An Object-Oriented Approach to a Scenario-Based System Dynamics Fleet Model

Gilbert Tay
Technical University of Munich
Research Associate, Institute of Aircraft Design
85748 Garching, Germany
Gilbert.Tay@tum.de

Raoul L. Rothfeld
Bauhaus Luftfahrt e.V.
Research Associate, Economics and Transportation
82024 Taufkirchen, Germany

ABSTRACT

Expanding the current capabilities of a scenario-based system dynamics fleet model is paramount in order to incorporate wider ranging boundary conditions for the assessment of the impact of future aircraft technologies at a global aircraft fleet level. These expanded capabilities are necessary in order to better assess the impact of novel aircraft technologies and ambitious environmental policies in order to mitigate the environmental impact of the civil air transport industry. However, this further expansion of capabilities would not be possible without a paradigm shift in the way the current fleet model is implemented, as the increased complexity would make the fleet modeling an extremely inefficient and daunting task. An object-oriented approach to address the current limitations of the current system dynamics fleet model as well as the incorporation of future expanded capabilities of the new fleet model are presented.

KEYWORDS: *system dynamics, global fleet, aircraft fleet modeling, object-oriented*

NOMENCLATURE

Variables:
added_acft_data - creation of added aircraft data
Adjust_data - formats the result data
Adjust_data_all - contains all operations for adjusting the simulated data
Calc_FB_trend - calculates the fuel burn trend for the current year
Calc_Freq_from_BH - calculates the frequency from the block utilization hours
Calculate_Total_Fuel_Consumption - calculates the total fuel consumption
createFBMatrix - creates an aircraft Fuel Burn Matrix
Data_Sync - synchronizes the data from the FCCT into the main script operations
FAP - Fleet Assignment Problem
FAP_Call - calls the actions to solve the FAP
FCCT - Fleet Consumption Calculation Tool
FCCT_Call - calls the Fleet Consumption Calculation Tool
fmincon_Function - finds a local minimum of four defined algorithmic values
get_FleetData - copies the fleet variables of the excel input files into the MATLAB script files
getBADAPosition - declares and positions variables using the Base of Aircraft Data
GetFBMatrix - summons the Fuel Burn Matrix
Global_Mission_Calculator - calculates data for all global flight missions
LoadAC_Clusters - loads the aircraft clusters
New_acft_calc - calculates number and compositon of new aircraft
New_acft_calc_no_pc - as in New_acft_calc but without considering the production capacity
New_acft_calc_pc - as in New_acft_calc while considering the production capacity
New_acft_calc_pc_sa_exceeded - as in New_acft_calc but with the production capacity exceeded for single aisle aircraft
New_acft_calc_pc_sa_exceeded_by_newclusters - as in New_acft_calc but with the production capacity exceeded for single aisle aircraft of the new defined aircraft clusters

New_acft_calc_pc_sata_exceeded - as in New_acft_calc but with the production capacity exceeded for single and twin aisle aircraft
New_acft_calc_pc_ta_exceeded - as in New_acft_calc but with the production capacity exceeded for the twin aisle aircrafts
New_acft_calc_pc_ta_exceeded_by_newclusters - as in New_acft_calc but with the production capacity exceeded for twin aisle aircraft of the newly defined clusters
Pre_calc - prepares data for the main script
Production_capacity - calculates fleet production capacity
Quotient_calc - calculates the quotient for the different used clusters
Result_files_producer - produces the final result files
SA_limit - single aisle limit reached (==1)
Share_new_acft_2 - produces a ranking for the newly produced aircraft clusters
Share_new_acft_TA - produces a ranking for the new produced twin aisle aircraft clusters
Share_oc_acft_2 - produces a ranking for the old aircraft clusters
Single_Flight_Mission - calculates a single flight mission for a particular aircraft
TA_limit - Twin aisle limit reached (==1)
Transform_z_FleetIS_Route_trend - copies all "transformed" data of the previous and current total fleet, so it can be presented to the user in a compact way

1 INTRODUCTION

The International Air Transport Association (IATA) has defined a set of ambitious environmental goals to reduce carbon emissions from the aviation industry at a global level. These include a 1.5%-improvement in average fuel efficiency per year between 2009 and 2020, carbon-neutral growth from 2020 onwards and a 50%-reduction in net CO_2 emissions by 2050 relative to 2005 levels [1]. Various other governmental institutions like the European Commission [2] and aviation associations like the Air Transport Action Group (ATAG) have also adopted these milestones [3]. IATA furthermore defined a four-pillar strategy comprising measures referring to the use of improved technology, the implementation of effective operations, the creation of an efficient infrastructure and the introduction of market-based economic measures. The context of this requirement to evaluate the civil air transport system from a global aircraft fleet perspective was the initial motivation for a scenario-based system-dynamics fleet model also known as the Fleet System Dynamics Model (FSDM) that incorporates the Fleet Performance Calculation Tool (FPCT) [4]. Figure 1 shows the aircraft technology assessment technique at fleet-wide level (ATTESST) [4].

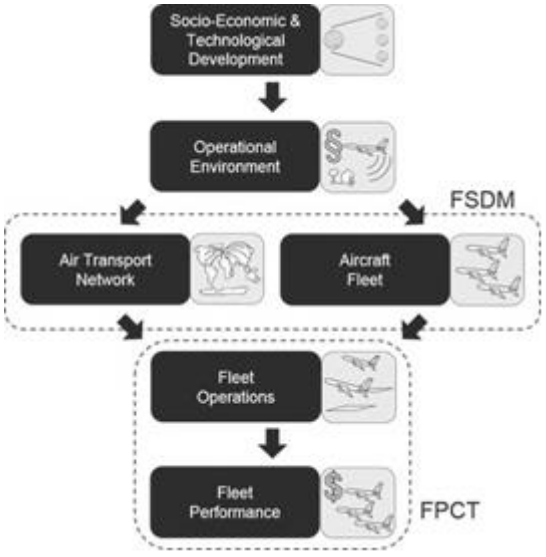

Figure 1: Aircraft technology assessment technique at fleet-wide level (ATTESST) [4]

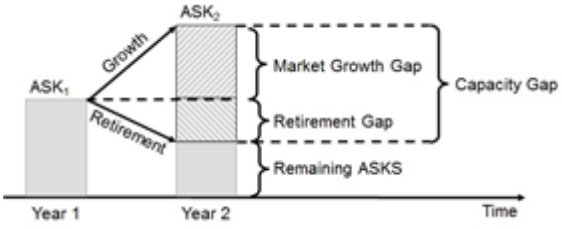

Figure 2: "Macro approach to fleet planning" by Clark [5]

The FSDM is based on the "macro approach to fleet planning" described by Clark [5]. Figure 2 visualizes the computation of the capacity gap required in a given year taking into account market growth, fleet retirement and present market volume (ASK_1). The sum of all available seat kilometers

(ASK) globally is interpreted as the overall transport capacity available to the world market[1]. By using scenario-specific market growth factors as input parameters for the FSDM, the global fleet development for the evaluation of the environmental goals for the global aviation industry can be modeled.

Modeling of the global air transport network and aircraft fleet is also performed in the FSDM. In order to reduce the complexity of the global fleet and air transport network, all of the global air transport routes are aggregated and simplified to form 21 air transport route groups. Figure 3 shows schematically the FSDM air transportation network. [4]

Figure 3: Schematic display of the FSDM air transport network [4]. Arrows represent the 21 intra- and intercontinental route groups [4].

ID	Name	Representative Aircraft
1	Long-Range Combi	Boeing MD-11
2	Long-Range Heavy	Boeing 747-400
3	Mid-Range Freighter	Boeing 767-300F
4	Jet Commuter	Embraer 190
5	Long-Range Freighter	Boeing 747-400F
6	Turboprop Commuter	ATR 72-500
7	Mid-Range Passenger	Boeing 767-300
8	Long-Range Passenger	Boeing 777-200
9	Short/Mid-Range Passenger	Airbus A320-200

Figure 4: FSDM aircraft clusters [4]

The initial global aircraft fleet of 198 different aircraft types from 2008 [6] is clustered in FSDM to form nine discrete aircraft categories using a k-medoids clustering algorithm according to "preselected aircraft parameters, including aircraft type-specific seat and cargo capacity, typical distance flown, and type of propulsion". [4] Figure 4 shows the nine aircraft clusters of the initial fleet in 2008. [7]

The scenario-based system-dynamics fleet model has been successfully implemented using the logic of an algorithm to minimize fuel burn for the fleet assignment problem to evaluate the environmental impact of new aircraft concepts [7].

Further enhancements to the model however would require a new approach to address the current limitations of the system-dynamics fleet model, as further extensions to the current model would result in an exponential increase in complexity. This paper therefore proposes an object-oriented approach to the current system-dynamics fleet model as well further enhancements planned.

2 PROBLEM CHARACTERIZATION

Due to the complexity and the multitude of variables in the fleet model by Randt [4], any further expansion in program capabilities would lead to an exponential increase in complexity and development time. Figure 5 shows the various functions in the non-object-oriented structure of the fleet model by Randt.

An excerpt of the flowchart of the, exemplarily-taken, script "New_acft_calc" is shown in Figure 6 which elucidates the increase in complexity with any further expansion in capabilities. The complex encapsulation of the various functions within each other highlights the imperative necessity of a new object-oriented paradigm to the scenario-based system-dynamics fleet modeling approach.

[1] Only the seat transport capacity is described here in order to reduce the scope of this paper. The freight transport capacity is calculated accordingly.

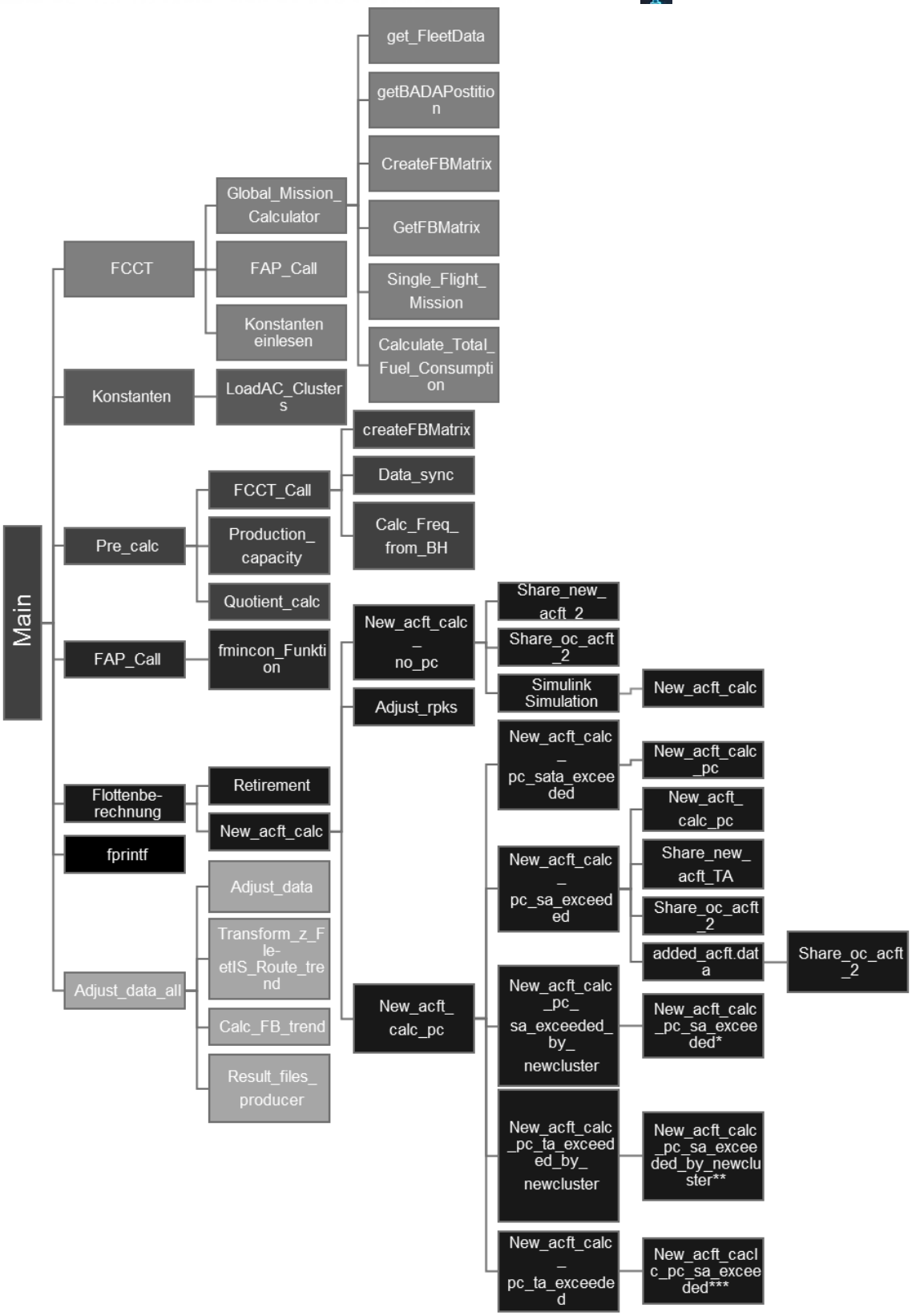

Figure 5: Basic structure of scripts in the non-object-oriented fleet model

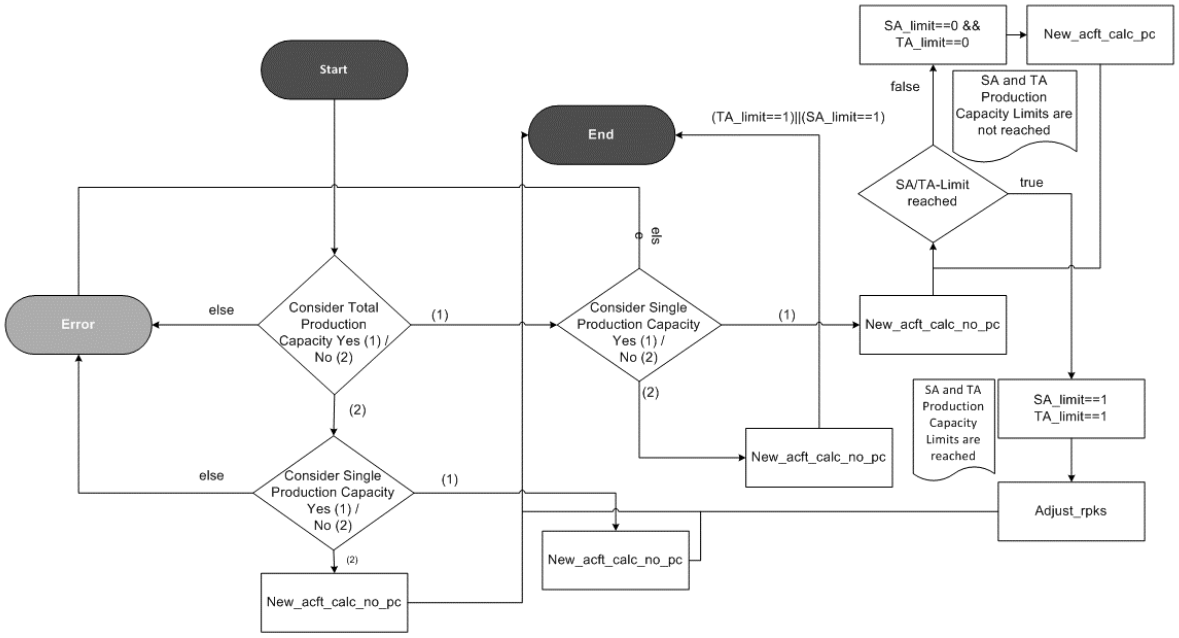

Figure 6: Flowchart of the script "New_acft_calc"

The necessity for an object-oriented approach became even more evident with the multitude of variables within the script-based environment, as variables cannot be defined with a specific, finite scope. Despite adhering to naming convention, keeping an oversight over all variables and their current state proved to be very labor-intensive – especially, when working in a joint development team with members from different institutes and backgrounds. Further, variables in the script-based environment where to be accessed directly without defined get or set functions, requiring each developer to – not only be aware of all variables and their purpose – but also about their underlying data structure. Lastly, the vast number of required helper function had to be stored in subscripts, leading to an inscrutable accumulation of scripts and subscripts, as shown in Figure 5. To remedy these shortcomings, an object-oriented fleet model is to be developed.

3 METHODOLOGY

3.1 Program Structure

The developed model operates with a multitude of variables and conducted operations, all which build a very complex program. To fully comprehend the extent of the object-oriented Fleet Development Model (oo-FDM), the structure of the program has to be categorized in a logical way. The program is therefore divided into six time- and action-dependent phases. These are successively connected by their interfaces, which are used to pass data from a hierarchically higher to a hierarchically lower phase. This relationship is illustrated in Figure 7.
At its highest level, the oo-FDM can be divided into a macro and a micro framework. As illustrated in Figure 7, it comprises of an initiation, a simulation, and an output phase.

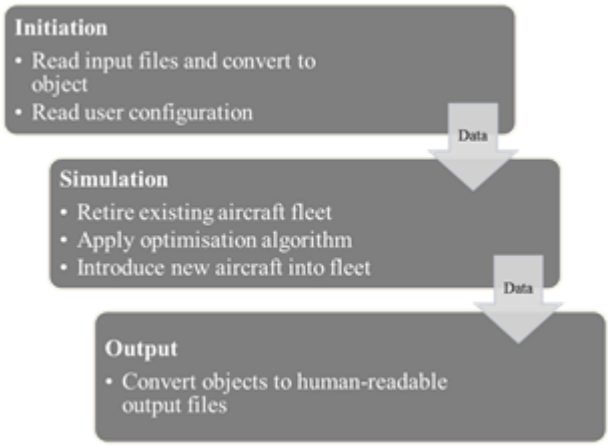

Figure 7: Macro phase structure of the object-oriented program.

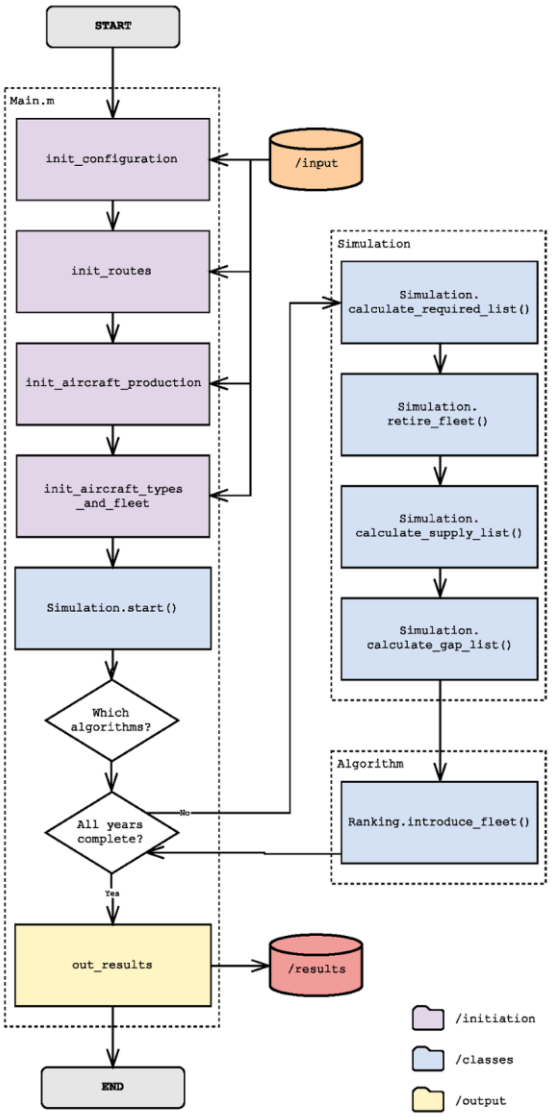

Figure 8: Flowchart of the object-oriented fleet model implementation

In order to provide all information, which is required for the simulation, relevant aircraft data has to be gathered. Additionally, the analyzed aircraft have to be classified into clusters or aircraft types. Aircraft types are aircraft classes which have similar characteristics and are represented by a typical or representative aircraft. This information is then used to make plausible assumptions and calculations such as revenue passage kilometers (RPK), available seat kilometers (ASK) or production capacities. Among others, the required information include initial fleet sizes, average flight frequencies, average fuel burn and direct operating costs per trip, and average seats per year for each aircraft type. These input files are being read during the initiation phase and converted to objects for further processing.

As the next step, the created objects are used within the simulation phase of the program. This phase, in which all calculations and simulations of the different variables are conducted, is the centerpiece of the program. Furthermore, this phase can be divided into three subparts: (1) retirement of old aircraft within the existing (or initial) fleet, (2) application of an optimization algorithm for future fleet introductions (e.g. fuel burn minimization), and (3) introduction of new aircraft into the fleet – as illustrated in Figure 8.

After the simulation has completed, i.e. run for the user-specified duration, the output phase commences. During the simulation, all intermediate results are stored within the appropriate objects. The output phase retrieves these intermediate results, combines and aggregates them, and converts

the results into human-readable tables, which are written to disk. Afterwards, the result files can be evaluated and plotted for further analysis.

Figure 8 shows the information flow as well as the micro phase structure of the object-oriented fleet model. This figure also describes the relation between the phases and remaining scripts. The starting and ending point are represented by two round-edged boxes marked as "Start" and "End".

The program's entry point is the Main.m script from which all three micro phases are being initiated. Firstly, the initiation scripts are executed, reading in input files from the input folder and converting them into objects. The files to be read comprise the simulation configuration, route data, information about yearly aircraft production numbers, and – most importantly – data about the various aircraft types to be included in the simulation run. After all objects have been initialized, the simulation can be commenced via its start() function. The optimization algorithm and variable is set by the user via the simulation configuration file. While there is currently only one optimization algorithm implemented, the user can already chose whether to introduce new aircraft under the premises of minimizing fuel burn or direct operating costs.

Within the simulation, the current air traffic market size is being calculated and grown or shrunk, according to user-defined market forecasts, for the next year. The previous year's global aircraft fleet is being aged by one year, with some aircraft retiring based on the aircraft's age. The remaining fleet's productivity in terms of available seat and ton kilometers (ASK or ATK) is aggregated and compared to the desired available seat and ton kilometers required for the coming simulation year. The difference between the desired and supplied available seat and ton kilometers is stored as the so-called ASK- and ATK-gap list. Based on the chosen optimization algorithm/variable and aircraft production limits, new aircraft are introduced into the global fleet, until either the production capacities are diminished or the ASK/ATK gap has been filled. This process is repeated until the user-defined, final simulation year has been reached and completed. Afterwards, the output script is started and writes the result files into the results folder – concluding the fleet model program flow.

3.2 Application of an Object-Oriented Philosophy

As described in the structure of the program, the Fleet Development Model (FDM) is put together by a large number of variables and objects, which have a certain association with each other. Furthermore, the fleet comprises a multitude of aircrafts-types, which also increases the complexity of the model. In order to achieve a solution for this problem and reduce complexity, an object oriented approach was implemented.

The main systems, such as aircraft and pre-defined route groups, are described as classes. Furthermore, the used variables in the systems can be represented as objects of a class. Every object is an instance of a class and can be described as a set of properties and methods. Object can communicate with each other via class methods, which achieves a desired association of working-variables. Property and method names are unique, which prevents unnecessary programming mistakes during the implementation process. The nature of the object-oriented approach allows more efficient incorporation of associated work, which leads to a compact program that is easily comprehensible.

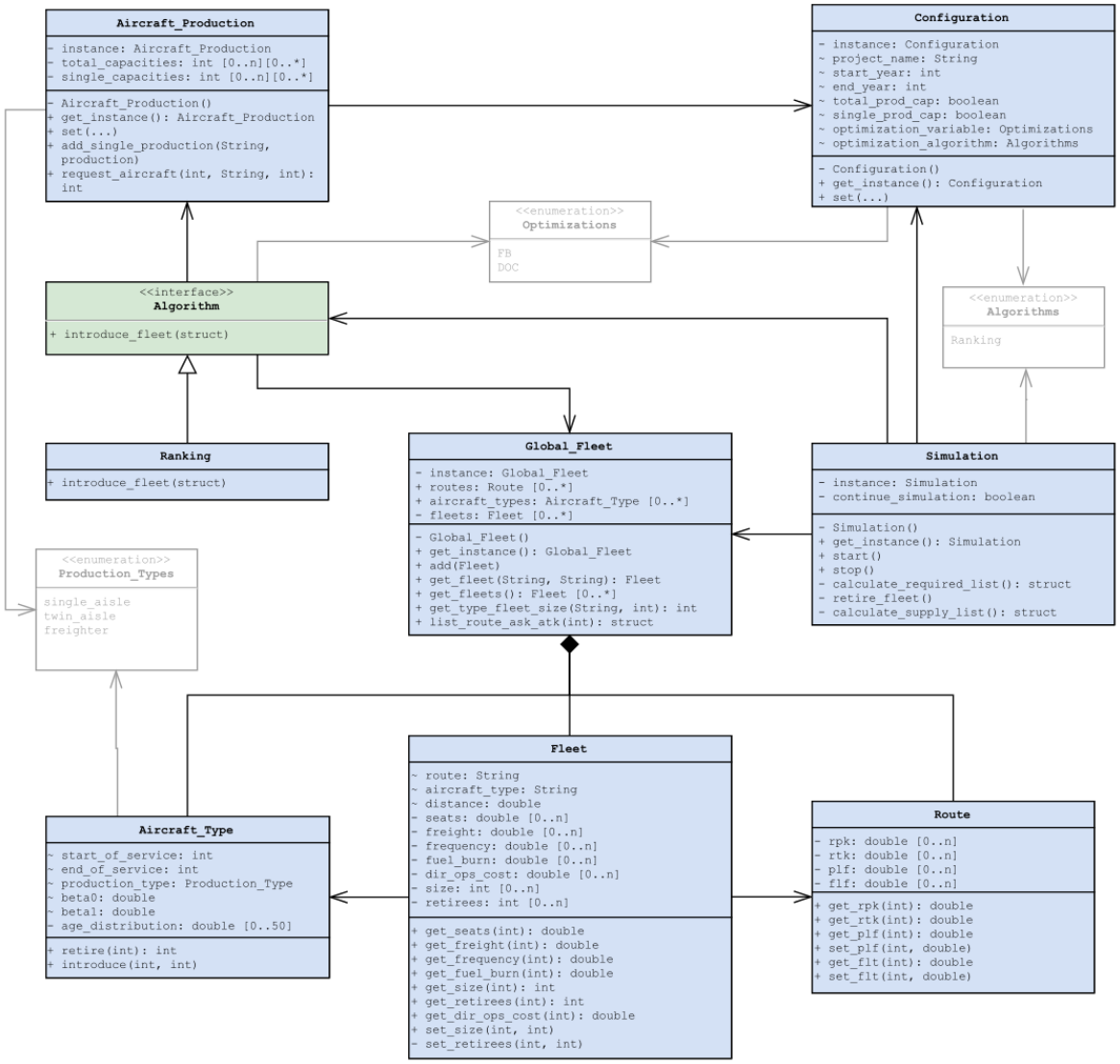

Figure 9: UML diagram of the object-oriented fleet model implementation

The oo-FDM program consists of eight classes and one interface, as illustrated in Figure 9. The Route class is used to spawn 21 objects, which refer to defined route groups. The 21 routes are formed from the combinations of intra- and intercontinental combinations. There are six different continental declarations. These declarations are, AS for Asia, EU for Europe, ME for Middle East, AF for Africa, NA for North America and LA for Latin America. Each route object then contains growth factors for revenue passenger and ton kilometers and yearly average seat and freight load factors. The Aircraft_Type class is used to hold information specific to each aircraft type, such as its date of introduction or beta-values which describe the curve with which aircraft of that type retire (via a probability of survival curve). The Fleet class is used to represent a specific aircraft fleet of a particular aircraft type, operating on a particular route. Additionally, the Fleet class then stores additional information, such as average flying distances, number of seats or freight capacity, and the number of aircraft for each year. All Fleet objects are gathered and are only accessible via the Global_Fleet class, representing the sum of all aircraft types on all routes. The Global_Fleet class acts as a controller between the simulation and optimization algorithm(s) and the underlying model (i.e. aircraft types, routes, and fleets).

4 IMPLEMENTATION

4.1 Object-Oriented Program Structure

The implementation process of the object oriented Fleet Development Program, is shown in Figures 8 and 9. During the implementation process, a connection between the defined phases and used functions is achieved. To understand the iteration process of the program it is essential to describe the functions and dependencies. The main goals are a reduction in complexity and an increase in understandability for users to operate within a defined time and action scale. Furthermore, the program is easier to evolve and adapt to emerging needs and changes.
The process flow of the main script is illustrated in Figure 4. The process is being started by the "Start" block and ended by its equal "End" block. The information from the user input function block is fed into the initiation blocks. This is where the directional paths of all input variables are listed. Important variables such as the simulation end year and the production capacity are generated. The initiation blocks produce the variables that will be used in the simulation process and initializes all the classes shown in Figure 9 required for simulation. The main function of the initiation blocks can be described in four steps. Initialization of the (1) configuration objects, (2) route group objects, (3) aircraft production objects, (4) global aircraft and fleet objects. The simulation blocks generate the global fleet according to the optimization algorithm used, for each simulated year. The optimization algorithm implemented in this instance is the fuel burn-minimizing algorithm. The fuel burn-minimizing algorithm solves a fleet assignment problem according to the logic of minimizing fuel burn, which preferentially allocates the most fuel-efficient aircraft type to the fleet. In this function, the optimum output fleet of the first simulation year is being generated. To solve the fleet assignment problem the integrated "fmincon" MATLAB function is being used, which finds a local minimum of four algorithmic values. This simulation function, which solves the FAP, also wipes out all irrelevant variables after completing the fmincon simulation. For example, all aircrafts that cannot fly a specific route are eliminated. The simulation blocks operate on data from the previously generated variables. These blocks constitute an iterative process, in order for the fleet to "age" accordingly and updates the relevant properties according to the boundary conditions. Finally, the script out_results delivers the output and clears all unnecessary variables, so that the final results are readable and easier to use. The initiation phase, as well as the out_results function (result phase) create an interface between the program and the operator.
As can be seen, the main goals are achieved by the described object oriented implementation. The complexity of the program is reduced by operating with classes that have a defined number of objects. Additionally, the structure allows users to understand its operations and minimize the relevant output variables (adjustment of the result). Flexibility in code adaptation according to various needs is also guaranteed.

4.2 Implementation of Future Capabilities

Based on the object-oriented structure of the fleet model, it is possible to further enhance the model with additional and more detailed capabilities. Among others, it is planned to enhance the model with the following features:

- Introduce the possibility to retrofit existing aircraft within the global fleet
- Introduce inverse simulation mode, where a specific CO_2-reduction has to be met in a specific year and the simulation provides required steps to achieve set goal
- Elaborate the distance calculations per route (use a distance-flight distribution instead of an average distance)
- Introduce airport types and distinguish differently sized airports, including their capacity constraints
- Introduce airline types and distinguish different airline business models
- Introduce market dynamics with competition between airports, airlines, and aircraft manufacturers
- Introduce a fleet model application programming interface (API), as to allow its connection with other system dynamics models
- Automated aircraft live cycle cost calculations (currently performed externally)

Additionally, it is also planned to enhance the program itself by providing a graphical user interface (GUI), in addition to the current command-line interface, and to enhance the program's fault tolerance as to allow non-expert usage.

5 RESULTS

5.1 Comparison of both Program Structures

The comparison between the two MATLAB© programs, the new created object oriented Fleet Development Model (oo-FDM) and the existent scenario based Fleet Development Model (FDM), reveal many similarities as well as differences. Both share the same logical structure and operate on similar methods. If the optimizer algorithm and boundary conditions are identically implemented, the results of both programs is the same. This was also used in the verification process.
Despite their similarities, the two programs also have a few differences, especially regarding the implementation process. In the object-oriented program, the input data is well organized into classes, so that every variable can be referenced and accessed directly during the simulation process. The main advantage of the object-oriented approach is that the code can be adapted much more easily compared to the non-object oriented approach. Because of the inherent structure and modularity of object oriented programming, the code can be modified easily by adjusting a certain variable. This is possible due to the iteration steps being based on the number of objects of a class. As an example, if the user wants to increase or decrease the 21 objects of the Route class in any way, this can be achieved by adding or subtracting objects to those classes, without having any further adjustments to the previous or following code. Furthermore, the translation of the FDM into the oo-FDM has led to an improved overall simulation performance. The accomplished structure also allows someone, who is not highly versed in the intricacies of the FSDM, to easily understand the steps and operations involved in the simulation process.

5.2 Expanded Capabilities

Using Randt's [4] script-based fleet model as a starting point, the newly developed object-oriented fleet development model has already been enhanced by numerous capabilities, on top of the transition to an object-oriented structure.
A major enhancement in usability brought the introduction of human-readable input and output files. While previously, all files were stored in a MATLAB format, they are now available and editable as comma-separated value files or EXCEL spreadsheets. Further, the user is repeatedly updated about the program's state during execution with the introduction of a logging system. A rudimentary implementation of aircraft manufacturer's production capabilities has been elaborated to allow each aircraft type to be assigned a specific yearly production limit, as well as a global limit for the introduction of single-aisle, twin-aisle, and freighter aircraft. Lastly, the previous fleet introduction, aimed at minimizing global fuel burn, has been expanded with the option for introduction new aircraft based on minimizing global direct operating costs.

6 CONCLUSION AND FUTURE OUTLOOK

As described in the previous sections, this scientific work describes the procedure of creating an object oriented, scenario based, system dynamics fleet model, called the object oriented Fleet Development Model (oo-FDM). This model is implemented as a MATLAB program, that can deliver a variety of output values such as the global fleet size, their age distribution and parameters like the revenue per seat kilometres (RPK), that help to understand the worldwide fleet evolution. The main goal is developing a program where the variables used would be introduced in a logical and structured way, with the purpose of reducing complexity so that future extensions would be easy to apply. Additionally the program has to be implemented in such a way, so that it can be easily adapted to a larger or lesser number of objects and object properties. These goals were achieved by the above described phase structure and program implementation.
For the above reasons the developed program has great simulation potential that can be fully exploited in future versions. The program can also be modified to include cost calculation and prognosis in the simulation process by including parameters like maintenance hours and airport fees for ground services. Additionally, boundary conditions such as the capacity of major airports can be

included in the simulation. Major airports struggle to keep up with the growing number of flight attendances due to political, economic and ecological reasons. By accounting for these boundary conditions, assumptions about future fleet development can be made with greater accuracy. In the future, where new aircraft-types may emerge, that are not sufficiently described by the current cluster composition, the need for new clusters may arise. Additionally new route group objects could be created. For example, the region of Asia (AS) could be further divided into its geographical regions like South, West, Southeast Asia and East Asia. This would further improve the model, taking into account the operational diversity of this diverse continent and at the same time maintain an appropriate level of complexity.

REFERENCES

1. International Air Transport Association (IATA) Technology Roadmap, 4th Edition, June 2013;
2. European Union, "Flightpath 2050 – Europe's vision for aviation: Maintaining global leadership and serving society's needs," Luxembourg, 2011;
3. Air Transport Action Group (ATAG), "The right flightpath to reduce aviation emissions," Durban, South Africa, 2011;
4. Randt, N. P., "Foundations of a Technology Assessment Technique Using a Scenario-Based Fleet System Dynamics Model", *Proceedings of the 13th AIAA Aviation Technology, Integration and Operations (ATIO) conference*, 2013;
5. Clark, P., Buying the big jets: Fleet planning for airlines, 2nd ed, Ashgate Pub., Aldershot, Hampshire, England, Burlington, VT, 2007;
6. OAG Aviation Solutions, Official Airline Guide Flight Schedules Database November 2007 – October 2008, UBM Aviation Worldwide Ltd., 2008;
7. Randt, N. P., "Aircraft Technology Assessment Using Fleet-Level Metrics", *Ph. D. Thesis*, Technische Universität München, Munich, 2016;