

Once implemented the first fuel tankering numbers was shown in the console.

4.1.3 Problems

- It is uncertain if the fuel tankering results are correct because there is no way of verifying the results.

• One or more values for tankering calculations are not being reset correctly sometimes which affects the result.
• Right now it takes approximately twenty times longer than normal to get the fuel calculation because of the tankering calculations. This needs to be optimized because there is a lot of redundant code at the moment.
• There were problems at first to get different trip fuel costs when adding extra fuel. It was discovered that if you selected an alternative airport the extra fuel was not calculated correctly, but if you did not select an alternative airport it was. Therefore the calculations will only work on non-alternative flights until this bug is fixed.
• The performance values for the flights are not saved between sessions. It has something to do with our fuel tankering calculations because it works without our implementation.
• The trip fuel cost values for a flight with a certain amount of extra fuel are not always correct.
• Sometimes it is not possible to add extra fuel between the iterations. This results in an infinite loop, because the weight or fuel capacity is never exceeded.

4.1.4 Result

After creating a flight the following data is presented in the development console.

```
Trip: 948.9      Tankered: 0       Total: 2788.7   (NaN kr/ton)      Profit: 0.0
Trip: 953.6      Tankered: 500     Total: 3297.2   (1879 kr/ton)     Profit: 939.5
Trip: 958.5      Tankered: 1000    Total: 3806.1   (1876 kr/ton)     Profit: 1875.9
Trip: 971.0      Tankered: 1500    Total: 4322.5   (1809 kr/ton)     Profit: 2713.9
Trip: 997.5      Tankered: 2000    Total: 4853.0   (1684 kr/ton)     Profit: 3368.3
Trip: 1016.8     Tankered: 2500    Total: 5376.2   (1647 kr/ton)     Profit: 4118.1
Trip: 1042.9     Tankered: 3000    Total: 5907.4   (1593 kr/ton)     Profit: 4778.8
Trip: 1062.4     Tankered: 3500    Total: 6431.9   (1579 kr/ton)     Profit: 5525.6
Trip: 1088.8     Tankered: 4000    Total: 6963.4   (1545 kr/ton)     Profit: 6182.0
Trip: 1107.8     Tankered: 4500    Total: 7487.4   (1541 kr/ton)     Profit: 6935.5
Trip: 1127.8     Tankered: 5000    Total: 8012.4   (1535 kr/ton)     Profit: 7675.5
Trip: 793.3      Tankered: 5500    Total: 8183.0   (2368 kr/ton)     Profit: 13023.5
Trip: 1037.1     Tankered: 6000    Total: 8931.8   (1809 kr/ton)     Profit: 10853.8 Loss
Trip: 1062.5     Tankered: 6500    Total: 9462.3   (1773 kr/ton)     Profit: 11523.8 Loss
Trip: 1081.8     Tankered: 7000    Total: 9986.6   (1753 kr/ton)     Profit: 12273.3 Loss
Trip: 1100.2     Tankered: 7500    Total: 10510.4   (1738 kr/ton)    Profit: 13034.0
Trip: 1119.7     Tankered: 8000    Total: 11035.3   (1722 kr/ton)    Profit: 13780.0
Trip: 1138.5     Tankered: 8500    Total: 11559.4   (1710 kr/ton)    Profit: 14535.7
Trip: 1150.8     Tankered: 9000    Total: 12077.1   (1708 kr/ton)    Profit: 15375.7
Trip: 1169.7     Tankered: 9500    Total: 12601.4   (1698 kr/ton)    Profit: 16129.6
Trip: 841.0      Tankered: 10000   Total: 12778.0   (2140 kr/ton)    Profit: 21403.7
Trip: 1076.3     Tankered: 10500   Total: 13518.7   (1842 kr/ton)    Profit: 19345.0 Loss
Trip: 1094.1     Tankered: 11000   Total: 14041.9   (1828 kr/ton)    Profit: 20113.4 Loss
Trip: 1112.5     Tankered: 11500   Total: 14565.7   (1815 kr/ton)    Profit: 20874.0 Loss
Trip: 1124.5     Tankered: 12000   Total: 15084.5   (1810 kr/ton)    Profit: 21718.1
Trip: 1135.6     Tankered: 12500   Total: 15602.4   (1806 kr/ton)    Profit: 22574.1
Trip: 1147.9     Tankered: 13000   Total: 16121.6   (1801 kr/ton)    Profit: 23414.2
Maximum fuel capacity reached
Price at Airport ESMS: 13.0
Price at Airport ESGG: 15.0
Most money earned at: 13000 kg of tankered fuel
You will earn: 23414.19264680159 kr (1801 kr/ton).
```

Clearly visible above is the inconsistencies in fuel required for the trip where for example at 5500kg of tankered fuel trip costs takes a major dip down.
For now this text is put into the developer-console but will be implemented somehow into FOCS in the following prototypes.

At first the costumer wanted to get the result in the unit kr/ton, but that value changes depending on how much the pilot is going to tanker. For this reason it would be better to present some kind of list for the pilot. A mail explaining this was sent to the customer.

4.2 Prototype 2

4.2.1 Proposed solution

In this prototype it will first be ensured that the solution from prototype 1 is correctly implemented and that the fuel tankering values are correct. This will be ensured by speaking to the manager of the software department at Flygprestanda and with a contact at City Airline.

Next, the bugs in prototype 1 must be taken care of and some stress tests will be made to find more bugs.

If there is time left, this will be spent by optimizing the code and implementing new performance methods that are going to be used only for fuel calculations (in prototype 1 performance methods were used that calculated more than fuel consumption, which made the calculations very slow).

An idea on how to solve this problem is to reduce the amount of iterations in the while-loop. In prototype one the flight started with zero amounts of tankered fuel, and then increased the extra fuel by a given value (right now the flight is loaded with 500 kg extra fuel for each iteration. A larger value yields faster calculation but is less accurate. This is also something that needs some further research to get the optimal value). Iterations were made until ATOM, ALM, or fuel capacity was exceeded. It was discovered that fuel tankering is often most profitable when reaching the MTOM/ALM or maximum fuel capacity, so therefore it would be better to start at the maximum fuel tankering amount and then decrease the fuel tankering iteratively, because then it would converge after a few iterations to the most profitable tankering amount. If a list of different tankering results is required to be presented this method cannot be used.

4.2.2 Bug fixing & Optimization

The first things that were fixed in prototype 2 were the major bugs in prototype 1.

The infinite loop bug was fixed by forcing the performance calculator to recalculate with different amount of extra tankered fuel. The performance calculator is written to be run only once per flight but by writing a new method, it was possible to get around that and make it run several times on the same flight but with different amount of extra fuel to get different tankering results.

In an effort to optimize and hopefully fix some of the bugs, a large part of the code was moved to an earlier stage. Now the fuel tankering calculations are performed after the standard calculations. This eliminated the bug where the fuel values were not saved.

All the methods were reduced until it only contained the essential operations. For example in prototype 1, methods that did performance corrections for different altitudes was used, which gave the exact same tankering results with or without it so this piece of redundant code was removed. When this was done, the bug where some values did not always reset was solved. This also made a huge difference on the time required for tankering calculations. The time required for the tankering calculations was reduced to be even faster than in the standard calculations, and yet this can be even faster by reducing the amount of tests.

The bug that caused the trip fuel to be lower when adding additional fuel was not actually a bug. It was caused by changes in flight levels (fuel burn rates changes depending on the flight level). This has to do with the algorithm for fuel calculation, because it is always trying to get the optimal fuel cost for the aircraft by, in this case lowering the flight level. Considerations were made about having a locked value on the flight level but after some discussion it reached the conclusion that it should not be, because the pilot probably wants to be flying at the given flight level for his aircraft weight anyway and the option to lock the flight level is already implemented into FOCS.

But even with this in mind, sometimes the trip fuel values got too much lower than the others. The bug was found at another FOCS server as well where this solution is not implemented so this bug has nothing to do with the implementation, though the fuel calculations depend on getting the correct values. Apparently, there is no climb calculation at some segments, which makes the trip fuel cost much lower. This bug has been reported and it is on its way to be fixed by others. The best way to bypass this right now is just to ignore the values if the trip fuel is much too lower than the previous one.

Mentioned earlier was an idea on how to reduce the amount of iterations by starting at MTOM/MLM/fuel capacity. This is no longer relevant because the result is going to be shown as a list with different amount of tankered fuel instead of just a number with profit per tonne of fuel.

4.2.3 Enhancements

There is a view in FOCS that is called Management where it is possible to store fuel prices for airports. The first enhancement in this prototype that was implemented was to gather the fuel prices from this view to the fuel tankering calculations instead of hard-coding it like before. The first solution was to try to get the data from another class but it was noticed that if you change the value of an already existing price, the value did not change in its flight plan. For this reason the data had to be extracted directly from the database.

This implementation led to several minor bugs. If the user stored prices in different currencies between the airports, the wrong result was obviously returned. A class that contained methods for converting a currency into another was found but this class was not fully implemented so for now fuel tankering is blocked when different currencies are used. It is also blocked when the price at departure is higher than at destination for obvious reasons.

An answer returned from the customer regarding how the data should be presented where he agreed with us and wanted to get the result as a table with Fuel tankered, trip fuel, total fuel, and profit. He wanted the results to show the top 5 most profitable results into the first page of the tripkit documents. An empty field was found on the first page where our table would fit perfectly so the goal is to put the results there.

To be able to do this, the results needs to be saved in the database. A new table called `tankering` was created in the database which contains information about how much trip fuel, total fuel, profit, there is at a specific amount of tankered fuel, and which performance calculation it is tied to.

Attempts were made to save fuel tankering data to the database from the class, but it was not the easiest thing to understand. After some discussion with the other developers the problem was better understood.

One class `TankeringImpl` is required that symbolizes a row in the actual table, where the attributes represent the different columns, and another class `ServerTankeringProvider`(including interface) is needed for establishing the connection between the class and the table. In this class implementation of methods on how to save/retrieve data from the table is needed.

When this was done, an object could be created and filled with data from the calculations and then added to a list with tankering objects and then saved to the database.

As mentioned above, the customer only wanted the top 5 most profitable results. This means that no other rows needed to be saved so some basic logic was added to pick the two surrounding rows around the optimal result and save them to the database as well.

When trying to retrieve the data from the table a problem was encountered. The specified row could not be found because the foreign key connecting the performance calculation with the tankering table was being overwritten with a new completely different id. The problem was that every row in every table required a column with a unique id and even though there were (the initial key used two columns to be unique) a new one was being generated. So a new column `pdId` had to be added which contained information about which performance calculation it is tied to. When this was done the fuel tankering data could finally be retrieved/saved and investigations could be made on how to get the results into the trip kit.

After some searching an .xml file `TripkitData` was found which contains information about the appearance of the trip kit. A java class with the same name is used to store all the information that is going to be shown in the trip kit. The first thing that had to be done was to try to access the fuel tankering results from the database to this class and this succeeded.

Retrieving the data from the trip kit was not as straight forward as one might think. A new class `TankeringInfo` (a static one) had to be created that contains the information that is going to be presented into the trip kit. This class will be serialized in order to be able to show the data in the xml file. Once this was done only the formatting remained.

The first thing attempted when trying to get our results visualized on the trip kit was to just write out the column names. When this was done it was attempted to retrieve the actual data from TankeringInfo and then iterate through the tankering list so a table could be presented with the tankering results.

The whole procedure is illustrated in *Figure 6*.

4.2.4 Problems

- All the problems in prototype 1 have been fixed. Some problems were encountered on how to save/collect data from the table in the database but all of this has been fixed so there are no major problems right now, just some minors that will be described in the next prototype.
- It is uncertain if the solution is fast enough so a meeting will be arranged to discuss if further optimization is needed in the next prototype.
- A couple of more enhancements need to be implemented in the next prototype that will be described in the chapter for prototype 3.

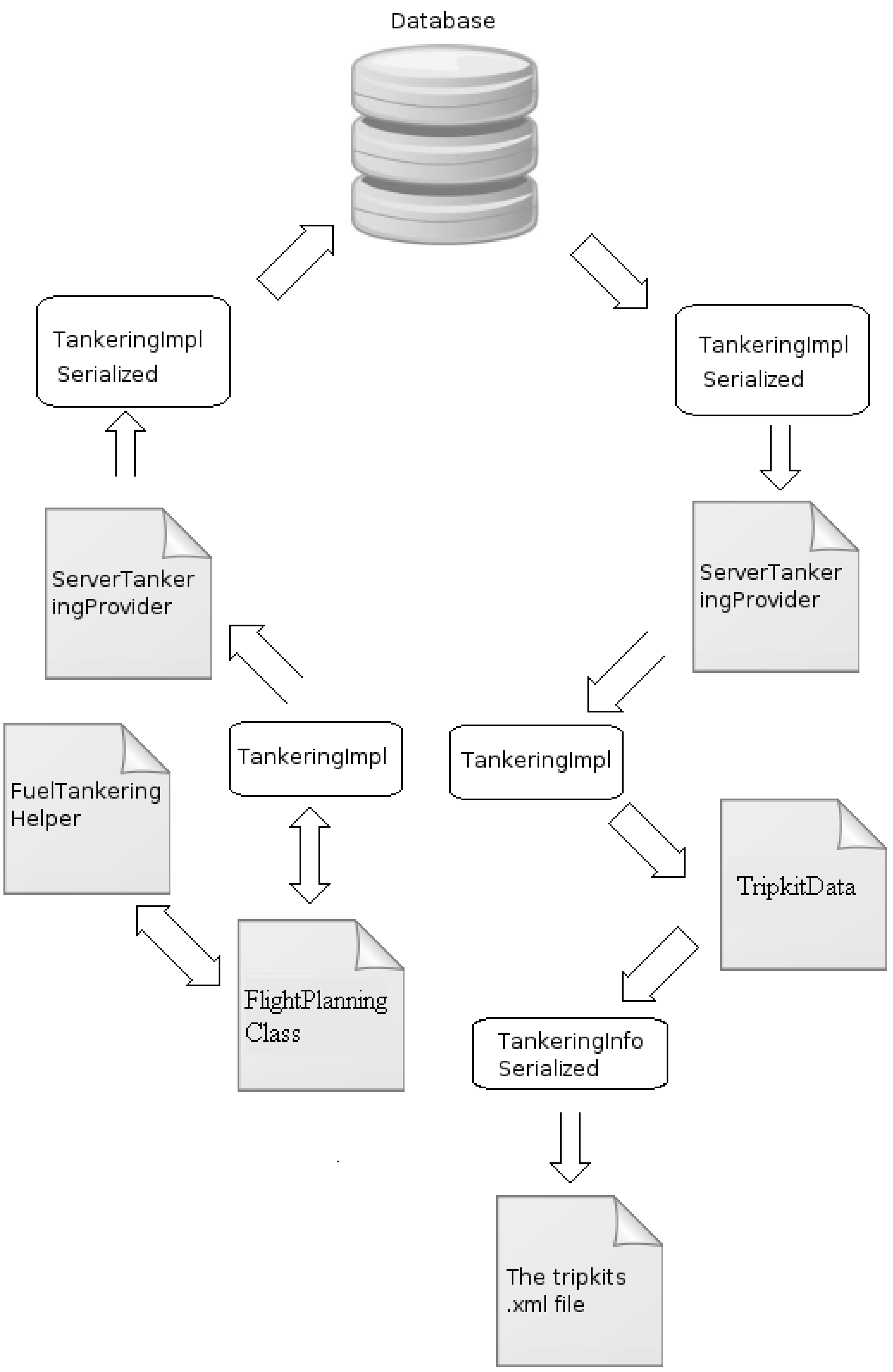


Figure 6 – The procedure between our classes that describes how they are communicating with the system and each other.

4.2.5 Result
When the user has created and activated a flight the fuel tankering results are going to be shown at the first page in the tripkit. The result of this prototype can be seen in the appendix under *Prototype* 2. The result is displayed as a list where the columns are Tankered, Trip, Total, and profit in given currency (highlighted in grey). In this case most profit was gained by filling up as much as possible until it reached the fuel capacity limit.

Some sensitive information has been censored in the trip kit in respect of Flygprestandas privacy but the tankering result is still fully shown.

4.3 Prototype 3
4.3.1 Proposed solution
Most of the major bugs were fixed in the previous prototype and enhancements where made which allowed fuel tankering data to be saved/retrieved and be shown in the trip kit. In the time plan conclusions were made that four prototypes will be the needed, but there has been great progress so it is probable that prototype 3 can be released as the final release. This prototype will focus mostly on further enhancements as most of the bugs are already fixed. Adding new enhancements can of course lead to several more bugs, but there is enough time to solve these within the time span.

Some of the bugs that are going to be fixed:
- When the aircraft has fuel left over from the previous flight, the user will add this to the extra fuel column. However, this is not taken care of (this value is being reset because it is used for tankered fuel calculations) in the fuel tankering calculations so this needs to be fixed.
- It is not possible to load a tripkit from the database the first time when starting the focs server without recalculating it. It has something to do with serialization of one of the tankering objects.
- If there are less than 5 rows in the results, sometimes the optimal rows text font does not get bold.
- If there is a loss instead of profit there are errors when trying to retrieve the loss from the database.

Some of the enhancements that are going to be implemented:
- Make a new column in the tankering results called ”underload”. The underload shows how much weight the airplane can take on before MTOM/MLM is exceeded. This is useful for the pilot so he can make decisions with this number in mind.
- Correctly round the prices in the tripkit for every valid currency. Right know the profits are shown as integers which just cut the decimals.

- Investigate/Discuss whether or not it is needed to have some error windows for the user, e.g. if departure- or arrival-airport does not have fuel prices, or if they are in different currencies, will the user get any error message, if so, how should this be presented?
- Investigate/Discuss whether or not it is needed to optimize the solution further.
- Investigate/Discuss if a button is needed in FOCS that enables/disable fuel tankering.
- When the space for fuel is less than the amount of fuel it increases between iterations, it simply breaks. For this prototype the aim is to make this go higher up to a safe amount that does not exceed any weight or fuel capacity limits.

4.3.2 Bug fixing

Before adding more enhancements it would be wise to fix the remaining bugs.

The first bug fixed was the ”extra fuel” bug described in the previous prototype. This was an easy fix. A temporary mass object was saved called `initialExtraFuel` that stored how much extra fuel the pilot had over from the previous flight. This amount was then added with a tankered amount like *data.setExtraFuel(initialfuel + tankeredAmount)* instead of *data.setExtraFuel(tankeredAmount)*. This solved the problem.

A previously saved trip kit could not load the first time a server has started. This was caused by the fact that the `TankeringInfo` was not included into a configuration file which loads all the different data into the tripkit.

The bug where the optimal rows font did not get bolded was caused by an `IndexOutOfBoundException`. The top 5 optimal rows are added in to an arraylist. When five rows were not available, a row that did not even exist was attempted to be saved. This was fixed by a simple algorithm that returns two values, bottom, and top, which determines the span on which rows that will be saved to the database. It is only the top 5 most optimal results that will get saved in the database.

When retrieving profits as a negative number (loss) from the database errors were encountered when deserializing. This was because the price object is not built to contain negative numbers when deserializing. This led to some problems, but it was fixed easily by making a different approach. Before the profits are saved to the database, there is a check to see if the profit is negative or not. If it is negative, the profit is set to zero. Then when retrieving the results, there is a check to see that if all the rows profit is 0, then there is no profit gained. Then there will just be a line of text in the trip kit: ”No profit can be made by tankering extra fuel” instead of printing out the results. This solution made it easier to implement and certainly easier for the pilot to understand.

At the end of this phase a lot of testing was made to see if there were still any remaining bugs left. A couple of bugs were found where the underload went negative, tankering values got flagged as optimal when it was not (again) and the fuel capacity was exceeded. After a whole afternoon of debugging these bugs were fixed.

At this point, all of the bugs that have been found are fixed.

4.3.3 Enhancements

The rounding problem was easy to fix. Before the price object was returned as an integer, but a method was found inside the price object that could round the price within given precision, so now the price is rounded before it is converted into an integer.

A meeting with the mentor took place to discuss what he thought should happen if the user does not specify fuel prices at destinations etc. Conclusions were made that there should not be any info in the tripkit if there are fuel prices missing at airports, because this is not relevant for the pilot. He suggested it should be as it is right now, but with just another print in the tripkit that says that ”No tankering calculations could be made” that will be printed if anything goes wrong, for example no fuel prices, different currencies, etc.

Discussions also took place about if FOCS should have a button for enabling/disabling fuel tankering. Decisions were made that it should always be enabled because there is actually no reason to turn it off as a user.

4.3.4 Result

The result is shown in the appendix under *Prototype 3*.

There is now a new column ´underload ´ in the trip kit and a more precise maximum tankering value. In this case most profit was gained by filling up as much as possible until it reached the fuel capacity limit.

Due to time limits prototype four is going to be skipped since sufficient goals have been achieved in this prototype.

5 Conclusion

The project started with a question: “Can profit be made by tankering fuel and if so, how much and how can it be calculated?” This part was entirely theoretical and was spent investigating what different variables that had to be considered that would impact the result. A conclusion that it was possible was reached and the implementation of the first prototype was started.

The first prototype was mostly about learning how the system was built and finding where the solution would fit best. Having found this, a simple algorithm was made that wrote all the results into the console. It was slow and unstable but results were accurate.

For prototype two a lot of the code was restructured and was made a lot faster this way. Prototype one left a lot of bugs to correct and there were a lot of discussions on how the results were to be displayed. Having spoken to the software manager and the customer it was concluded that the result should be shown as a table in the tripkit and proceeded to implement this.

Prototype three brought even more bug-fixing and some optimization of the code. But most of the work here was to prevent unhandled errors and making the max tankering amount more precise while still being safe in regards to fuel capacity and MTOM/MLM. Furthermore some data was missing in the tripkit and the table in the database contained redundant information which was also taken care of here.

The time plan specified a fourth prototype but there was no need for another prototype and that it was ready for a final demo so the software department of Flygprestanda was gathered and a fuel tankering presentation was held. At the end of the demo a discussion was started concerning future enhancements and we got a chance to say what could be improved with fuel tankering in FOCS. The manager of Flygprestanda approved of what we had done. Our mentor gave us a clear pass and told us this is something that will really come to good use. In accordance with the goals an accurate fuel tankering module, with no user interaction and code that follows the coding standards of Flygprestanda has been implemented and integrated in a way that offers enhancements to be added easily.

Other than the skipping of prototype four, the work has been conducted exactly as specified in the time plan.

6 Further enhancements

There is a lot of work that still can be further developed regarding the fuel tankering solution. What has been implemented is the ground for the fuel tankering algorithm and there is a lot that still can be developed further. Unfortunately, due to the limited scope for this project there was no time to enhance it further with more features, so we have tried to make the implementation easy to further enhance for the other developers at Flygprestanda.

There are a couple of more variables that would have given slightly more accurate results, such as Carbon Dioxide taxes and terminal charges.

Cases when flying with one or more alternative destinations is not treated, either.

In later versions of FOCS there is going to be implemented a feature that stores data about how long the runways are on airports. The more weight an aircraft has, the longer the landing distance is going to be. There can be a problem when tankering, because when more is tankered the weight of the aircraft gets higher, which will require a longer runway than without tankering. In the worst case, the pilot will not be able to land because the aircraft weighs too much. When flying with an alternate destination this airport also needs to be taken into consideration. This was not implemented simply because FOCS does not have the data for this solution right now. When this data is available there should not be any problem to implement it into this solution.

6.1 Currency conversion

Under the management view in FOCS, where fuel prices at airports can be entered, the user can choose which currency the price is going to be. The problem is that there is no implementation in FOCS which makes it possible to convert a currency into another yet. Because of this, the user is forced to use the same currency at every airport in order to make fuel tankering calculations. There is a lot of work that needs to be done to implement this successfully, probably so much work that this problem could be a thesis on its own.

6.2 CO2 emission tax

With extra weight comes extra fuel consumption. With extra fuel consumption comes increased CO2 emissions. These emissions are tax-based on how large they are and so as they increase with fuel tankering they are relevant to the profit made with it. Although the difference in profit may be small, it is still a feature that other competing software maybe does not account for. This tax varies between countries so there is probably a lot of investigation needed into this one to find out where and how you can retrieve this data.

6.3 Three or more flight legs

Under *What data is required* it is mentioned a way to make extra profit when having multiple flight legs planned. This is something that there was no time to implement but is something that can greatly improve the accuracy of profit made. The problem with this kind of implementation is that it requires the pilot to tanker the exact amount of fuel specified as optimal or else the calculations will be rendered obsolete for the following flights.

6.4 Mask fuel-prices in tripkit

Because fuel-prices are very secret for an airline company it is important to protect this kind of information. The route used and profit gained is enough information to make out hints as to what the fuel price can be, certain information needs to be hidden or encrypted somehow, for example like a price index. With this method the pilot has a table of different indexes that are tied to prices, so in the trip kit there will only be an index which the pilot needs to look up in his table to see the actual profit.

6.5 Optimization

The problem with the speed of our solution was discussed with the system architect. The conclusion was reached that it should be no problem. The calculations may seem slow, but when running on a standalone server the calculations is much faster than if you run the client and server at the same computer and our computer did not meet the recommended amount of memory for running the server.

Although, it was noticed that on longer routes the calculation time for the solution can take longer time than the original calculation (one iteration can take almost 1 second). This is not good and right now there is no good solution. After some investigations into this it was concluded that this has to do with the performance calculations that the algorithm is iterating through, which gives us all the data that we need to do our tankering calculations. There are two options; go deeper into the performance calculation methods and remove parts that is not necessary, or make less iterations.

Making less iterations is not the easiest thing to implement, and it should not make a big difference because the aim is to always make at least five iterations (if possible) so there are five different tankering values in the results. This would only have an effect on bigger airplanes such as Boeing 737, where the MTOM/MLM and fuel capacity is far greater. There was an idea before where it could start iterating at maximum tankering value, and then decrease until it has got five results. This is not possible, because in order to find out the maximum tankering amount, a mean trip fuel gain value is used which finds out on how much on average the trip fuel increases per tankering amount and this value is not accurate before there are a couple of different tankering amounts. It would be possible to make 3-4 tankering values when starting from zero, and then find out the mean trip fuel gain value, then finding out the maximum tankering amount and then go further down from that amount. But this method would not be pretty and would certainly lead to many new bugs.

It was decided not to optimize any further, because it is not within this projects scope and due to time constraints other features were prioritized.

7 Terminology

FOCS – Flygprestandas flight planning software.

NOTAM – Notices To Airmen, messages created and transmitted by government agencies and airport operators to alert pilots of any hazards.

Payload – the carrying capacity of an aircraft including cargo and extra fuel.

DAD – Flygprestanda's database containing information about everything from overflight charges to how much fuel is required at a certain weight/altitude to maintain cruise flight and so on.

MTOM – Maximum Take-Off Mass, the highest mass allowed for an airplane to take off on an airports runway.

ATOM – Actual Take-Off Mass, the actual mass off the airplane when taking off.

MLM – Maximum Landing Mass, the highest mass allowed for an airplane to land on an airports runway.

ALM – Actual Landing Mass, the actual mass off the airplane allowed when landing on an airports runway.

NM – Nautical Miles, unit for measuring flight distance. 1 nautical mile = 1.852 kilometers.

Leg – A route between two airports

Performance Corrections - These are calculations based on the same flight but at different flight levels, therefore the amount of trip fuel burned and the time for the trip are different.

Tripkit – A document that is handed to the pilot that contains all the information for a certain flight. Our results are presented in the tripkit.

Segment – A route is built using Route Segments where a route is comprised of many route segments. These route segments contains performance data about the actual flight.

phpMyAdmin – A tool that is used to maintain the database that is run through the web browser.

8 References

[1] Quarles & Brandy LLP: *Jet Fuel Consortiums*
Retrieved February 2, 2012 from
http://www.quarles.com/jet_fuel_consortiums/

[2] Flygprestanda AB: *Flygprestanda AB Performance Engineering.*
Retrieved February 2, 2012 from
http://www.flygp.se/

[3] Transport Canada: *Theory and Aircraft Performance*
Retrieved February 4, 2012 from
http://www.tc.gc.ca/eng/civilaviation/publications/tp10643-chapter2-theory-203.htm

[4] IATA: *Airport and Air Traffic Control (ATC) Charges.* Retrieved February 13, 2012 from
http://www.iata.org/whatwedo/airport-ans/charges/pages/airport-atc-charges.aspx

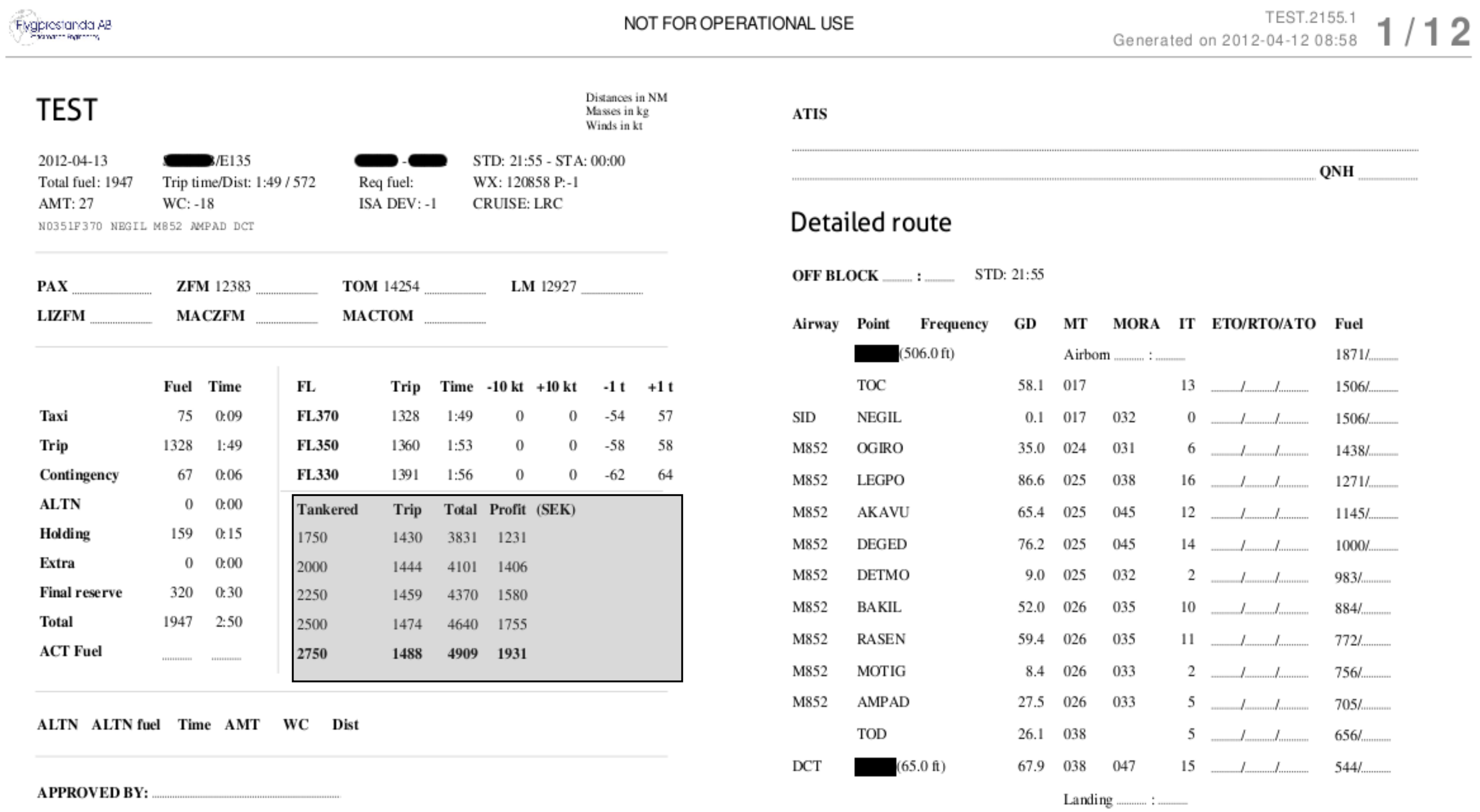
[5] Eurocontrol: *Frequently Asked Questions (FAQ).* Retrieved February 13, 2012 from
http://www.eurocontrol.int/faq/route-charges/

[6] Federal Aviation Administration**:** *Aeronautical Information Manual.*
Retrieved February 2, 2012 from
http://www.faa.gov/air_traffic/publications/atpubs/aim/

[7] PPrune: *Discussions about fuel tankering formulas*
Retrieved February 8, 2012 from
http://www.pprune.org/tech-log/148577-fuel-tankering-formula.html

9 Appendix

9.1 Prototype 2



Flygprestanda AB

NOT FOR OPERATIONAL USE

TEST.2155.1
Generated on 2012-04-12 08:58 **1 / 12**

TEST

Distances in NM
Masses in kg
Winds in kt

2012-04-13	[redacted]/E135	[redacted]-[redacted]	STD: 21:55 - STA: 00:00
Total fuel: 1947	Trip time/Dist: 1:49 / 572	Req fuel:	WX: 120858 P:-1
AMT: 27	WC: -18	ISA DEV: -1	CRUISE: LRC

N0351F370 NEGIL M852 AMPAD DCT

PAX	**ZFM** 12383	**TOM** 14254	**LM** 12927 ....
LIZFM	**MACZFM**	**MACTOM**	

	Fuel	Time
Taxi	75	0:09
Trip	1328	1:49
Contingency	67	0:06
ALTN	0	0:00
Holding	159	0:15
Extra	0	0:00
Final reserve	320	0:30
Total	1947	2:50
ACT Fuel	....	....

FL	Trip	Time	-10 kt	+10 kt	-1 t	+1 t
FL370	1328	1:49	0	0	-54	57
FL350	1360	1:53	0	0	-58	58
FL330	1391	1:56	0	0	-62	64

Tankered	Trip	Total	Profit (SEK)
1750	1430	3831	1231
2000	1444	4101	1406
2250	1459	4370	1580
2500	1474	4640	1755
2750	**1488**	**4909**	**1931**

ALTN	ALTN fuel	Time	AMT	WC	Dist

APPROVED BY: ....

ATIS

.... **QNH** ....

Detailed route

OFF BLOCK ..... : ..... STD: 21:55

Airway	Point	Frequency	GD	MT	MORA	IT	ETO/RTO/ATO	Fuel
	[redacted](506.0 ft)			Airbom ..... :				1871/.....
	TOC		58.1	017		13	...../...../.....	1506/.....
SID	NEGIL		0.1	017	032	0	...../...../.....	1506/.....
M852	OGIRO		35.0	024	031	6	...../...../.....	1438/.....
M852	LEGPO		86.6	025	038	16	...../...../.....	1271/.....
M852	AKAVU		65.4	025	045	12	...../...../.....	1145/.....
M852	DEGED		76.2	025	045	14	...../...../.....	1000/.....
M852	DETMO		9.0	025	032	2	...../...../.....	983/.....
M852	BAKIL		52.0	026	035	10	...../...../.....	884/.....
M852	RASEN		59.4	026	035	11	...../...../.....	772/.....
M852	MOTIG		8.4	026	033	2	...../...../.....	756/.....
M852	AMPAD		27.5	026	033	5	...../...../.....	705/.....
	TOD		26.1	038		5	...../...../.....	656/.....
DCT	[redacted](65.0 ft)		67.9	038	047	15	...../...../.....	544/.....
				Landing ..... :				

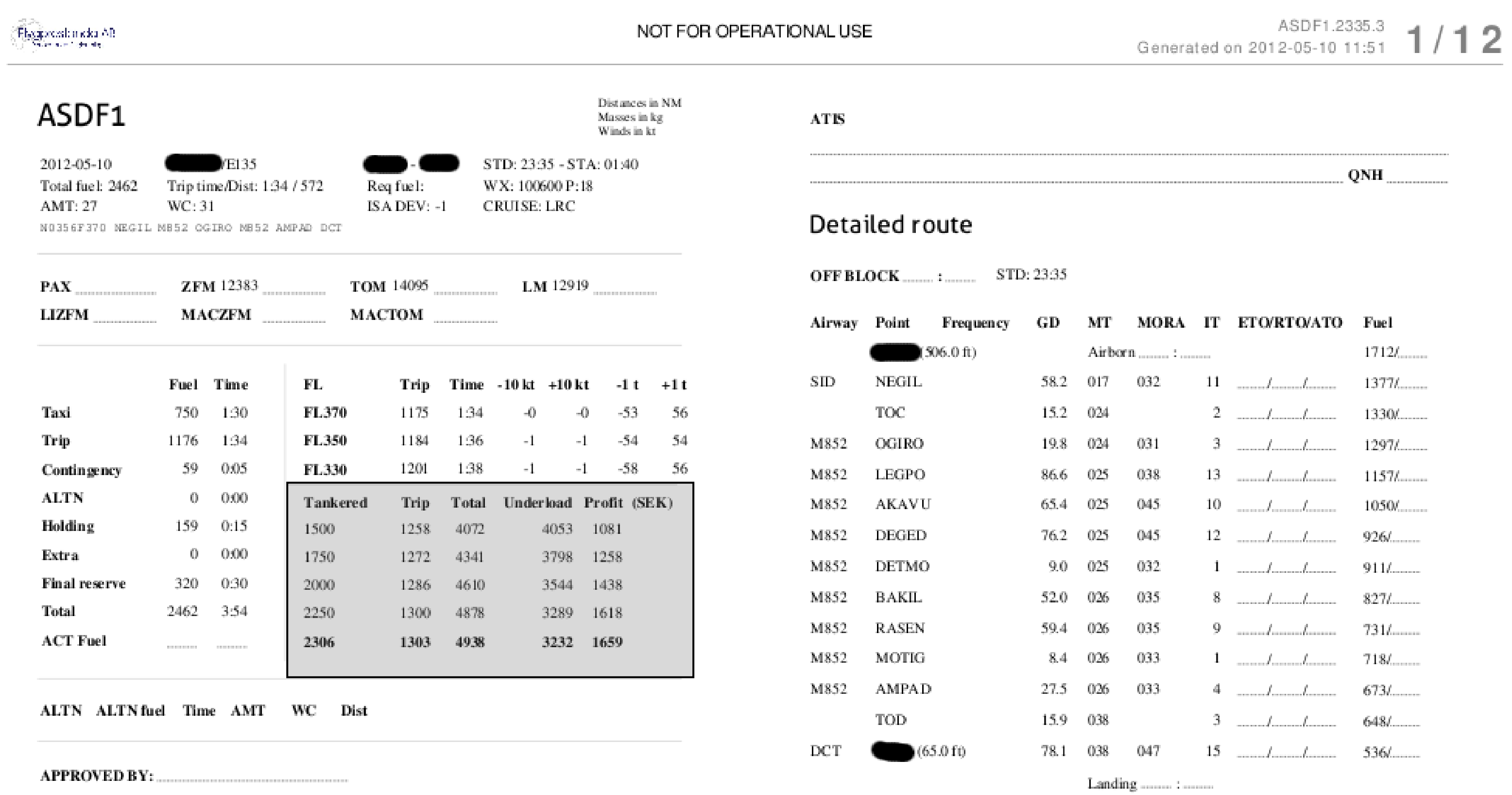
ON BLOCK ..... : ..... STA: 00:00

ATIS

.... **QNH** ....

Figure 7 – Our results for prototype 2, presented in the tripkit. Our solution is presented in the highlighted area.

9.2 Prototype 3



NOT FOR OPERATIONAL USE

ASDF1.2335.3
Generated on 2012-05-10 11:51 **1 / 12**

ASDF1

Distances in NM
Masses in kg
Winds in kt

2012-05-10	[redacted]/E135	[redacted] - [redacted]	STD: 23:35 - STA: 01:40
Total fuel: 2462	Trip time/Dist: 1:34 / 572	Req fuel:	WX: 100600 P:18
AMT: 27	WC: 31	ISA DEV: -1	CRUISE: LRC

N0356F370 NEGIL M852 OGIRO M852 AMPAD DCT

PAX .... **ZFM** 12383 .... **TOM** 14095 .... **LM** 12919 ....
LIZFM .... **MACZFM** .... **MACTOM** ....

	Fuel	Time
Taxi	750	1:30
Trip	1176	1:34
Contingency	59	0:05
ALTN	0	0:00
Holding	159	0:15
Extra	0	0:00
Final reserve	320	0:30
Total	2462	3:54
ACT Fuel	....	....

FL	Trip	Time	-10 kt	+10 kt	-1 t	+1 t
FL370	1175	1:34	-0	-0	-53	56
FL350	1184	1:36	-1	-1	-54	54
FL330	1201	1:38	-1	-1	-58	56

Tankered	Trip	Total	Underload	Profit (SEK)
1500	1258	4072	4053	1081
1750	1272	4341	3798	1258
2000	1286	4610	3544	1438
2250	1300	4878	3289	1618
2306	**1303**	**4938**	**3232**	**1659**

ALTN **ALTN fuel** **Time** **AMT** **WC** **Dist**

APPROVED BY: ....

ATIS

.... **QNH** ....

Detailed route

OFF BLOCK .... : .... STD: 23:35

Airway	Point	Frequency	GD	MT	MORA	IT	ETO/RTO/ATO	Fuel
	[redacted](506.0 ft)			Airborn .... :				1712/.........
SID	NEGIL		58.2	017	032	11	......../......../........	1377/.........
	TOC		15.2	024		2	......../......../........	1330/.........
M852	OGIRO		19.8	024	031	3	......../......../........	1297/.........
M852	LEGPO		86.6	025	038	13	......../......../........	1157/.........
M852	AKAVU		65.4	025	045	10	......../......../........	1050/.........
M852	DEGED		76.2	025	045	12	......../......../........	926/.........
M852	DETMO		9.0	025	032	1	......../......../........	911/.........
M852	BAKIL		52.0	026	035	8	......../......../........	827/.........
M852	RASEN		59.4	026	035	9	......../......../........	731/.........
M852	MOTIG		8.4	026	033	1	......../......../........	718/.........
M852	AMPAD		27.5	026	033	4	......../......../........	673/.........
	TOD		15.9	038		3	......../......../........	648/.........
DCT	[redacted](65.0 ft)		78.1	038	047	15	......../......../........	536/.........
				Landing .... :				

ON BLOCK .... : .... STA: 01:40

ATIS

.... **QNH** ....

Figure 8 – Our results for prototype 3, presented in the tripkit. Our solution is presented in the highlighted area.