BOT FOR CONTINUING AIRWORTHINESS:

PROOF OF CONCEPT FOR PASSENGER CARRYING DRONES

UNIVERSITY OF TWENTE.

27th of August 2021

J.K. Wieringa (Joran) – s2173743

Company: Royal Netherlands Aerospace Centre

BOT FOR CONTINUING AIRWORTHINESS:
PROOF OF CONCEPT FOR PASSENGER CARRYING DRONES

-

Bachelor Thesis

By

Joran Wieringa

Bachelor Thesis

Submitted to the BMS Faculty

University Of Twente

For the Course Bachelor Thesis IEM

Commissioned by the Royal Netherlands Aerospace Centre

Under Supervision of Dr. D.R.J. Prak, R. Brink, MSc. & Dr. P.B. Rogetzer

Industrial Engineering and Management, PO Box 217, 7500 AE Enschede, tel. +31(0)534899111

August 2021

Preface

Dear reader,

You are about to read the bachelor thesis "Bot for continuing airworthiness: proof of concept for passenger carrying drones". This research has been conducted at the Royal Dutch Aerospace Centre in Amsterdam as graduating assignment for my bachelor Industrial Engineering and Management at the University of Twente. This thesis aims to explore the possibility of introducing autonomy in the process of continuing airworthiness in urban air mobility. The slogan of the Royal Dutch Aerospace Centre, "Dedicated to innovation in aerospace" is suitable for this research, as this bachelor thesis is the first to research possibilities in this field.

Personally, I have had a great time working at and with the Royal Dutch Aerospace Centre. I am extremely grateful for this opportunity and feel like I have grown as a person working there. I would like to thank Rob Brink for guiding me in both professional as well as personal growth.

Furthermore, I would really like to thank my supervisor at the University of Twente Dennis Prak. I have really enjoyed all our meetings we had together and would like to thank you for the opportunities and trust you put in me. Your feedback was highly insightful and helped me to think more critically. I feel like because of you, this paper, and my bachelor thesis will be of a higher academic level. I would also like to thank my second supervisor at the University of Twente Patricia Rogetzer. Thanks for all the time and effort you put into giving feedback for my bachelor thesis. The critical view on my bachelor thesis was really useful, reminding me that not everyone has a background in aerospace.

With great joy have I worked on this bachelor thesis for the last couple of months. I hope that you, the reader, enjoys reading my thesis just as much.

Joran Wieringa

August 2021

Management summary

Continuing airworthiness produces a significant administrative footprint, as every event, inspection and decision must be documented. Currently the administrative footprint causes all sorts of problems for air vehicle owners such as being: cost intensive, time intensive and prone to human error. Continuing airworthiness entails all the processes ensuring that, at any time in its operating life, the air vehicle complies with the airworthiness requirements in force and is in a condition of safe operation.

To solve the problem a bot is developed. The bot autonomises the administrative processes of three continuing airworthiness processes namely the defect-, configuration- and maintenance management. To research whether the problem is solved and the bot is viable an experiment is executed. The experiment is a case study, which means for this research that the experiment involves a up-close, in-depth, and detailed examination of four cases, within a real-world context. The method used for data analysis is compliance by similarity.

The results of the experiment show that, by compliance of similarity, the bot takes just as good decisions from a safety perspective. Because the human and bot made the same take-off decisions four out of four times. Secondly, from a timely perspective, the bot performs better than humans, the bot provided an average time reduction of approximately 99,95%. Thirdly, from a cost perspective, the bot is less expensive to operate.

From the research can be concluded that the bot can perform continuing airworthiness management for drones used in urban air mobility, by creating a solid workflow net of all the continuing airworthiness processes, implementing the A-statement, implementing smart maintenance scheduling algorithms, defining functionality as a set of continuously asked questions and inserting all the decision making procedures by code into the bot.

However, the validity and reliability are limited. This is due to the fact that the results might not be fully representative, as the research utilises a relatively small number test scenarios. Secondly, the research faces subjectivity, furthermore, it is almost impossible to guarantee future success based on historical data. Even more, the maintenance scheduling process is quite limited as variables such as resources availability are not taken into account.

Table of contents

List of figures

List of tables

List of abbreviations

AD = Airworthiness Directive

AMM = Aircraft Maintenance Manual

ATL = Aircraft Technical Log

CAMO = Continuing Airworthiness Management Organisation

CDL = Configuration Deviation List

CMM = Component Maintenance Manual

EASA = European Union Aviation Safety Agency

MEL = Minimum Equipment List

SB = Service Bulletin

SRM = Standard Repair Manual

YAWL = Yet Another Workflow Language

1. Introduction

This chapter aims to introduce relevant background information to the research, explain why the research is conducted and highlight its relevance.

1.1 Background

Over the past decades advancements in aircraft technology enabled growth of the air transport industry, which developed into a $961.3 billion market (ATAG, 2020). In the industry a new market is emerging, both advancing economically as well as technologically, namely the urban air mobility market. Urban air mobility development is progressing due to technological advances, in both manned and unmanned air vehicle directions (Al Haddad et al., 2020). Urban air mobility is defined as transportation systems in the air in urban areas. The concept was initially developed as a solution to urban congestion, however there are many other applications (NASA, 2018).

Currently, the urban air mobility market is categorised by NASA into three segments: 1. Last-mile delivery, 2. Air metro and 3. Air taxi. The segments are defined by NASA as follows: last-mile delivery focuses on rapid package delivery from local distribution hubs to a receiving hubs, where deliveries are unscheduled and flight times are determined as orders are placed. Air metro resembles current public transit options such as subways and buses, with pre-determined routes, regular schedules, and set stops in high-traffic areas throughout each city in the air. Air taxi is a door-to-door ride-sharing operation that allows consumers to call air vehicles to their desired pick-up locations and specify drop-off destinations at rooftops throughout a given city. With air taxis, the destinations are chosen by the passengers (NASA, 2018).

For air metro and air taxi to become a suitable market, innovative air vehicles are being developed, namely, passenger carrying drones. These vehicles either fly autonomously or are being flown by a pilot on the ground, while having 2 to 8 passengers on board.

To ensure safety of air vehicles a continuing airworthiness management organisation, or CAMO, is required to be in place. This organisation is responsible for implementation of continuing airworthiness management tasks. Continuing airworthiness entails all the processes ensuring that, at any time in its operating life, the air vehicle complies with the airworthiness requirements in force and is in a condition of safe operation.

1.2 Problem definition

Passenger carrying drones come with exciting advancements and applications (NASA, 2018). However, these innovations also bring new unresolved and unexpected issues (Vascik et al. 2018). These technical, physical, operational, and integration challenges need to be overcome in order for urban air mobility to become a viable market in 2030 (NASA, 2018). One of these challenges is the administrative footprint of the continuing airworthiness process and its complementary problems.

Continuing airworthiness produces a significant administrative footprint, as every event, inspection and decision has to be documented. Currently the administrative footprint causes all sorts of problems for air vehicle owners such as being: cost intensive, time intensive and prone to human error.

1.2.1 Cost intensive

This administration can only be processed by licensed personnel authorised by a CAMO. That is why, regulations are in place that require air vehicle owners are to hire or setup a CAMO to process such an administrative footprint (European Union Aviation Safety Agency, 2021). Naturally, the CAMO is also responsible for planning and coordinating all maintenance activities such that the aircraft remains airworthy.

Air vehicles owners need to employ licensed personnel due to their lack of airworthiness regulatory knowledge, however because of the scarcity of this personnel and their required expertise, they are expensive to employ (NASA, 2018). Consequently, there is a barrier for entering the urban air mobility market. Subsequently, air vehicle owners either do not enter the market or establish their own CAMO, both scenarios are problematic.

Firstly, not entering the urban air mobility market is evidently problematic. Secondly, it is considered problematic that air vehicle owners setup their own CAMO. A CAMO is an extremely important and complex organisation. Lack of communication, distraction, lack of resources, stress, complacency, lack of teamwork, pressure, lack of awareness, lack of knowledge, fatigue, lack of assertiveness and low norms are all potential causes for errors (Clare & Kourousis 2021). When a CAMO operated by an air vehicle owner is authorised by EASA, the air vehicle owner is allowed to inspect and approve their own continuing airworthiness, thereby leaving open the possibility of unsafe cost reduction, tampering with maintenance records, while only having to report back to authorities after hazardous events have occurred. This compromises safety in urban air mobility.

1.2.2 Time intensive

The process for continuing airworthiness is an incredibly time intensive process. All handling is currently done manually and requires the reading of many documents, furthermore does the documentation also have to be done manually on paper. Combining the fact that the continuing airworthiness process is a gigantic process and every part of the process is handled manually, the process results to be very time consuming.

1.2.3 Human error

The CAMO is a complex organisation where people perform diverse tasks in a work environment where time pressure, sparse feedback and sometimes difficult environmental circumstances play a role. Furthermore, people have generic human erring tendencies, consequently varied forms of errors are prone to happen. Such errors can cause air turn-backs, delays in aircraft availability, gate returns, diversions to alternate airports and effect general productivity and efficiency of airline operations and inconvenience the customers. Where the most extreme examples of errors result in accidents and loss of life (Latorella & Prabhu, 2000).

1.3 Problem specification

This thesis focuses on the following three processes producing continuing airworthiness administration: firstly defect management, secondly configuration management, and thirdly maintenance management. Where defect management concerns how a defect will be handled safely, configuration management reviews an air vehicle related to the configuration, assessing the components and parts installed on the air vehicle, and maintenance management assesses when maintenance is necessary. It is important to note that a defect does not necessarily mean that the air vehicle will be grounded, with a defect an air vehicle can still be allowed to fly for a certain number of hours before some form of action is required to deal with the defect.

1.4 Problem solving

To solve the problem a bot will be developed. The bot will autonomise the administrative processes of the defect-, configuration- and maintenance management.

Solving the problem would lower the barrier for entering the urban air mobility market while guaranteeing the same safety level. Furthermore, human error will be removed, due the autonomation, the workload of the CAMO will be lowered and the bot will have a positive impact on the environment, since the need for paper documents will be eliminated by the digitalisation of the administration.

1.5 Research questions

The supporting research question for the development of the bot is as follows:

"How can a bot perform continuing airworthiness management for drones used in urban air mobility?"

In order for a bot to perform continuing airworthiness management, it must first be known what continuing airworthiness management entails. This leads to the following (sub)question:

1. What are the CAMO tasks and processes?

After it is known what the continuing airworthiness management tasks and processes are, in the scope of defect-, configuration- and maintenance management, then these process should be converted into a executable form for the bot. This leads to the following (sub)questions:

2. How can the defect-, configuration- and maintenance management processes be autonomised?
    a. What is the workflow of the CAMO tasks and processes?
    b. What are the decisions to be made for CAMO tasks and processes?
    c. How can a bot persistently provide reliable CAMO tasks decisions as output?

Once the bot has been developed that autonomises the defect-, configuration- and maintenance management processes, the bot should be validated. To do so, an experiment will be designed, that answers the following (sub)questions:

3. "Does the bot for continuing airworthiness management tasks for drones in urban air mobility perform just as good, better or worse than humans do?"

2. Theoretical framework

To research how a bot can perform continuing airworthiness management for passenger carrying drones in the field of urban air mobility, several topics should be addressed. Firstly, continuing airworthiness processes are defined further. Secondly, autonomation in aviation is briefly reviewed. Next, literature is reviewed which form a basis for bot development, as they provide a framework on how processes can be translated into a workflow. At last, theory is reviewed that shows maintenance scheduling challenges and opportunities.

2.1 Continuing airworthiness

For air vehicles to fly safely they have to be airworthy. A set of processes monitoring safety, performing maintenance, repair and maintenance to an air vehicle and its parts in order to keep them compliant with airworthiness requirements is known as continuing airworthiness. Research often approaches these processes in general aviation with the goal of optimising it for air vehicle owners and creating mathematical models to do so (Goncharenko, 2018). Or approach continuing airworthiness from a safety perspective and try evaluate reasons for failure (Clare & Kourousis, 2021). Continuing airworthiness in UAM however, is a topic that yet has to be explored in both academic as well as regulatory directions (NASA, 2018).

2.2 Autonomy

Autonomation is a topic in aviation that has not always been free from challenges and controversies (Hobe & Scott, 2017). Introducing autonomy raises safety concerns and is seen as a big barrier (Perez, Clothier & Williams, 2013), on the other hand can introducing autonomy in aviation actually improve safety according to Shakir & Iqbal (2018).

Ferrel & Anderegg (2020) research assurance of system safety of autonomous systems in UAM. They try to find solutions by comparison of the automobile industry, which is similarly challenged by autonomous systems. However, in the research they focus on autonomation in sensing and perception of components used for autonomous flying and the complementary predictive AI and machine learning required. Ferrel & Anderegg (2020) concluded that regulations and applications used for autonomation in the automobile industry can be of use in the aviation industry. They highlight the advantages of continuously monitoring status of components. Even though the research does not speak of autonomation in the CAMO, usage data of components is an important input for the CAMO in making airworthiness maintenance task decisions.

2.3 Workflow development

Control flow patterns provide a yardstick for expressing process orchestrations. A control flow pattern is independent of different process languages. Basic control flow patterns include *sequence*, *and split*, *and join, or split*, *or join*, *exclusive or split* and *exclusive or join* (Weske, 2012).

As mentioned earlier, continuing airworthiness can be described as a set of processes. Consequently, continuing airworthiness can be expressed in work flow nets. Workflow nets consist out of event-driven process chains and enable representation of control flow structures. Workflow nets represent processes, focussing on activities and their execution constraints (Weske, 2012). The workflow acts as a theoretical model for bot development, as it is a visual presentation of the process.

2.4 Yet Another Workflow Language

The theoretical model must be implemented into a process language to develop a bot. The language utilised is Yet Another Workflow Language or YAWL.

YAWL is a process language that directly supports all control flow patterns and utilises the principles of workflow nets. The graphical representation of process models in YAWL is closely related to workflow nets. YAWL has excellent support for multiple instance patterns, which can be described as a process inside of a process, in other words a sub-process (Weske, 2012).

The YAWL system is an open source business process management system from science for science and is deemed perfect for the university research environment. Due to this, the system is continuously improving and expanding as new ideas for usage and applications are developed. One of them is blockchain integration where distributed ledger technologies can be leveraged to support interorganisational workflow without the need for a trusted intermediary (Adams, Hense & ter Hofstede, 2020).

2.5 Maintenance scheduling

Running the continuing airworthiness process in a bot created in YAWL should result in an overview maintenance tasks. These maintenance tasks will be scheduled by the bot as well.

Existing research with regards to aircraft maintenance scheduling and planning are limited in their capacity to deal with contingencies arising out of inspections (Samaranayake & Kiridena, 2012). Maintenance tasks that arise from regulatory limitations do allow for mathematical modelling and quantitative optimisation of maintenance task intervals (Petrov, 2014). These two maintenance scheduling decisions are interdependent but are often made independently (Cassady & Kutanoglu, 2003). Integrating the two process enables major scheduling benefits with other resources, such as personnel scheduling (Alfares & Bailey, 1997).

2.6 Conclusion

Continuing airworthiness can be defined as a set of processes that ensures safety for air vehicles, the bot should operate in accordance with those processes, however there are no regulations of continuing airworthiness in urban air mobility yet, consequently do these processes not exist for urban air mobility. For the bot to perform continuing airworthiness, autonomy is introduced. Autonomy and safety is a highly debated topic, whereby there is not a lot of research in the field of autonomy for continuing airworthiness. This research therefore explores the possibilities of autonomation in continuing airworthiness and delivers a bot that ensures safety in this field.

The bot ultimately will perform a workflow. A workflow can display a set event-driven processes, and is the choice of use in theoretical bot development. The language of choice for bot development is YAWL. YAWL uses the principal of workflow nets and is excellent in multiple instance patterns, so that sub-processes can easily be modelled. To ensure safety in airworthiness the bot assess the status of the air vehicle, and deems it airworthy or not. Resulting in an overview of maintenance task that will ensure airworthiness. These tasks will be scheduled in an unique way that both includes the random maintenance tasks that arise out of inspections as well as plannable cyclical maintenance tasks.

3. Theoretical model

The development of the CAMO bot is based on a theoretical model. This theoretical model will be elaborated on in this chapter. The chapter focuses on the CAMO processes on which the model is based, how the processes work together and the functionalities of the bot.

It is noteworthy that the theory below was not provided by the Royal Dutch Aerospace centre and is the product of research.

3.1 Introduction

Section 2.1 provided the general description of continuing airworthiness by defining it as a set of processes monitoring safety, performing maintenance, repair and maintenance to an air vehicle and its parts in order to keep them compliant with airworthiness requirements is known as continuing airworthiness. And section 1.1 explained that the CAMO is responsible for these processes. To explain the processes in more detail section 3.3 is constructed, such that sub question 1 *"What are the CAMO tasks and processes"* can be answered.

As explained earlier, there are currently three major problems with continuing airworthiness, as it is time- and cost-intensive and prone to human error. The bot will tackle these issues by introducing smart functionalities, such as, the A-statement, unique maintenance scheduling algorithms and the overall way the continuing airworthiness processes are modelled, which is elaborated on in section 3.2, 3.4 and 3.3 respectively.

3.2 The A-statement

The A-statement is a digital document constructed to store and keep track of take-off decision by the bot after it assessed the airworthiness of the air vehicle.

Every continuing airworthiness process ends with a task that creates either a "go" or "no-go" statement, these statements are stored in the A-statement. Resulting in an easy to trace drop-down list of "go" or "no-go" statements per subprocess, categorised per process. This way, errors can be traced back easily to the process that created the "no-go", which is illustrated in figure 1. A "no-go" is given when the process does not deem the air vehicle airworthy under the current state that the aircraft is in, a "go" is given when the air vehicle is considered airworthy by the process.

A statement:	No-go

Defect management:	No-go	Configuration management:	Go	Maintenance management:	Go
Subprocess A	Go	Subprocess A	Go	Subprocess A	Go
Subprocess B	Go	Subprocess B	Go	Subprocess B	Go
Subprocess C	No-go	Subprocess C	Go	Subprocess C	Go
Subprocess C1	Go	Subprocess C1	Go	Subprocess C1	Go
Subprocess C2	No-go	Subprocess C2	Go	Subprocess C2	Go
Subprocess C3	Go	Subprocess C3	Go	Subprocess C3	Go
Subprocess D	Go	Subprocess D	Go	Subprocess D	Go

Figure 1: the A-statement

The A-statement introduces a structure that makes it possible for maintenance engineers to quickly locate the problem and focus on what requires fixing. Consequently, problematic processes, subprocesses and even components can be identified and kept track of. This data opens the door for predicative maintenance and component life-time optimisation research.

Furthermore, the A-statement saves time. Tracking and locating issues are made simply by storing them structured in one place. Even more, human errors are made less likely, as the problems are identified and stored in the A-statement by the bot.

3.3 Processes of a Continuing Airworthiness Management Organisation

As mentioned earlier does the continuing airworthiness processes exist out of three processes: 1. Defect management 2. Configuration management and 3. Maintenance Management. These processes are in place to ensure that an air vehicle is kept in a condition where it remains airworthy throughout its life, such that it is technically fit for flight. Deciding on inspection methods, inspection intervals, repair actions, modifications and timescales are all part of these processes. The processes will be elaborated on into more detail in the following subsections 3.1.2, 3.1.3 and 3.1.4.

3.3.1 Set of continuously asked questions

Processes can be described by some set of continuously changing dependent variables, such that they can modelled (Fortier & Michel, 2013). Continuing airworthiness management is defined by a set of three processes, namely defect-, configuration- and maintenance management processes. Consequently, the three processes should all be described by some set of continuously changing dependent variables, as the processes are to be modelled.

The continuously changing dependent variables for the processes result from continuously asked questions that the bot asks itself, the questions are elaborated on per process in subsection 3.3.2, 3.3.3 and 3.3.4. It is possible to construct models of complex systems, by defining the set of continuously asked questions, even more, it is possible to study how they are to interact (Fortier & Michel, 2013).

3.3.2 Defect management

Section 1.3 defined defect management as the process that regulates how a defect will be handled safely when it occurs. In other words, the process assess the degree of urgency of the defects and determines the order of treatment of the defects.

3.3.2.1 Scope

The process of defect management makes an assessment that involves the defects of the air vehicle. The processes looks at all the defects of the air vehicle and decides both whether the air vehicle is considered airworthy under the current circumstances and how the defects are to be handled, if there are any.

A defect can be handled in different ways, variables to consider in decision making are: the state of the air vehicle, maintenance scheduling and type of defect. The following are options for handling a defect: modification, replacement, repairment or rectification. Any decision made should be within the boundaries of instructions for continuing airworthiness.

3.3.2.2 Input

Instructions for continuing airworthiness are considered as operational constraints for the process. The instructions for continuing airworthiness are obtained from 4 documents, which act as input for the process: 1. Aircraft Maintenance Manual (AMM) 2. Component Maintenance Manual (CMM) 3. Standard Repair Manual (SRM) 4. Minimum Equipment List (MEL).

The aircraft technical log, or ATL, is the last document of input for the defect management process. The ATL contains defect reports, which evidently is necessary for the process to run.

3.3.2.3 Output

The defect management process provides three outputs. Firstly, defect rectification advisory in the form of a maintenance tasks, which will be scheduled in the maintenance schedule. Secondly, status information, current status information of the air vehicle will

be updated and be visible in the administration. Thirdly, the A-statement, the process will provide either a “go” or “no-go” based on the airworthiness state of the air vehicle.

3.3.2.4 Process

The functionality of the model can be explained as a set of questions that the bot continuously asks itself. The following questions cover this:

- Is the defect registered?
- Can the defect be deferred?
- What is the status of the defect?
- When will the defect be rectified?

Combining the continuously asked questions with the instructions for continuing airworthiness a process for continuing airworthiness can be designed.

The process starts with the task “Reading defects”, where the defect reports from the ATL are read to establish whether there any defects. Secondly, the task “Check: is sensor on?” stars, where a check is performed whether the defect registrations sensor of the air vehicle, assuming passenger carrying drones have such a sensor built in, is working. The task is introduced in the process to add a layer of safety. By asking the sensor whether it is on, the process prevents that any existing defects are present but not recorded. If in the task is established that the sensor is off, the process immediately ends. When the sensor is on, the process proceeds to the task “Take-off check”. At this task it is checked whether the aircraft is still airworthy under the current state of the defects, and produces either a “go” or “no-go” in accordance with the AMM, CMM, SRM and MEL. After which, the defect management process terminates.

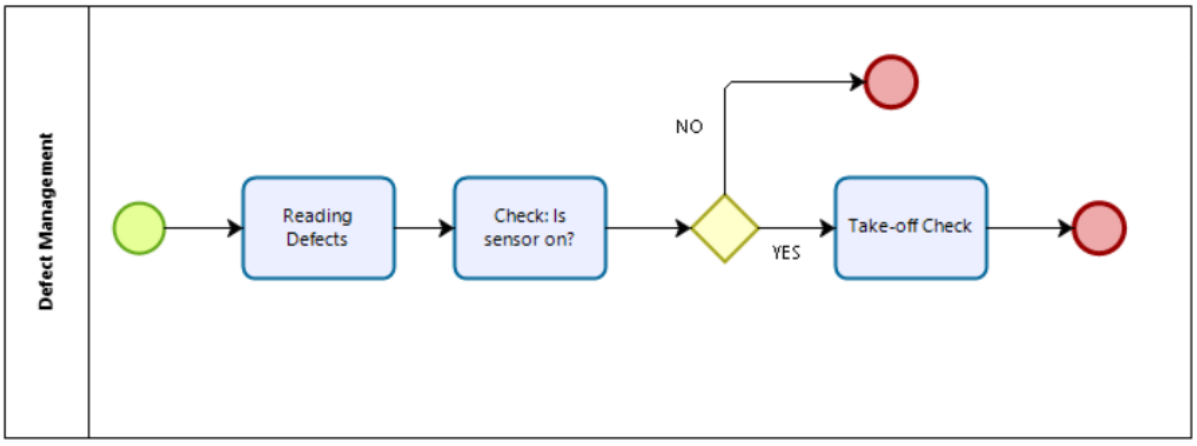

Figure 2: defect management control process

3.3.3 Configuration management

Section 1.3 defined configuration management as the process that reviews an air vehicle related to the configuration, assessing the components and parts installed on the air vehicle.

Configuration management is also referred to as the pre-flight check, this check can also be described as a set of questions that the bot continuously asks itself. The following questions can be asked:

3.3.3.1 Scope

The process of configuration management makes an assessment that involves the way the components of the air vehicle configured. The processes looks at all the components of the air vehicle, what serial number it has, possible modifications and its history.

The process reviews whether there are any configuration discrepancies that need to be resolved. Furthermore, the process reviews whether there are any applicable airworthiness directives or service bulletins that need to be embodied. Both reviews result into a maintenance task, if action is required.

An airworthiness directive is a mandatory maintenance task, it contains legally enforceable rules to correct an unsafe condition in an air vehicle, issued by either EASA or FAA. An airworthiness directive always contains information to which serial numbers of what air vehicle it must be enforced and has due date. A service bulletin is a document used by manufacturers of air vehicles, engines or components to communicate details of modifications which can be embodied in the air vehicle, and therefore is not mandatory.

3.3.3.2 Input

For the process function, input is required. Firstly, the allowed configuration is required, the allowed configuration provides the standard that the air vehicle should meet. Secondly, the current configuration is necessary, the current configuration is the current state of the air vehicle and is compared against the norm of the allowed configuration. Thirdly, information about non-mandatory modifications is required as input, as they altered the current configuration and should be taken into the comparison equation. Modification of components should be recorded closely as safety is a top priority. Fourthly, the airworthiness directives are required to stay compliant with allowed configuration. At last the service bulletins, which provide information for possible configuration modification.

3.3.3.3 Output

The configuration management process provides two outputs. Firstly, maintenance tasks, the process assess whether there are disappearances or airworthiness directives to be followed, and creates maintenance tasks accordingly. Secondly, based on the configuration of the air vehicle, does the process provide “go” or “no-go” into the A-statement.

3.3.3.4 Process

A set of questions that the bot continuously asks itself explains the functionality of the model. The following questions cover this:

- Is the configuration correct?
- Is the configuration complete?
- Is the configuration up-to-date?

The process for configuration management exists out of a main process, containing a subprocess. The main process starts with the task “Read allowed configuration”, where the allowed configuration for the air vehicle is being read. After completion of this task, a subprocess named “Up to date check” starts. The subprocess is visible in figure 3 and the main process of configuration management is visible in figure 4.

This process starts with the task “Read airworthiness directives and service bulletins”, where all airworthiness directives and service bulletins published are gathered. Next in the process is an AND-split, meaning that bot the “Check airworthiness directives” and “check service bulletins” tasks are executed simultaneously. The tasks assess whether there are any airworthiness directives or service bulletin applicable to the serial numbers of the components of the air vehicle. If there are, then a maintenance task and A-statement will be created accordingly. The maintenance task will be scheduled accordingly with the due date of airworthiness directive or service bulletin. It is possible that the allowed configuration is not up-to-date but a “go” is provided, this is the case when the due date of the document is not yet met, consequently the aircraft is legally deemed allowed to fly, else a “no-go” is provided. When both tasks are finished, the “Up to date check” task starts, which collects the “go” or “no-go” statements from the “check airworthiness directives” and “check service bulletins” tasks and provides a final “go” or “no-go” for the A-statement.

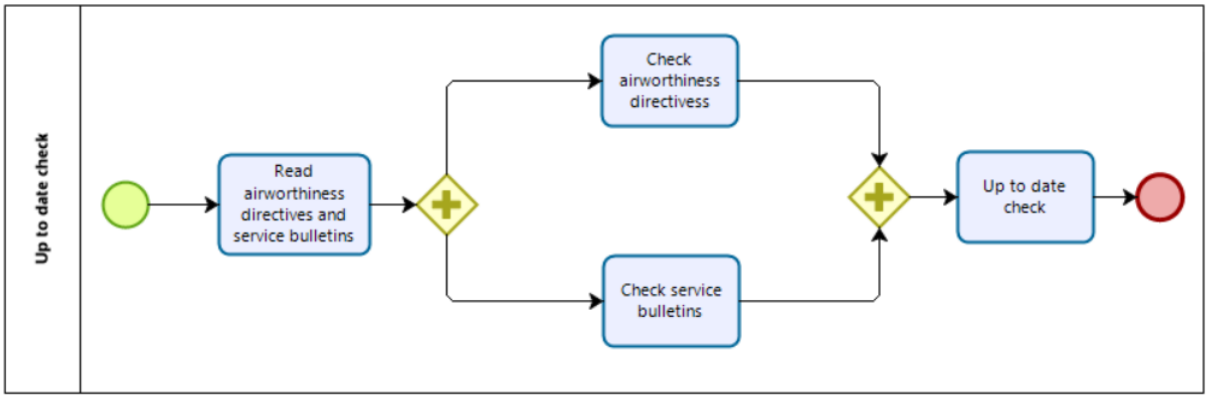

Figure 3: up to date check control process

After the "Up to date check", the main process continues. Where the current configuration is checked against the allowed configuration simultaneously for the component status, software load and life time items, where a "go" or "no-go" is provided accordingly. These statements are gathered at the "Take-off check" task where one final "go" or "no-go" for the A-statement is provided.

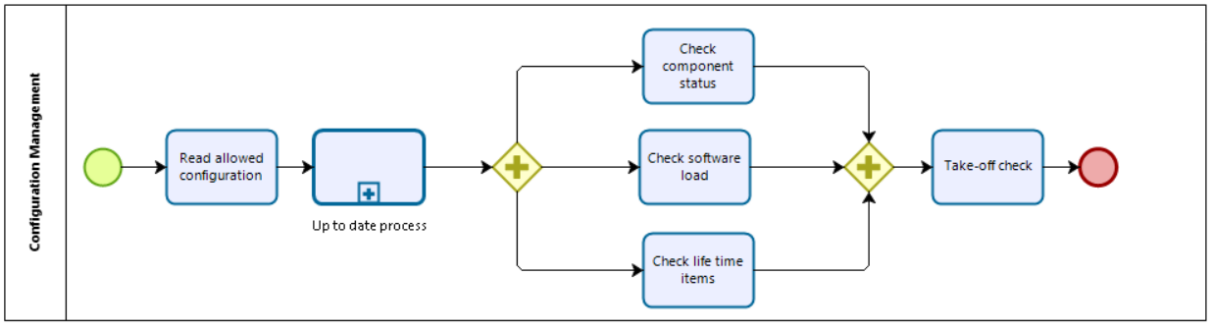

Figure 4: configuration management control process

3.3.4 Maintenance management

Section 1.3 defined maintenance management as the process that assesses when maintenance is necessary. In other words, the process monitors usage, performance and condition of the air vehicle and prescribes corrective actions when necessary.

3.3.4.1 Scope

The process of maintenance management involves all the maintenance task. The processes look at when the maintenance tasks are to be scheduled and makes sure that maintenance is done when needed.

3.3.4.2 Input

The maintenance management process requires usage data and the aircraft maintenance program to run. The aircraft maintenance program is considered as operational constraints for the process and he usage data is to be processed within those boundaries.

3.3.4.3 Output

The maintenance management process results in three outputs. Firstly, the maintenance schedule, the maintenance schedule exists out of an overview of maintenance slots and assigned maintenance tasks. Secondly, maintenance records, all the maintenance that is performed is documented and stored. Thirdly, the A-statement, where the "go" or "no-go" can be found from the maintenance management process.

3.3.4.4 Process

The functionality of the model is described as a set of questions that the bot continuously asks itself. The following questions cover the functionality of the maintenance management process:

- Which maintenance should be done?
- When should maintenance be done?
- How should maintenance be done?

The process starts with the "Read maintenance input" task, which reads all the existing maintenance tasks. Next, a subprocess starts. The subprocess is displayed in figure 5 and the main process is displayed in figure 6.

The subprocess "Creating a schedule" starts with the task "Check existing schedule", where is checked whether there already exists a schedule to which the maintenance tasks can be added, or a schedule should be created from scratch. When a schedule exists, the subprocess is terminated and the main process continues, if not, the task "Create a new schedule" will create a new schedule. In this schedule are cyclical tasks, which are inspections who are required to be done on a cyclical basis, assigned. Next, the "Assign semi-slots" task introduces semi-slots into the schedule. The "Create a new schedule" and "Assign semi-slots" are split, because it opens the possibility to optimisation in the future. In the current version of the bot, there is one semi-slot is implemented into the schedule. By splitting the tasks, it has become easier to set the slot number differently and run simulations testing optimal slot numbers.

When the subprocess is finished, the process continues with the task “Assign tasks into schedule”, whose algorithm is elaborated on in subsection 3.5. At last, the task “Take-off check” provides the “go” or “no-go” to the A-statement.

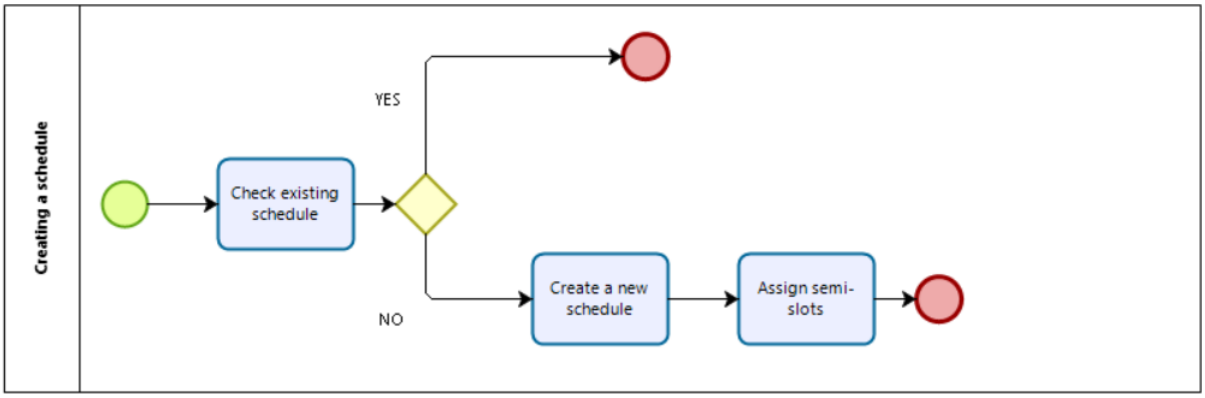

Figure 5: creating a schedule control process

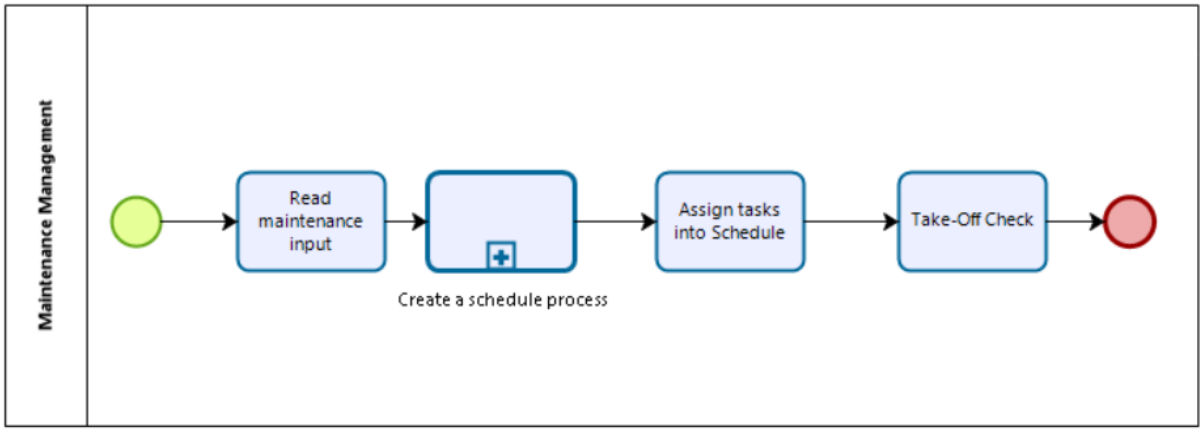

Figure 6: maintenance management control process

3.4 Integration of processes

The entire “CAMO Process” starts with reading in the input data. Processes have to start with reading input data due to software limitations. Namely, every process utilises a different database connection and therefore requires a separate reading task, the software architecture is elaborated on in subsection 4.4.

Next, the “Configuration management process” and “Defect management process” tasks are run simultaneously. The “Maintenance management process” task will start when both of the previous tasks are finished. After which, the final “Take off check” task will start, where the final A-statement, as displayed in subsection 3.2, is constructed.

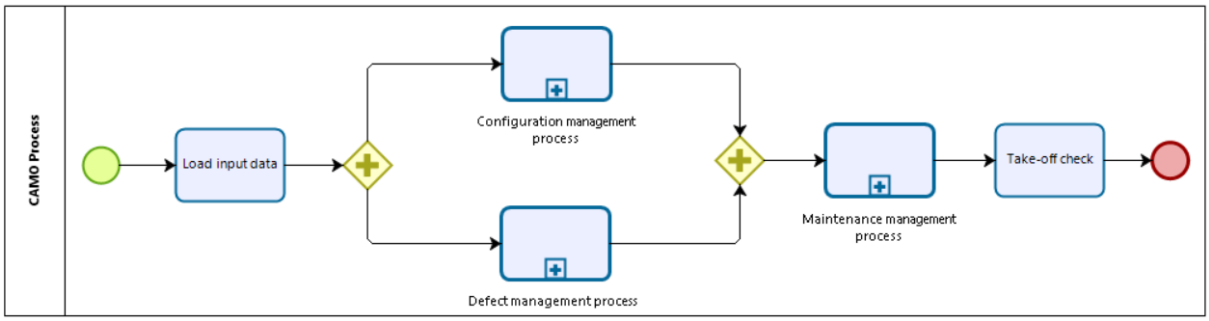

Figure 7: CAMO control process

3.5 Maintenance scheduling

When a defect occurs the bot decides whether the air vehicle is still considered airworthy or not, either way, the defect has to be repaired sometime. That is why, when a defect occurs a maintenance task will be created and then scheduled.

The maintenance schedule that will be developed will consist out of three types of slots. Namely, scheduled, semi-scheduled and unscheduled. The scheduled slots form the basis of the maintenance schedule, as they correspond with pre-known cyclical maintenance tasks. In between the scheduled slots will a semi-scheduled slot be placed, these will only be utilised if a defect is assigned to the slot. The bot will continuously asses the defect and check if they air vehicle is considered airworthy until the scheduled slot, if not, the corresponding maintenance task will be scheduled at the nearest slot. The unscheduled slot will be used if the defect makes the air vehicle not airworthy, meaning the corresponding maintenance task should be performed as soon as possible.

Semi-scheduled slots in the maintenance planning introduce a more structured way of handling defects, making it possible to integrate the maintenance schedule with more resources benefits (Alfares & Bailey, 1997).

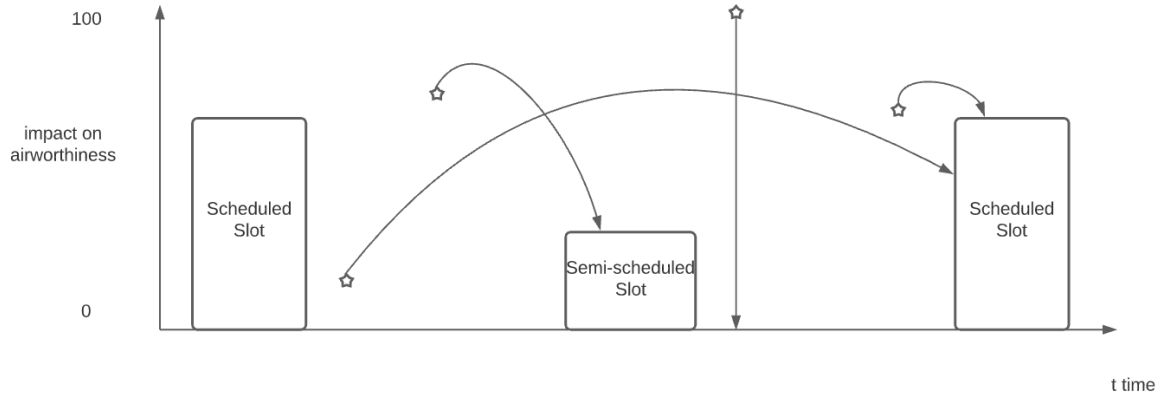

Figure 8: visual representation behind the logic of maintenance scheduling

To visualise the logic of the maintenance scheduling has figure 8 been constructed. It shows on the x-axis the time and on the y-axis the impact on airworthiness from 0 to 100, where the impact is inversely proportional with the ability to defer maintenance tasks. An impact of 100 would mean that the air vehicle would not be deemed airworthy, meaning the corresponding maintenance task should be performed as soon as possible. The scheduled and semi-scheduled are made visible with blocks, the defects with stars and the arrows point to the slot that the defect is assigned to.

3.6 Conclusion

The CAMO process can be divided into three processes namely the defect-, configuration- and maintenance management process. All of them end in a "go" or "no-go" which is placed in the A-statement. The A-statement is a digital document constructed to store and keep track of take-off decision by the bot after it assessed the airworthiness of the air vehicle.

The foundation of the maintenance schedule is based on pre-known cyclical maintenance tasks. The semi-slot feature introduce a more structured way of handling defects, making it possible to integrate the maintenance schedule with more resources. The number of semi-slots are set to one for this research, however future optimisation research is possible by varying the number of semi-slots.

4. Model development

This chapter translates the theoretical model explained earlier into a working bot, consisting of three communicating components, namely YAWL, PostgreSQL and IntelliJ. PostgreSQL is a free and open-source relational database management system and IntelliJ is an integrated development environment written in Java for developing computer software The chapter will start off by elaborating on the three components separately and finish off by explaining how information flows between the components.

4.1 YAWL layer

YAWL functions as the engine of the process, as YAWL is the program where the workflow is created. YAWL can be divided up unto three layers: 1. Workflow layer, 2. Resource Layer and 3. Data Layer. YAWL is able to combine the three layers into an executable web application on the YAWL server.

4.1.1 Workflow layer

The workflow layer is what makes the workflow going and consists out of a sequence of steps, every step is depicted in YAWL. In this subsection the control processes given in section 3.3.2.4, 3.3.3.4, 3.3.4.4 and 3.4 will be translated into a workflow net in YAWL. Translating the control process into YAWL will not change the order of function of the tasks. However, visual presentation can differ due to differences in language. Thus, this chapter merely shows the visual representation of the workflow as the logic and the functionality of the workflow is already provided above.

First, the workflow of the defect management control process is the following:

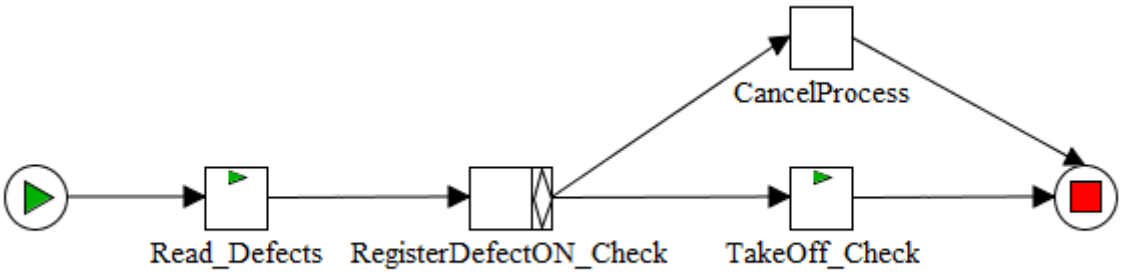

Figure 9: defect management workflow

Secondly, the subprocess of the configuration management process, “Up to date check” is as following:

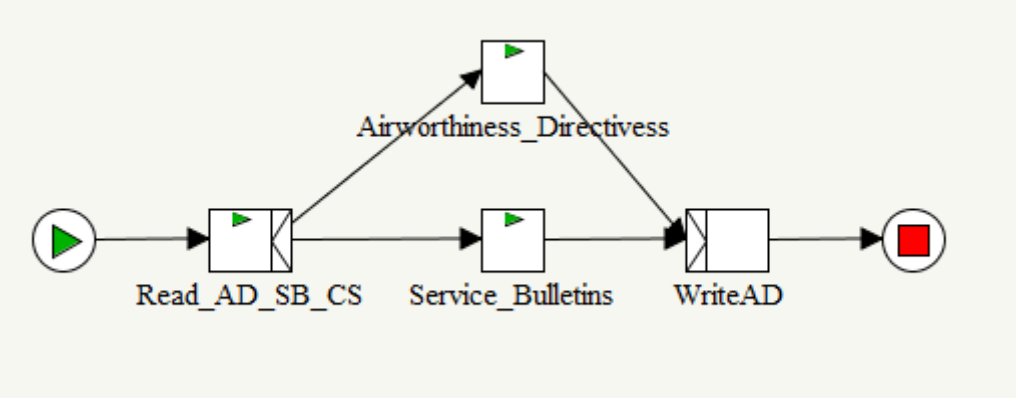

Figure 10: Up to date check workflow

With the main process of the configuration management process being:

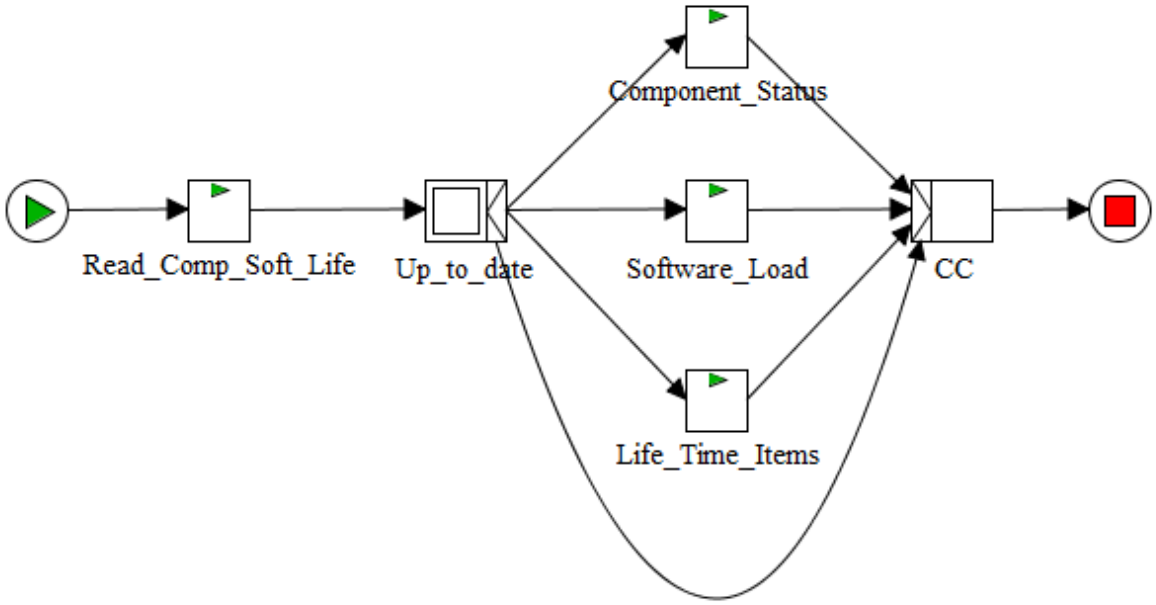

Figure 11: configuration management workflow

Thirdly, the subprocess of maintenance management process "Creating a schedule" is given:

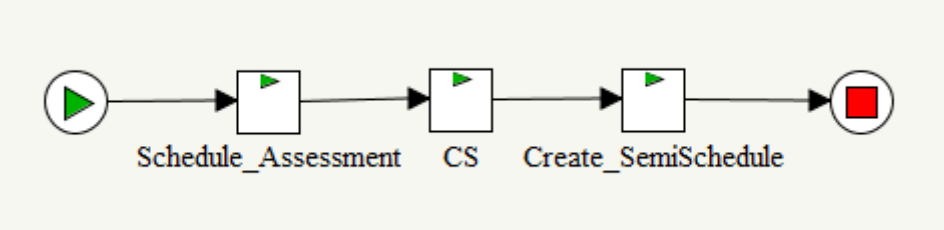

Figure 12: creating a schedule workflow

After which the maintenance management process with the scheduling tasks can be displayed:

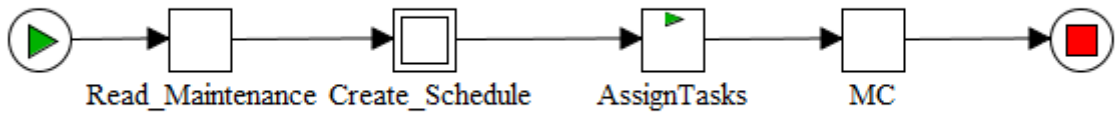

Figure 13: maintenance management workflow

After integration of all the processes, the following workflow in YAWL can be constructed:

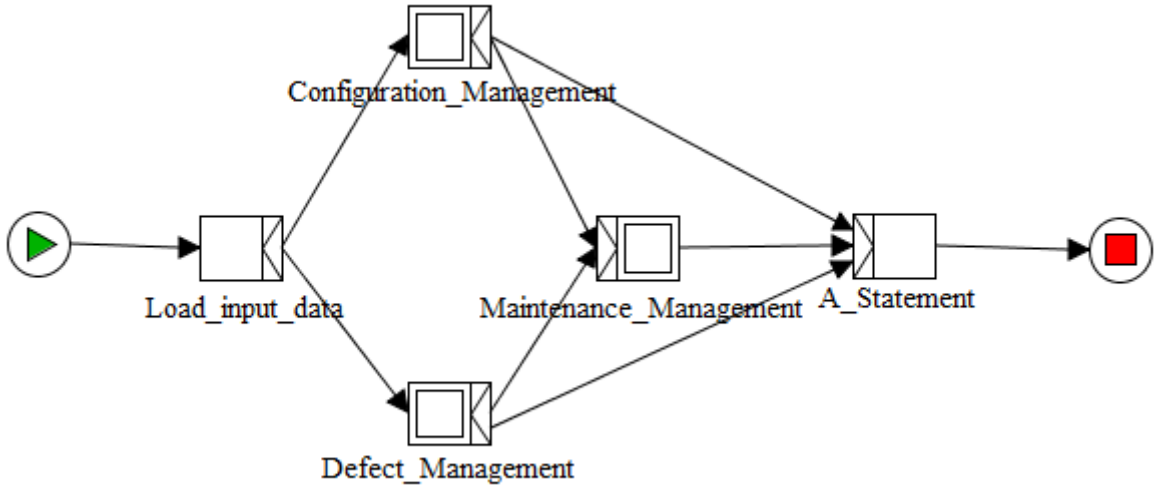

Figure 14: CAMO workflow

4.1.2 Resource layer

The resource layer concerns who is going to execute the task during the run time. In the bot there is only one participant defined in the resource layer, as the main goal of the bot is to perform continuing airworthiness management autonomously.

The participant is defined as Operator 1 and is in reality a person from the CAMO who reads the information provided by the bot. Even though the bot only defined 1 participant in the workflow, the possibility of using multiple participants in the future should not be excluded. Since, maintenance tasks could autonomously be scheduled into a roster, where the tasks are assigned to specific personnel of the maintenance team and each member of the team acts as a participant.

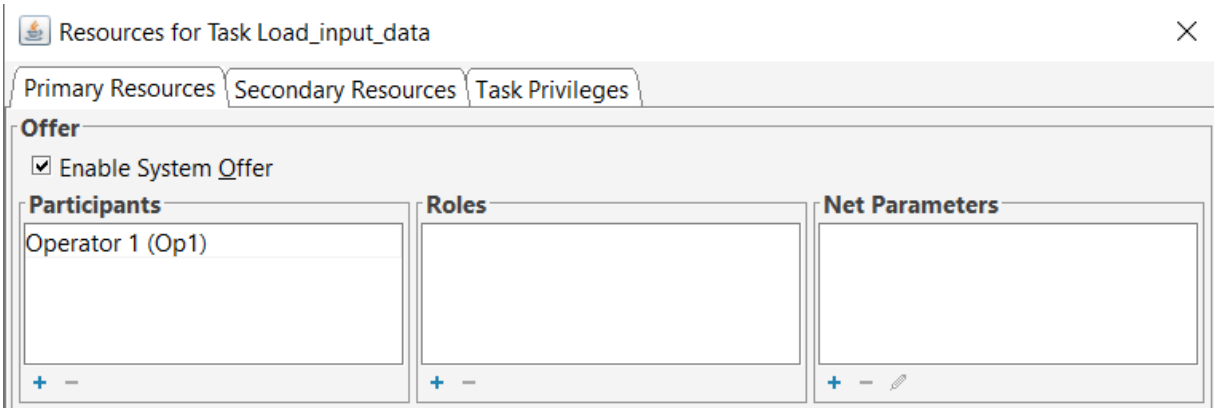

Figure 15: resources screen in YAWL

4.1.3 Data layer

The data layer consists out of a declaration of variables and their datatype, for each of the tasks involved in the workflow specification. In the workflow specification the model has identified two types of data that has the be put through, namely the input for the A-statement and the maintenance tasks.

For the main process, consisting out of the configuration management task, defect management task, maintenance management task and A-statement task, the following data variables are defined: A_statement, ConfigCheck, DefectCheck and MaintCheck.

The ConfigCheck, DefectCheck and MaintCheck are the result of the corresponding processes and entail the “go” or “no-go” with regards to airworthiness. These variables then again act as the input for the A-statement task. Only if all three variables are “go”, or in the data terms “true”, then the A_statement variable will also be “go”, which is the final approval before take-off. A visual display of the data variables mentioned can be

found in figure 15, similarly the depiction of all the subprocesses will be visible in the appendix.

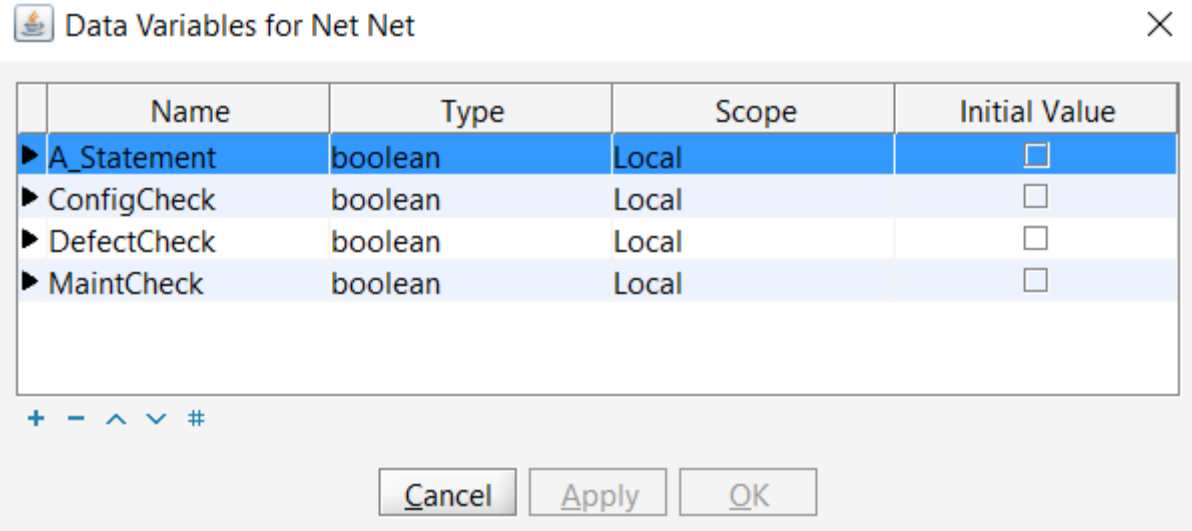

Figure 16: display of data variables in YAWL

4.2 Database layer

The database is built in PostgreSQL and functions as the place where all the data is stored, such as the aircraft status, aircraft history, maintenance schedule, but also restrictions gathered from documents such as the MEL.

For each of the processes a separate database is created to bring more structure to the management of the databases. Each database consists out of multiple tables, containing the rows and columns filled with information.

4.2.1 Configuration management

The configuration management database contains all the data required to run the configuration management workflow and make configuration management decisions. The database consists out of 7 tables namely:

1. Component Status
2. Component Status Identification
3. List of Limited Life-Time Items
4. Software Load
5. CS_Check
6. LTI_Check
7. SL_Check

The component status table contains a list of all the components of the air vehicle. Every component has their own Component_ID, Component_Type_ID and Classification_ID. Making the components easily trackable, recordable and identifiable.

	Component_ID [PK] integer	Name character varying	Serial number character varying	Quantity character varying	Component_Type_ID integer	Classification_ID integer
1	1	Fuselage	M1280010	1	1	1

Figure 17: component status table

The Component_Type_ID table refers to a primary key in the component status identification table, as this table shows the origin of the component. Using the IDs to categorise components makes the entirety of the list more compact, easier to sort and more accessible to programming languages.

	Component_Type_ID [PK] integer	Identification character varying
1	1	Airframe
2	2	Instrument Panel
3	3	Documentation
4	4	Battery Box
5	5	Accesoires

Figure 18: Component_Type_ID table

The table List Of Limited Life_Time Items stores information of life limitations of certain component with its relevant information. The table contains 14 columns of information.

Life_Time_ID [PK] integer	Item character varying	Part_Number character varying	Type character varying	Serial_Number character varying	Date_Installed date	Approved_Life_Limit_Months integer

Approved_Life_Limit_Hours integer	Conduct_Life_Limit_Date date	Conduct_Life_Limit_Hours integer	Conduct_Life_Limit character varying	Classification_Id integer	Check character varying	Current_Hours integer

Figure 19: List Of Limited Life_Time Items table

The component software of an air vehicle consists out of different types of software, that is why a separate table for the software is constructed. Where is shown for every type of software if it is critical, whether and whether it is present in the air vehicle, making it easy to perform the "go" / "no-go" check for the software part of the configuration checks.

Software_ID [PK] integer	Software character varying	Part_Number character varying	P/N_Revision character varying	Critical_Software boolean	OnBoard boolean	Check boolean

Figure 20: Component software table

For every table is another table constructed that stores the final "go" / "no-go" for that checking process. What makes tracing back faulty components more easy if a "no-go" is presented in the final A-statement under configuration management.

4.2.2 Defect management

The defect management database contains all the data required to run the defect management workflow. The database contains 2 tables and mainly acts as a storage of the defect reports, who on itself act as input for the maintenance management workflow. The 2 tables are:

1. DefectReport
2. DH_Check

The DefectReport table consists out of 11 columns. The Defect_ID is in place to track progress or any changes made to a defect, as condition can worsen. The WorkOrder_ID contains information of the maintenance task, as the task can differ between for instance replacement or repairment. The Component_ID is in place to track the history of defects of the component and the Critical_Defect shows whether the maintenance task will eventually be scheduled in an unscheduled slot or not.

Defect_ID [PK] integer	WorkOrder_ID [PK] integer	Description character varying	Component_ID integer	Quantity integer	Date_of_Defect date	Status character varying
1	1	Altimeter	26	1	2020-07-21	NEW

Remarks character varying	Current_date date	Manufactured_in_conformity boolean	Critical_Defect boolean
Type approval: S-55-97...	2021-07-22	true	false

Figure 21: DefectReport table

The DH_Check table again is for storage of the “go” / “no-go” statement. And checks whether any of the defects is critical.

4.2.3 Maintenance management

The maintenance management database contains all the data required to run the maintenance management workflow and make maintenance management decisions. The database consists out of 8 tables:

1. Maintenance_Slots
2. Maintenance_Schedule
3. Tasks
4. Task_Origin
5. DefectReport
6. UpToDate
7. AD_Check
8. SB_Check

The Maintenance_Slots table has the information about the on what date which type of slot is placed and how many tasks are scheduled on that day.

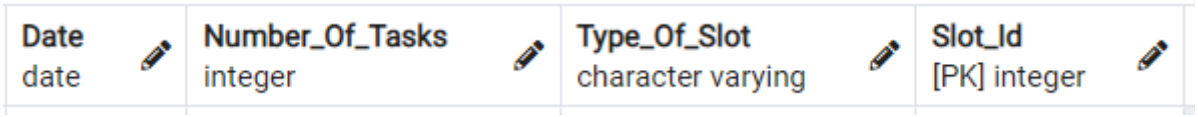

Figure 22: Maintenance_Slots table

The Maintenance_Schedule table contains the actual planning, including the specific tasks that are scheduled on which day.

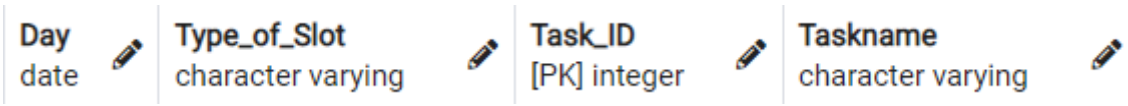

Figure 23: Maintenance_Schedule table

The details of these tasks can be found by its corresponding Task_ID in the Tasks table. Such as the category, the scheduling decision day, the day the maintenance task has started and ended, a safety performance indicator, the corresponding Defect_ID, the last date after which the defect becomes critical and information about the origin of the defect.

Taskname character varying	Task_ID [PK] integer	Task_Category character varying	Scheduling_date date	Start_date date

End_date date	SPI character varying	Defect_ID integer	Limit_date date	Task_Origin_ID integer

Figure 24: Task_ID table

The Task_Origin table contains the information about the origin of the task.

Task_Origin_ID [PK] integer	Task_Origin character varying
1	Airworthiness Directive
2	Service Bulletin
3	Unscheduled Maintenance (Defect)
4	Scheduled Maintenance

Figure 25: Task_Origin table

The DefectReport table is exactly the same as the DefectReport table from Configuration Management, but has to be copied due to integrational software constraints between database management and YAWL.

Defect_ID [PK] integer	WorkOrder_ID [PK] integer	Description character varying	Component_ID integer	Quantity integer	Date_of_Defect date	Status character varying
1	1	Altimeter	26	1	2020-07-21	NEW

Remarks character varying	Current_date date	Manufactured_in_conformity boolean	Critical_Defect boolean
Type approval: S-55-97...	2021-07-22	true	false

Figure 26: DefectReport table

The UpToDate table is in place for storing a variable that checks whether the Maintenance_Schedule is up to date with regards to outstanding tasks and scheduled tasks. The AD_Check and SB_Check are in place for similar reasons, they however store variables that check whether there are any new airworthiness directives or service bulletins that are missed.

4.3 Java layer

The built Java layer functions as the brain of the bot, as the Java code contains the executable created logic. The code provides the logic that performs the actual checking and scheduling that is mentioned before. To perform the checking procedures of the air vehicle,

information about the air vehicle is required. Information that is stored in the database. Via code in Java, this information can be read and utilised. This is done by building a database access object, or DAO, in Java, having such a class makes it possible to built other classes in Java that communicate, read, write, and calculate with the data from the accessed database. All the code discussed in section 4.3 can be found in appendix A.

4.3.1 Database Access Object class

The DAO class will be elaborated on, taking as example the DAO for the configuration management process. The page starts off with the import of multiple Java packages for the bot to be able to communicate with the database, which uses SQL and is displayed in figure 30. Then a connection to the database is built in a public, which is visible in figure 31.

After a readable connection is built, methods can be built that utilise this connection. A method is a block of code which only runs when it is called. These methods, visualised in figure 33, contain SQL statements that specific which data should be read from the database. Figure 32 is an example of such an SQL statement, where the SQL statement will find a task according to its Task_Origin_ID.

4.3.2 "Go" / "No-Go" code

Every checking process ends with a "Go" or "No-Go" statement, indicating whether air vehicle is deemed airworthy on that front. These checks are performed by combining SQL statements and check columns in the database, all being read and executed by Java code.

For example, code of one of the configuration management process. The process starts by writing a check for every component, depending on the number of components that is required on the air vehicle and the actual number of component that is present on the air vehicle, where the requirements are categorised by the Classification_ID. When the check indicates true, then that component is deemed airworthy. This is visible in figure 34 as an SQL statement. Which then again will be executed in Java by the piece of code displayed in figure 35:

After all the checks are set for every component, all checks should be true for the air vehicle to be deemed airworthy. That is why the next piece of code performs a COUNT DISTINCT in SQL. The DISTINCT function eliminates the repetitive appearance of the same data, where the COUNT function counts the number of rows returned. In other words, if 1 row is counted, and it is not false, then all checks are true. This is prepared by

the SQL statement found in figure 36. Which then again will be executed in Java by the piece of code displayed in figure 37.

Simultaneously, the statement writes the outcome, either true or false, into the database where the variable will be stored. For this example, the outcome is stored in the CS_Check table.

4.3.3 Maintenance scheduling code

The maintenance scheduling code consists out of two parts, firstly the code that creates the schedule and secondly the code that assigns the maintenance tasks into the schedule.

First, it is assessed whether there currently exists a maintenance schedule, or that a new schedule is required. A new schedule is required, on the moment that the last scheduled maintenance slot is in the past of the current date. To have an overview of this the UpToDate table is constructed, that stores the variable whether the current maintenance schedule is up to date. The piece of code visible in figure 38 checks this and updates the table when necessary.

The moment a new schedule is required to be made, the following piece of code creates the schedule and semi-schedule respectively. Displayed in figure 39 and figure 40 respectively.

Before maintenance tasks can be assigned to the maintenance schedule, they should first be created. Defect reports will be converted into maintenance tasks by the piece of code in figure 41.

After all the maintenance tasks are read into the task table in the maintenance management database, tasks can be assigned to their maintenance slot. Which is done by running the piece of code displayed in figure 42.

All the pieces presented in code in this subsection are statement that are ran by being part of the method displayed in figure 43.

4.4 Integration of YAWL, database & Java

For the bot to operate, YAWL and the database must communicate. To do so, codelets can be implemented in a task in YAWL. A codelet is a piece of Java code in the form of a .jar file. When a codelet is implemented into a task, YAWL recognises as automated and will not require any human input for the task to be completed as the task will be completed after the .jar file is ran.

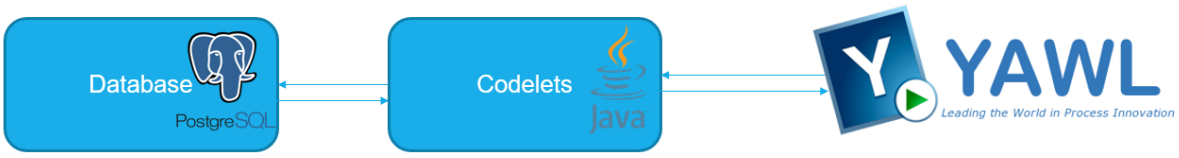

Figure 27: communication between YAWL, Java and PostgreSQL

A .jar file is created per Java class. Instead of creating a database connection per class one central connection has been built, that every separate class can call and use. This way every method is stored in a central place, the DAO, and every other class can call the methods that they require. This way the methods are organised and can easily be assigned to a task in the workflow. To display the structure the figure 28 has been constructed.

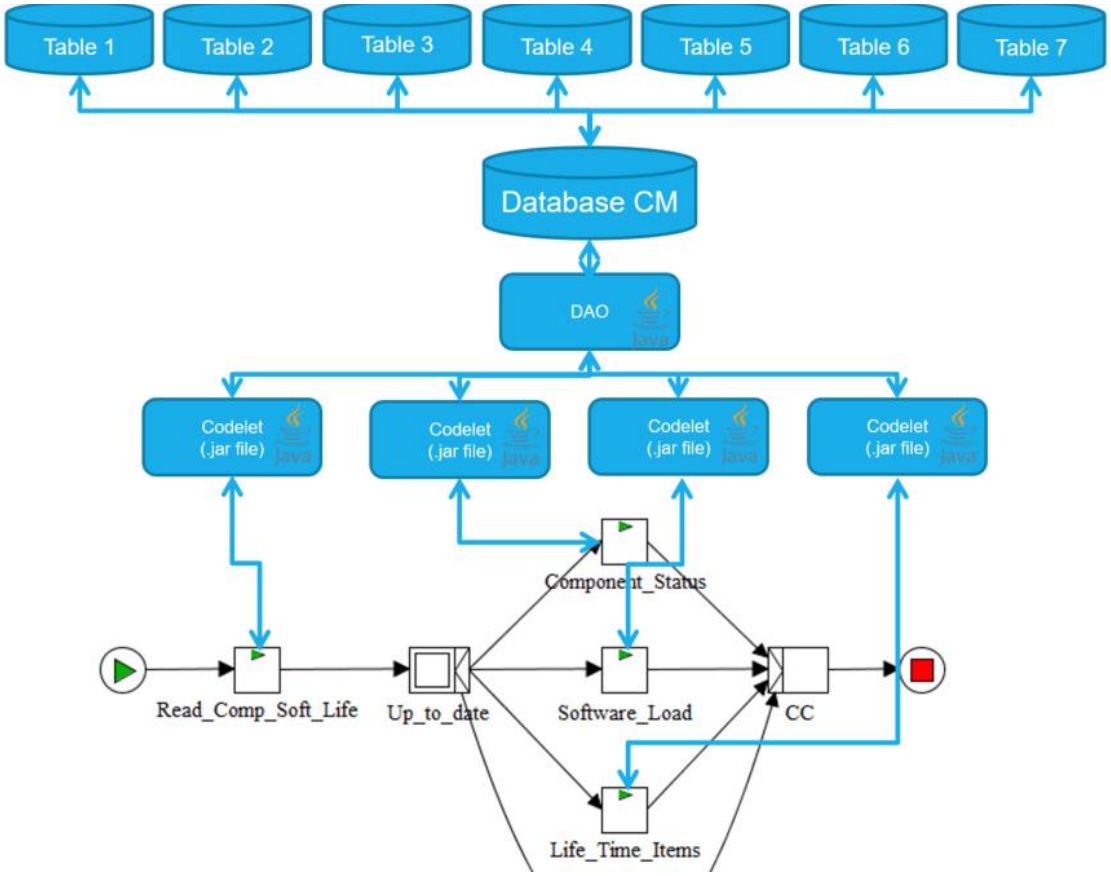

Figure 28: Communication structure

The components work together as follows. The process will start in YAWL, where the workflow and order of the task is displayed. Certain tasks contain codelets, meaning a .jar file will be requested. This .jar file is a Java class, in the class a method is called from the DAO. This method contains command in SQL or calculation, which then will be executed. The DAO receives the requested data from the connected database from the requested

table. After the method has been executed the tasks is finished, and the next task in YAWL will start.

For the configuration-, defect- and maintenance management processes there exists a workflow and a corresponding database. Since the databases are separate, are separate DAOs required as well.

5. Experiment

To answer the research question: “Does the bot for continuing airworthiness management tasks for drones in urban air mobility perform just as good, better or worse than humans do?” an experiment has been designed. This chapter will elaborate on how the experiment is designed and how it will be executed. The experiment is a case study, which means that the experiment involves a up-close, in-depth, and detailed examination of a particular case or cases, within a real-world context.

5.1 Assumptions

Before description of the experiment, assumptions about the model and the case should be discussed.

Firstly, since both the market of UAM as well as the vehicles are still under development it will be assumed that regulations with regards of continuing airworthiness of the current aviation industry will also be applicable to passenger carrying drones. Furthermore, because no historical data of passenger carrying drones are available for experimentation, historical data provided by the NLR of the Slovenian ultralight air vehicle manufacturer Pipistrel is assumed representative. Because Pipistrel is a pioneer in the field of ultralight air vehicles and is developing air vehicles in the field of urban air mobility.

Secondly, it is assumed that maintenance planning has no further restrictions, such as limited time or limited material.

Thirdly, it is assumed that the input data given to the bot as well as the CAMO specialist are without errors, as in reality human errors prior could be possible. Consequently faulty decision making based on faulty data is excluded.

5.2 Experiment design

Data is be gathered by comparison of output generated by a CAMO specialist and the developed bot.

5.2.1 Step 1: entering input data

The first step of the experiment is entering the input data into the database. This had to be done manually. The primary goal of entering the input data is creating different scenarios, or in other words cases. By creating different scenarios, the reliability of the bot was tested. The scenarios can be created randomly, since the experiment does not focus on the probability of any event happening, but focuses on how an event is handled when it

happens. The scenarios each test specific functionalities and requirements of the bot, the scenarios constructed can be found in the appendix.

5.2.2 Step 2: running the not

Secondly, the bot was run. This involves two parties, the bot and a CAMO specialist. Both the bot as well as the CAMO specialist will generate output data. For every scenario the output was documented and the time that it took was measured, on top of that the amount of documents that the CAMO specialist consulted was measured.

Furthermore, the CAMO specialist will be asked to write down the reasons behind made continuous airworthiness management decisions, to compare the logic and considerations between the bot and the CAMO specialist.

Measurement of the decisions, time and documents will contribute to showing the value behind the bot for continuing airworthiness.

5.2.3 Step 3: evaluating the bot

After the bot has generated its output, it must be validated. This will be done by the CAMO specialist, as the decision made by the CAMO specialist acts as the norm of good continuing airworthiness and is authorised by EASA to perform continuing airworthiness management.

Assessment of the authorised CAMO specialist will act as the norm of good decision making.

5.3 Data analysis validity

Primary quantitative data produced by the bot is collected and compared against human performance, where an analysis is performed whether the bot produced just as good, better, or worse results than the human.

This method of analysis is defined in the field of aviation as compliance by similarity (FAA, 2017). This method of analysis requires a design standard that, if met, proofs, or accomplishes, the safety. This method of analysis is often used in aviation, and approved and recognised as a solid method of analysis by both the Federal Aviation Administration, FAA, and EASA (FAA, 2017). This method of analysis lives up to the standards of EASA, consequently this method is valuable for the Royal Netherlands Aerospace Centre and suitable for academic research.

5.4 Experiment limitations

The designed experiment does have limitations, which can raise questions about validity. The experiment is designed to answer the question: "Does the bot for continuing airworthiness management tasks for drones in urban air mobility perform just as good, or better, than humans do?" However, both validity as well as reliability can be questioned.

First of all, only four scenarios are presented to the bot in the experiment, which is a relatively small number. Any conclusions that are made might not be fully representative. Consequently, results may not be generalised.

Secondly, the research faces subjectivity, as only one CAMO specialist is included in the experiment. This CAMO specialist determines the norm of good decision making. Experience, working circumstances, personal issues can all influence working speed differences between CAMO specialists. Logically, when repeating the experiment, results may differ.

Furthermore, the bot is limited by the capabilities of the designer in handling with uncertainties in the future. That the bot is able to work with historical data and known defects, does not guarantee that the bot will be able to deal with unknown events in the future.

6. Results

In this chapter, experiments are performed with 4 different scenarios to see how the bot and the CAMO specialist perform. For every scenario there is section, divided into the subsections containing the results from the specialist or the bot. Finally, this chapter ends with a conclusion that answers the following research question:

3. Does the bot for continuing airworthiness management tasks for drones in urban air mobility perform just as good, better, or worse than humans do?

6.1 The experiment

In this chapter the scenarios used for experimentation are displayed.

Scenario 1: Missing Documents

- Pre-Flight Check
- Is the aircraft allowed to take off?

Situation: There are a couple of paper documents missing that should be present in the plane. The following document is nowhere to be found: Weight & Balance Report

Scenario 2: Software Error

- Is the aircraft considered airworthy?

Situation: During inspection of the aircraft multiple software errors occur, consider the following software to be defect:

- System Controller
- BMS aux (front, rear brake)
- Instrument EPSI 570
- DC-DC converter
- Inverter

Scenario 3: Aircraft crash

Somewhere in Belgium an aircraft accident occurs. To investigate the crash the Belgium Civil Aviation Authority flies in to investigate and crashes.

After investigation and due diligence by EASA the following airworthiness directives are deployed:

Due to confidentiality reasons the actual airworthiness directive is not published, however the essence of the airworthiness directives are displayed in the following table.

Number	Limit Date	Mandatory	Applicability
1	20-06-2021	YES	NO
2	21-10-2021	YES	YES

Table 1: airworthiness directive data

Scenario 4: Pilot Notices & Reports Defect

The pilot notices an irregularity during flight and investigates further after landing. Defects have been noticed and reported in the ATL. These defect now have to be handled.

Description character varying	Component_ID integer	Quantity integer	Date_of_Defect date	Status character varying	Remarks character varying
Altimeter	26	1	2021-07-20	NEW	Type approval: S-55-97/74-96
Transponder	56	1	2021-07-20	NEW	[null]

Figure 29: defect reports

6.2 Results scenario one

In scenario one the configuration management process is tested.

6.2.1 Results scenario one: CAMO specialist

When scenario one was presented to the CAMO specialist the first act that was taken was checking again whether the document was missing. Once that was established, the ATL was consulted to check the last measurement date, which was at the factory. Since the document was missing, the CAMO specialist decided to consult the Continuing Airworthiness Management Exposition, or CAME. In the CAME chapter 1.14 was consulted and the procedures where read. Afterwards, the decision was made not to take off and a maintenance task was scheduled that the air vehicle needed to be weighted again.

The decision that the aircraft was not airworthy took 4 minutes and 18 seconds, whereby two documents were consulted.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
04:18	2	Negative	1

Table 2: Results scenario one CAMO specialist

6.2.2 Results scenario one: bot

The bot read as input that document was missing, this document was classified as necessary, consequently the decision was made that take off was not allowed. To fix the problem, a maintenance task is created and scheduled.

No documented were consulted, as the procedures that needed to be followed where built in the code. Which procedure needed to be followed was identified by the bot by usage of the Classification_ID.

Time passed	**Number of documents consulted**	**Take off decision**	**Maintenance task created**
66 milliseconds	0	Negative	1

Table 3: Results scenario one bot

6.3 Results scenario two

In scenario two the configuration management process is tested. This scenario is quite tricky as the software is not considered critical for flight in the configuration statement, however that statement does not have the authority to assess airworthiness, as by law only the MEL or CDL can do that.

6.3.1 Results scenario two: CAMO specialist

When the scenario was presented to the CAMO specialist the first thought was that these kind of decisions needed to be made in accordance with the MEL. So the MEL was consulted, however the MEL did not give a clear answer, afterwards the configuration list was consulted. It was established that the software is not critical, consequently the CAME is consulted as the MEL did not provide a clear answer. Procedure 1.3.1.2 is found and followed, the decision has been made to not take off. Then the CAME was consulted again to figure out how to report these multitude of defects in the Discrepancy Report, logically these documents were are consulted afterwards.

Time passed	**Number of documents consulted**	**Take off decision**	**Maintenance task created**
10:58	4	Negative	5

Table 4: Results scenario two CAMO specialist

6.3.2 Results scenario two: bot

The bot read the input that 5 pieces of software were missing, each of the pieces of software is organised by a Classification_ID. The bot therefore knew which procedures to follow and concluded that the air vehicle was not allowed to take off.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
40 milliseconds	0	Negative	5

Table 5: Results scenario two bot

6.4 Results scenario three

In scenario three the maintenance management process is tested and challenged. As two airworthiness directives are presented, however, not all the airworthiness directives are applicable.

6.4.1 Results scenario three: CAMO specialist

The first thing that the CAMO specialist looked up when once the airworthiness directives were presented to the CAMO specialist, were the applicability, so the serial numbers, and the limit date. The airworthiness directive handling procedures then were looked up in the CAME. Once the procedure was found, the CAMO specialist looked at whether the airworthiness directives are applicable to the air vehicle. To do so the type certificate of the air vehicle was checked, it was applicable to one air vehicle. However, now must be checked whether the airworthiness directive is applicable to the specific component type configured in the air vehicle. This is done by checking the serial number on the configured components, which can be found in the configuration aircraft statement. It now is established that the airworthiness directive needs to be executed, however the limit date has not yet been reached. Consequently, the aircraft is cleared for take-off.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
05:35	4	Positive	0

Table 6: Results scenario three CAMO specialist

6.4.2 Results scenario three: bot

The bot is able to read the airworthiness and directly determine whether the airworthiness directives are applicable as in depth information about the air vehicle and its configuration is part of the workflow. Furthermore, once the applicable airworthiness directive was established, its limit date was compared with the current date. As the limit date was not

met yet, the air vehicle was cleared for take-off and a maintenance task was scheduled in a maintenance slot before the limit date is due.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
41 milliseconds	0	Positive	1

Table 7: Results scenario three bot

6.5 Results scenario four

In scenario four the defect management process is tested and challenged. As two defect reports are presented, where one defect is critical.

6.5.1 Results scenario four: CAMO specialist

The first thing that the CAMO specialist wondered is whether the air vehicle was allowed to fly with the two aircrafts. To check this the CAME was consulted to look for the MEL procedures. In accordance with the MEL must the air vehicle have at least one working altimeter, in the defect report was reported that there is one defect altimeter. Now is the question, how many altimeters are present in the air vehicle, which can be checked in the configuration list. This process is repeated for the other defect, the transponder. It is concluded that the air vehicle remains airworthy without transponder, but is not deemed airworthy due to the broken altimeter.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
03:18	3	Negative	2

Table 8: Results scenario four CAMO specialist

6.5.2 Results scenario four: bot

For each of the defects a Component_ID and Classification_ID is known by the bot, consequently it is known how many components are and should be configured into the air vehicle. Based on this logic, the bot decided that the air vehicle could not take-off and that two maintenance tasks should be created and scheduled. As visible in table 9, is the runtime of the bot significantly higher than in the other scenarios. This can be explained by the fact that the algorithm used for this part of the process consists out of a bigger loop with more variables, and therefore takes more time to complete.

Time passed	Number of documents consulted	Take off decision	Maintenance task created
313 milliseconds	0	Negative	2

Table 9: Results scenario four bot

6.6 Comparison

The results of the experiment will be reviewed in his subsection.

First of all, there is a clear difference between the bot and the CAMO specialist in time spent on the decision making process. The bot significantly outperformed the CAMO specialist, this is made visible in the following table. Where the last column displays the reduction of time spent on the process in percentage.

Scenario	Time passed CAMO Specialist	Time passed Bot	Absolute difference	Percentual difference
1	04:18	00:00,066	04:17,934	-99,974%
2	10:58	00:00,040	10:57,960	-99,994%
3	05:35	00:00,041	05:34	-99,988%
4	03:18	00:00,313	03:17,687	-99,842%

Table 10: comparison of time

Furthermore, has the need for accessing documents been completely been reduced to zero. As shown in the following table.

Scenario	Number of documents consulted CAMO specialist	Number of documents consulted Bot
1	2	0
2	4	0
3	4	0
4	3	0

Table 11: comparison of number of documents consulted

More importantly, the both has made exactly the same decision with regards to the assessment of the airworthiness of the air vehicle. And therefore, by compliance of similarity, makes just as good decisions. On the matter of maintenance task created scenario 3 differed. The bot immediately created the maintenance task that eventually the airworthiness directive should be implemented, while the camo specialist did not.

7. Conclusion

In section 1.5, research questions were designed to answer the main research question:

"How can a bot perform continuing airworthiness management for drones used in urban air mobility?"

In the subsequent chapters the sub questions were answered. From analysing the CAMO and reviewing corresponding literature a theoretical framework and theoretical continuing airworthiness process have been constructed in chapter 2 and chapter 3 respectively. In chapter 3 it becomes clear what the CAMO tasks and processes are. The process is divided into three sub-processes defect-, configuration- and maintenance management, whose processes are defined as a set of questions that the bot continuously asks itself.

Literature from chapter 2 suggested that continuing airworthiness processes could be autonomised by translating them into a workflow net, which then again can be executed by YAWL. The development of such an autonomised process is presented in chapter 4. This chapter presents the workflow, with the corresponding decision making process, presented in Java code. With logic and workflow that is sound, a persistent reliable process has been created, with consistent decisions as output.

Chapter 6 shows the results from experimentation from which multiple conclusions can be drawn, within the boundaries of the limitations of the experiment design. These conclusions answer the question whether the bot takes just as good, better, or worse decisions as humans do.

Firstly, by compliance of similarity, the bot takes just as good decisions from a safety perspective. For the analysed scenarios, the human and bot made the same take-off decision. Secondly, from a timely perspective, the bot performs better than humans as Section 6.5 shows in table 10. Thirdly, from a cost perspective, the bot is less expensive to operate, assuming that the bot can replace CAMO specialists. Overall, has the bot outperformed the CAMO specialist in the created cases.

Concluding, a bot can perform continuing airworthiness management for drones used in urban air mobility, by creating a solid workflow net of all the continuing airworthiness processes, implementing the A-statement, implementing smart maintenance scheduling algorithms, defining functionality as a set of continuously asked questions and inserting all the decision making procedures by code into the bot .

8. Discussion

This research is the first to attempt to implement autonomy in the continuing airworthiness management processes. One of the reasons of implementing autonomy, is due to the removal of human error. However, whether the bot will eliminate human error and make the process safer, or just introduce more safety concerns is debatable. Either way, the research has shown that the bot is able to make sound decisions in way less time than any human would. However, due to limitations this statement is widely challenged. The case study has been very limited and has multiple threats to its validity and reliability. First of all, the results might not be fully representative, due to the relatively small number of scenarios tested. Secondly, the research faces subjectivity and thirdly it is almost impossible to guarantee future success based on historical data. Nevertheless, it has been shown that it is possible to implement autonomy in continuing airworthiness and make a bot perform airworthiness assessments.

Furthermore, the maintenance scheduling process is quite limited as variables such as resources availability are not considered. Also, predictive features in break downs are not considered. However, the bot has been scheduling maintenance in a smart way. Maintenance is often scheduled in a way that limits the ability to deal with maintenance tasks that occur sporadically, having to deal with them on a short notice without planning introduces all sorts of costs (Samaranayake & Kiridena, 2012). Introducing a slot and semi-slot system has given the air vehicle owners more structure, thereby opening the door to more integrational possibilities, with its complementary benefits (Alfares & Bailey, 1997).

9. Recommendations

Even though this research has limitations, does the concept of a bot for continuing airworthiness looks promising. This has been the first attempt made of introducing full autonomy into the continuing airworthiness process, the results of this thesis act as a proof of concept in this field. To continue the development in this field three paths will be proposed.

Firstly, the defect management process can be explored in more detail, as many opportunities arise in this field. This thesis has merely explored the three main processes in continuing airworthiness and looked whether a bot could autonomously perform these processes. However, more advanced decision making can be introduced into the defect management process, as only the current state of the aircraft was assessed in this thesis, without making calculations on the impact of the state of the decisions and possibilities of defects happening in the future. Looking ahead, Markov chain modelling can be introduced in the handling of defects. The possibilities of implementing Markov processes requires further research.

Secondly, this research introduced one semi-slot into the maintenance schedule to create more structure and integrational possibilities. However, one semi-slot might not be the optimal number of semi-slots for the timespan between scheduled slots. For further research, simulations should be run with different numbers of semi-slots while looking at the effect on for instance: costs, average break down time and component life time.

Thirdly, safety is the most important feature in continuing airworthiness and as mentioned earlier can autonomy be a threat. A study that assess the risk that occurs when introducing autonomy in continuing airworthiness is recommended.

At last, the database management system comes with exciting opportunities in combination with developments in YAWL. Safety and authenticity is very important in managing continuing airworthiness. Whether an air vehicle is considered airworthy can come down to a serial number with is authorised by a signature of an employee. The current system heavily relies on trust and has the danger of being exposed to adulteration (Aleshi, Seker & Babiceanu, 2019). By using a blockchain to record and enforce data-access policies, the need to trust a single entity with gate-keeping the data is removed (Hardin & Kotz, 2021). As YAWL is compatible with blockchain integration, this is a research path that is recommended.

10. References

Adams, M., Hense, A. V., & ter Hofstede, A. H. (2020). YAWL: An open source Business Process Management System from science for science. SoftwareX, 12, 100576. https://doi.org/10.1016/j.softx.2020.100576

Ahmed, S. S., Hulme, K. F., Fountas, G., Eker, U., Benedyk, I. V., Still, S. E., & Anastasopoulos, P. C. (2020). The Flying Car—Challenges and Strategies Toward Future Adoption. Frontiers in Built Environment, 6. https://doi.org/10.3389/fbuil.2020.00106

Al Haddad, C., Chaniotakis, E., Straubinger, A., Plötner, K., & Antoniou, C. (2020). Factors affecting the adoption and use of urban air mobility. *Transportation Research Part A: Policy and Practice*, *132*, 696–712. https://doi.org/10.1016/j.tra.2019.12.020

Aleshi, A., Seker, R., & Babiceanu, R. F. (2019). Blockchain Model for Enhancing Aircraft Maintenance Records Security. 2019 IEEE International Symposium on Technologies for Homeland Security (HST). Published. https://doi.org/10.1109/hst47167.2019.9032943

Alfares, H. K., & Bailey, J. E. (1997). Integrated project task and manpower scheduling. IIE Transactions, 29(9), 711–717. https://doi.org/10.1023/a:1018530303899

ATAG. (2020). Aviation Benefits Beyond Borders. https://aviationbenefits.org/media/167186/abbb2020_full.pdf

Cassady, C. R., & Kutanoglu, E. (2003). Minimizing Job Tardiness Using Integrated Preventive Maintenance Planning and Production Scheduling. IIE Transactions, 35(6), 503–513. https://doi.org/10.1080/07408170304416

Clare, J., & Kourousis, K. I. (2021). Analysis of Continuing Airworthiness Occurrences under the Prism of a Learning Framework. Aerospace, 8(2), 41. https://doi.org/10.3390/aerospace8020041

Ferrell, U. D., & Anderegg, A. H. A. (2020). Applicability of UL 4600 to Unmanned Aircraft Systems (UAS) and Urban Air Mobility (UAM). *2020 AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*. Published. https://doi.org/10.1109/dasc50938.2020.9256608

Fortier, P. J., Fortier, P. D., Michel, H., PhD, & Paul Fortier, D. S. (2003). Computer Systems Performance Evaluation and Prediction. Elsevier Gezondheidszorg.

Fu, M., Rothfeld, R., & Antoniou, C. (2019). Exploring Preferences for Transportation Modes in an Urban Air Mobility Environment: Munich Case Study. *Transportation Research Record: Journal of the Transportation Research Board*, *2673*(10), 427–442. https://doi.org/10.1177/0361198119843858

Goncharenko, A. (2018). Development of a theoretical approach to the conditional optimization of aircraft maintenance preference uncertainty. Aviation, 22(2), 40–44. https://doi.org/10.3846/aviation.2018.5929

Hardin, T., & Kotz, D. (2021). Amanuensis: Information provenance for health-data systems. Information Processing & Management, 58(2), 102460. https://doi.org/10.1016/j.ipm.2020.102460

HOBE, S., & SCOTT, B. I. (2017). International Civil Aviation and the Dehumanisation of Activities.

Holden, J., & Goel, N. (2016). *Fast-Forwarding to a Future of On-Demand Urban Air Transportation*. https://www.uber.com/elevate.pdf. Last accessed 15 August 2021.

Latorella, K. A., & Prabhu, P. V. (2000). A review of human error in aviation maintenance and inspection. International Journal of Industrial Ergonomics, 26(2), 133–161. https://doi.org/10.1016/s0169-8141(99)00063-3

Mounce, R., & Nelson, J. D. (2019). On the potential for one-way electric vehicle car-sharing in future mobility systems. Transportation Research Part A: Policy and Practice, 120, 17–30. https://doi.org/10.1016/j.tra.2018.12.003

Perez, T., Clothier, R. A., & Williams, B. (2013, May). Risk-management of UAS robust autonomy for integration into civil aviation safety frameworks. In Australian System Safety Conference (ASSC 2013) (pp. 37-45).

Petrov, A. N. (2014). Development of instructions for continuing airworthiness and aircraft logistic support analysis. In 29th Congress of the International Council of the Aeronautical Sciences, ICAS 2014.

Samaranayake, P., & Kiridena, S. (2012). Aircraft maintenance planning and scheduling: an integrated framework. Journal of Quality in Maintenance Engineering, 18(4), 432–453. https://doi.org/10.1108/13552511211281598

Shakir, H. M., & Iqbal, B. (2018). Application of Lean principles and software solutions for maintenance records in continuing airworthiness management organisations. The Aeronautical Journal, 122(1254), 1263–1274. https://doi.org/10.1017/aer.2018.65

Senkans, E., Skuhersky, M., Wilde, M., & Kish, B. (2020). A First-Principle Power and Energy Model for eVTOL Vehicles. *AIAA Scitech 2020 Forum*. Published. https://doi.org/10.2514/6.2020-1008

Weske, M. (2012). Business Process Management. Springer Publishing.

Yun, W. J., Jung, S., Kim, J., & Kim, J. H. (2021). Distributed deep reinforcement learning for autonomous aerial eVTOL mobility in drone taxi applications. ICT Express, 7(1), 1–4. https://doi.org/10.1016/j.icte.2021.01.005

11. Appendix

11.1 Appendix A: Code

ConfigurationDAO.java

```java
package org.yawlfoundation.yawl.resourcing.codelets;
// imports required for connecting the code to the database
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

Figure 30: database access object class

```java

public class ConfigurationDAO {
    //connection url to the database
    private static String connectionUrl = "jdbc:postgresql://localhost:5432/ConfigurationManagement";

    public static void ConfigurationDAO(String[] args) {
        // try block of code customerDAO.find, if an error occurs we are able to track where the error
        try {
            System.out.println(new ConfigurationDAO().find( Task_Origin_ID: 1));
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    private Connection con;

    public ConfigurationDAO() throws SQLException {
        super();
        // get connection username, password in PostgreSQL
        con = DriverManager.getConnection(connectionUrl,  s1: "postgres",  s2: "5h979zyd");
    }
```

Figure 31: database connection

```
SELECT * FROM public.\"Tasks\" WHERE \"Task_Origin_ID\" = ?
```

Figure 32: example of SQL statement

```java
// Prepares an SQL statement that find a customer according to its ID --> for read database codelets
public String find(long Task_Origin_ID) throws SQLException {
    // find by SQL query data from database
    PreparedStatement findStatement = con.prepareStatement( s: "SELECT * FROM public.\"Tasks\" WHERE \"Task_Origin_ID\" = ?");
    findStatement.setLong( i: 1, Task_Origin_ID);
    ResultSet resultSet = findStatement.executeQuery();
    if (resultSet.next()) {
        return resultSet.getString(resultSet.findColumn( s: "Check"));
    } else {
        return "Not Found";
    }
}
```

Figure 33: SQL query in Java

```
UPDATE public.\"Component Status\" SET \"Check\" = true::boolean \n" +
        "WHERE (\"Classification_ID\" = '1' AND (\"Quantity\" = '1' OR
\"Quantity\" = 'YES'));\n" +
        "UPDATE public.\"Component Status\" SET \"Check\" = FALSE::boolean
\n" +
        "WHERE (\"Classification_ID\" = '1' AND (\"Quantity\" = '0' OR
\"Quantity\" = 'NO'));\n" +
        "UPDATE public.\"Component Status\" SET \"Check\" = true::boolean
\n" +
        "WHERE (\"Classification_ID\" = '2');\n" +
        "UPDATE public.\"Component Status\" SET \"Check\" = true::boolean
\n" +
        "WHERE (\"Classification_ID\" = '3' AND (\"Quantity\" > '1' ));\n"
+
        "UPDATE public.\"Component Status\" SET \"Check\" = false::boolean
\n" +
        "WHERE (\"Classification_ID\" = '3' AND (\"Quantity\" < '2' ));
```

Figure 34: "go" / "no-go" SQL statement

```
public void updatePrepareCS2() throws SQLException {
    PreparedStatement updateStatement = con.prepareStatement("SQL STATEMENT
FROM FIGURE 33");
    updateStatement.execute();
}
```

Figure 35: "go" / "no-go" SQL in Java

```
WITH checker (\"Check\") AS (Select \"Check\" FROM public.\"Component
Status\")\n" +
        "UPDATE Public.\"CS_Check\"\n" +
        "Set \"Check\" = CASE\n" +
        "When ((Select count(DISTINCT \"Check\")FROM checker) = 2 or
((Select count(DISTINCT \"Check\")FROM checker) = 1 and (Select DISTINCT
\"Check\" FROM checker) = false)) then false else true\n" +
        "end;
```

Figure 36: SQL statement A-statement

```
public void updateWriteCS() throws SQLException {
    PreparedStatement updateStatement = con.prepareStatement("SQL STATEMENT
FROM FIGURE 35");
    updateStatement.execute();
}
```

Figure 37: Java execution code

```
WITH checker (\"check\") AS (Select \"Day\" FROM
public.\"Maintenance_Schedule\" WHERE \"Task_ID\" = 1 )\n" +
        " \n" +
        "Update public.\"UpToDate\" SET\n" +
        "\"UpToDate\" = case\n" +
        "WHEN (((Select \"check\" from \"checker\")) < Current_date) then
false \n" +
        "else true\n" +
        "end;\n" +
        "Select * FROM public.\"UpToDate\";
```

Figure 38: UpToDate table code

```
WITH creator (\"check\") AS (SELECT \"UpToDate\" FROM
public.\"UpToDate\")\n" +
        "\n" +
        "INSERT \n" +
        "INTO public.\"Maintenance_Slots\" (\"Date\",\"Type_Of_Slot\",
\"Slot_Id\")\n" +
        "VALUES\n" +
```

```
        "((CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false) \n" +
        "  \t\tTHEN (current_date + 140) \n" +
        "  \t\t\n" +
        "  \tEND)\n" +
        "\t,\n" +
        " (CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false) \n" +
        "  \t\tTHEN ('Scheduled') \n" +
        "  \t\t\n" +
        "  \tEND)\n" +
        " \t,\n" +
        " \t((CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false)\n" +
        "  \t\tTHEN (Select COUNT(\"Slot_Id\") FROM
public.\"Maintenance_Slots\") + 1 \n" +
        "\t\tELSE 0\n" +
        "  \tEND)\n" +
        "\t\t));\n" +
        "DELETE FROM public.\"Maintenance_Slots\" WHERE \"Slot_Id\" = 0;
```

Figure 39: schedule creation code

```
WITH creator (\"check\") AS (SELECT \"UpToDate\" FROM
public.\"UpToDate\")\n" +
        "\n" +
        "INSERT \n" +
        "INTO public.\"Maintenance_Slots\" (\"Date\",\"Type_Of_Slot\",
\"Slot_Id\")\n" +
        "VALUES\n" +
        "((CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false) \n" +
        "  \t\tTHEN (current_date + 70) \n" +
        "  \t\t\n" +
        "  \tEND)\n" +
        "\t,\n" +
        " (CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false) \n" +
        "  \t\tTHEN ('Semi-Scheduled') \n" +
        "  \t\t\n" +
        "  \tEND)\n" +
        " \t,\n" +
        " \t((CASE\n" +
        " \tWHEN ((SELECT \"check\" FROM creator) = false)\n" +
        "  \t\tTHEN (Select COUNT(\"Slot_Id\") FROM
public.\"Maintenance_Slots\") + 1 \n" +
        "\t\tELSE 0\n" +
        "  \tEND)\n" +
        "\t\t));\n" +
        "DELETE FROM public.\"Maintenance_Slots\" WHERE \"Slot_Id\" = 0;
```

Figure 40: semi-schedule creation code

```
Do $$\n" +
        "Declare\n" +
        "counter integer := 1;\n" +
        "Begin\n" +
        "\tWHILE counter < ((SELECT COUNT(\"Defect_ID\") FROM
public.\"DefectReport\") + 1) loop\n" +
        "\t\n" +
        "\tInsert \n" +
        "INTO public.\"Tasks\" (\"Task_ID\", \"Scheduling_date\",
\"Defect_ID\", \"Task_Origin_ID\", \"Taskname\", \"Limit_date\" )\n" +
        "  \n" +
```

```
        "VALUES \n" +
        "( \n" +
        "\t((SELECT COUNT(\"Task_ID\") FROM public.\"Tasks\") + 1),\n" +
        "current_date,\n" +
        "(SELECT \"Defect_ID\" FROM public.\"DefectReport\" WHERE
\"Defect_ID\" = counter),\n" +
        "4,\n" +
        "(Select \"Description\" FROM public.\"DefectReport\" WHERE
\"Defect_ID\" = counter),\n" +
        "CASE\n" +
        "\tWHEN \n" +
        "\t((Select \"Critical_Defect\" FROM public.\"DefectReport\" WHERE
\"Defect_ID\" = counter) = true) then current_date else\n" +
        "\tCASE\n" +
        "\t\tWHEN\n" +
        "\t\t(current_date > (SELECT \"Date\" FROM
public.\"Maintenance_Slots\" Where (\"Type_Of_Slot\" = 'Semi-Scheduled\n" +
        "'))) \n" +
        "\t\t  then (Select \"Date\" FROM public.\"Maintenance_Slots\"
WHERE \"Type_Of_Slot\" = 'Scheduled') \n" +
        "\t\t\t\telse (SELECT \"Date\" FROM public.\"Maintenance_Slots\"
Where (\"Type_Of_Slot\" = 'Semi-Scheduled\n" +
        "'))\n" +
        "\t\tend\n" +
        "\tend\n" +
        "\n" +
        ");\n" +
        "\t\n" +
        "\t\n" +
        "\tcounter := counter + 1;\n" +
        "\tend loop;\n" +
        "end $$;\n" +
        "\n" +
        "\n" +
        "(SELECT \"Date\" FROM public.\"Maintenance_Slots\" Where
(\"Type_Of_Slot\" = 'Semi-Scheduled\n" +
        "'));Select * FROM public.\"Tasks\"\n" +
        "Order by \"Task_ID\";
```

Figure 41: defect reports code

```
DELETE  FROM public.\"Maintenance_Schedule\";\n" +
        "\n" +
        "INSERT \n" +
        "INTO public.\"Maintenance_Schedule\" (\"Task_ID\", \"Taskname\",
\"Day\")\n" +
        "SELECT \"Task_ID\", \"Taskname\", \"Limit_date\"\n" +
        "FROM public.\"Tasks\"\n" +
        "\n" +
        "WHERE (((Select \"Date\" FROM public.\"Maintenance_Slots\" Where
\"Slot_Id\" = 1) > \"Limit_date\") AND ((\"Task_Origin_ID\" = 1) OR
(\"Task_Origin_ID\" = 2) OR (\"Task_Origin_ID\" = 3))) \n" +
        "\n" +
        ";\n" +
        "INSERT \n" +
        "INTO public.\"Maintenance_Schedule\" (\"Task_ID\", \"Taskname\",
\"Day\")\n" +
        "SELECT \"Task_ID\", \"Taskname\", \"Limit_date\"\n" +
        "FROM public.\"Tasks\"\n" +
        "\n" +
        "WHERE (\"Task_Origin_ID\" = 4);\n" +
        "\n" +
```

```
        "UPDATE public.\"Maintenance_Schedule\"\n" +
        "SET\n" +
        "\"Type_of_Slot\" = CASE\n" +
        "WHEN \"Day\" = current_date then 'Unscheduled'\n" +
        "WHEN\n" +
        "\t(\"Day\" < (SELECT \"Date\" FROM public.\"Maintenance_Slots\"
Where (\"Type_Of_Slot\" = 'Semi-Scheduled\n" +
        "'))) then 'Semi-Scheduled' else 'Scheduled'\n" +
        "end;\n" +
        "\n" +
        "UPDATE public.\"Maintenance_Schedule\"\n" +
        "SET\n" +
        "\"Day\" = CASE\n" +
        "WHEN \"Type_of_Slot\" = 'Unscheduled' then current_date\n" +
        "WHEN \n" +
        " \"Type_of_Slot\" = 'Scheduled' Then\n" +
        " (SELECT \"Date\" FROM public.\"Maintenance_Slots\" WHERE
\"Type_Of_Slot\" = 'Scheduled')\n" +
        "else\n" +
        " (SELECT \"Date\" FROM public.\"Maintenance_Slots\" Where
(\"Type_Of_Slot\" = 'Semi-Scheduled\n" +
        "'))\n" +
        "\n" +
        "end;
```

Figure 42: task assignment code

```
public void methodName() throws SQLException {
    PreparedStatement updateStatement = con.prepareStatement("SQL STATEMENT
");
    updateStatement.execute();
}
```

Figure 43: method for executing pieces of code