AUTOMATING THE COCKPIT

CONSTRUCTING AN AUTONOMOUS, HUMAN-LIKE FLIGHT BOT IN A SIMULATED ENVIRONMENT

Tamerius, Marten

Technical Report DKS-03-05 / ICE 05
Version 1.1, 06, 2003

Mediamatics / Data and Knowledge Systems group
Department of Information Technology and Systems
Delft University of Technology, The Netherlands

The Rotterdam Institute for Information Technology Programmes (RIVIO)
Hogeschool Rotterdam, The Netherlands

Tamerius, Marten S. (marten@ch.its.tudelft.nl)

"Automating the Cockpit: Constructing an autonomous, human-like flight bot in a simulated environment**"**

Technical Report DKS-03-05 / ICE 05
Version 1.1, 06, 2003

Mediamatics / Data and Knowledge Systems group
Department of Information Technology and Systems
Delft University of Technology, The Netherlands

The Rotterdam Institute for Information Technology Programmes (RIVIO)
Hogeschool Rotterdam, The Netherlands

http://www.kbs.twi.tudelft.nl/Research/Projects/ICE/

Keywords: ICE project, knowledge based system, flight simulator, bot, flight plan, cognitive model, flight analysis, autonomous bot, cockpit automation

"*...Postquam manus ultima ceopto imposita est, geminas opifex libravit in alas'ipse suum corpus motaque pependit in aura..!.*"[1]
(…Maar toen dit groots karwei dan toch voltooid was, bracht de knutselaar zijn lichaam op twee vleugels in evenwicht en bleef klapwiekend hangen in de lucht!)

Dat oogenblik vergeten we nooit: de sensatie van het buitengewone, toen die magere, lange man, daar los van de aarde met zijn toestel de lucht werd ingegooid en zweven bleef, hooger steeg, (alle wetten van zwaartekracht ten spijt). Toen riepen we: hoera, wij en allen die rondom stonden.[2]

"*Hos aliquis, tremula dum captat arundine pisces aut pastor baculo stivave innixus arator vidit et obstipuit, quique aethera capere possent, credidit esse deos*".[3]
(Een man die zat te vissen met een dunne rieten stengel, een herder leunend op zijn staf, een boer tegen zijn ploeg zagen hen gaan, verbijsterd, denkend dat het goden waren die door het luchtruim kunnen vliegen.)

de verwondering - het eerste opstijgen, P. van Dijck, 1999, HUISMUZIEK, after a text taken from the Dagblad van Noort- Brabant, dating June 28th 1909: "ooggetuige verslag van de eerste vlucht met een vliegtuig op Nederlands grondgebied"[4] and some verses from the story of Daedalus and Icarus taken from the eighth book of Publius Ovidius: Metamorphoses

[1] English translation: But when this big task was finally completed, the amateur balanced his body on two wings and stayed aloft.
[2] We will never forget that moment: The extraordinary sensation when the tall lean man became separated from the earth and was thrown into the air with his contraption, floating, climbing, (defeating all laws of gravity). At that moment we all shouted: hurray!
[3] A man who was fishing with a thin reed, a shepherd leaning on his staff, a farmer against his plough saw them go, confounded, thinking they were gods who could fly through the air.
[4] Eyewitness report of the first flight with an airplane on Dutch territory

Abstract

The increasing complexity of aircraft and air traffic has also increased the demands and stress on a pilot flying an airplane. To ease the workload on pilots and even take some of the tasks away from the attention of the pilot, research is in progress to filter the data shown to a pilot. In addition, some of the tasks are automatically performed by systems supporting the pilot. This may prevent disasters because of human errors and mistakes in judgment of the situation.

This report tries to answer the question: *Is it possible for an automated pilot to fly an airplane in a way resembling human pilot behaviour?* There are many automatic pilots that can fly an airplane, but most of them do not mimic human behaviour and have the same limitations humans have regarding, for instance, attention resources. The system designed in this project has the condition set specifically to support research projects focussed on testing human pilot behaviour under certain (critical) conditions.

The project starts with research on the manner in which human pilots interpret and respond to input cues provided by the instruments in the cockpit and visual cues outside the cockpit. A model is presented about the information processing and decision-making process of the pilot. Before the flight, a pilot is briefed about the details regarding the flight plan. The system designed in this project does this by analysing the flight plan, which is entered by a supervising person, carefully and converting it into a script of tasks and flight procedures that can be performed consecutively or, in some cases, concurrently. The *robot* that executes the flight is now easily able to interpret the script. Furthermore, it is designed according to the behaviour and constraints in the model that was discussed previously of human pilots.

The system's architecture is set up in a modular way to supports future extensions to the project. In addition to the research, a knowledge base had been set up, which describes the tasks and flight procedures a pilot is able to perform during a flight. The tasks and procedures are executed according the script that is generated before take off. The knowledge has been stored in the standard XML format for use by the system.

The prototype interfaces with Microsoft Flight Simulator 2002 to fly a Cessna 172SP Skyhawk. Finally, while the first prototype is still in an early phase, it is a first proof of concept. The robot performed well. Although it is not yet an advanced pilot incorporating complex artificial intelligence, it is able to execute a simple flight plan in normal environmental conditions.

Preface

This thesis was written in the course of my graduation project for the Rotterdam Institute for Information Technology Programmes (Hogeschool Rotterdam) at the Data and Knowledge Systems group of the Faculty of Information Technology and Systems at the Delft University of Technology. The graduation project was done as part of the ICE (Intelligent Cockpit Environment) project of the Data and Knowledge Systems group. The ICE project deals with a number of new techniques and technology to improve automation and pilot support in the cockpit. The goal is to design, test and evaluate those new techniques that can be used in the cockpit. The primary focus is artificial intelligence, which is used to fuse and process the available data and additionally it is used in the reasoning part of the system.

Project overview

The primary goal of the graduation project was to implement a fully autonomous robot that can execute a flight plan, entered by a supervisor, in a flight simulator. The actions of the bot should emulate the behaviour of a human pilot. At the beginning of the project, there was little knowledge about the behaviour of a human pilot regarding flying an airplane. Therefore research has been done to create a model of the human pilot and the tasks and procedures a pilot uses to fly an aircraft. When this was done, the implementation of the prototype started. At this point it became clear that the best way to acquire the flight plan from the supervisor (the person initiating the flight bot) was in a simple point-and-click manner. For this reason sectional maps were used that pilots also use in real-life navigation. The supervisor can point out the consecutive steer points and enter additional parameters for every steer point. With the flight plan in memory, the flight bot needs a way to understand what is being meant. At this stage, the flight plan is converted into a flight script, which is in essence a chronologically ordered list of tasks and flight procedures for the flight bot to execute. After this was done, the first prototype of the flight bot was implemented.

Acknowledgements

There are a number of people I wish to thank for their support. First and foremost are Dr. drs. Leon Rothkrantz and ir. Patrick Ehlert for their guidance and advice during my graduation project. The weekly sessions have stimulated me a lot to push on and to 'hit nails with heads'[5]. Furthermore I want to thank ir. Mohammed Abd El Ghany for his inspirational support from the Hogeschool Rotterdam. In addition, I owe thanks to my fellow graduates at the Mediamatics / Data and Knowledge Systems group with whom I worked closely: ir. Quint Mouthaan, Maikel van der Roest, Richard Harreman, Mahery Andriambololona and Pascal Lefeuvre. They all helped me in some degree to tackle many problems, to get culminating ideas for the design of the *flight bot* and just to have fun while working on the project. I would like to show my gratitude to my friends and family who have always shown interest in what I have been working on. Even though some of them did not entirely understand my work, a special thanks to them none the less. Last, but most certainly not least, I want to thank God for being the true inspirator of my life. Without Him I know I would not have come this far.

Report contents

This report consists of the following. The first chapter gives an introduction into the project. The following chapter discusses the view a (common) human pilot has on the aircraft system and tasks while operating an airplane. This leads to the manner in which a pilot undertakes a normal flight. It begins with the preparations before the actual flight, which is discussed in chapter 3. When the

[5] Explanation for the English reader: this is the literal translation of a Dutch saying meaning "go for it".

preparations are done, the actual flight begins. This calls for a pilot or robot that can execute the flight. The basic design and limitations of the *flight bot* that replaces the human pilot in this project, is discussed in chapter 4. The architecture and implementation of the application is explained in chapter 5. Much data is passed between modules and stored inside the application. Chapter 6 discusses the technique used for this purpose. The results of this project are mentioned in chapter 7. Finally, this report ends with chapter 8 discussing the conclusions and, in addition, some recommendations about future work regarding this project.

The appendices describe additional work related to this project. A preliminary study is included about recommendations concerning the selected flight simulator. In addition, the knowledge bases about the specification of the aircraft and the specification of the flight procedures and tasks that is used in the application are described in two appendices.

Marten Tamerius,
Delft, June 2003.

Contents

List of figures

Chapter 1 Introduction

1.1 Problem setting: Can pilots be automated?

Ever since man first flew a plane, flight rules have become increasingly complicated to the pilot. Were the Wright brothers in 1903 alone in the skies, nowadays pilots have to follow up guidance of the Air Traffic Controller, who keeps air traffic from becoming chaos and prevents disasters from happening. Moreover, not only air traffic has become increasingly dense, also planes have evolved rapidly. When flying, a pilot has to keep an eye on a lot of complex instruments and gauges to determine whether the plane is functioning correctly and flies on the desired course. These innovations have caused a dramatic increase of workload on the pilot, compared to the first fully operable planes.

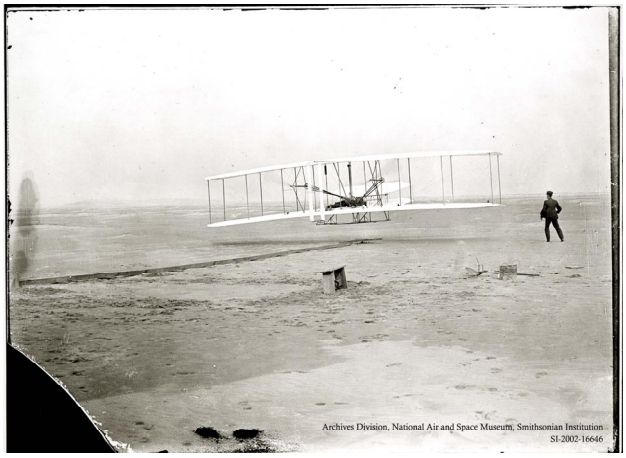

Figure 1 The first flight by the Wright brothers in 1903

This is where the Intelligent Cockpit Environment (ICE) project comes into play. The question is whether some, or all, tasks performed by the pilot can be automated. One approach is to use context-aware systems to monitor the airplane during flight. Such a system can be used to improve and filter the data that is shown to the pilot to reduce workload stress. For example, when a military airplane is under attack by an enemy, it is not of primary concern to the pilot whether a non-essential navigational instrument like the ILS (Instrument Landing System) is malfunctioning. This calls for an intelligent system that is aware of the situation of pilot and airplane at any particular moment.

Figure 2 The new Eurofighter Typhoon

In other cases, some pilot tasks may be fully automated to further reduce workload on the pilot. For the benefit of the ICE project, it is interesting to implement a fully autonomous (computer-driven) pilot, which can perform a great deal of the navigation a real pilot has to do to reach the destination. With this *flight bot,* particular situations can be simulated and analysed for multiple purposes. A pilot in training can compare a flight in a simulator with the one from the *flight bot* to review for what reasons particular decisions are made. A researcher may be able to find out in what part of a certain flight the most stressful situations occur for a pilot, and so on.

1.2 Automation of the pilot

1.2.1 Vision on this project

Will, one day, airplanes be as common in the sky as automobiles are on the roads of today? And, until that day, how will the aircraft evolve? With the current increase of earth's population, some day in the near future a solution will be demanded to the growing problem of traffic jams and the pollution of the environment because of the ever-increasing use of cars.

One solution is to seek the answer in the air. Aircraft can be designed to transport more people faster to their destination. However, as big airline jets are the highways of the skies, in the same way the small airplanes are the city streets of the air. This analogy also predicts the way traffic may be organised in the future. But as air traffic increases, so does the demand for pilots to fly the aircraft. If it comes to pass that the demand exceeds the number of trained pilots, new problems will arise.

With this scenario in mind, could it be possible to introduce systems into the cockpit that support pilots with the handling of an aircraft? Is it feasible to support even untrained pilots to fly a plane safely from start to destination using a software application? Is it really preposterous to try to automate the pilot out of the cockpit by giving a computer full control of the airplane? Of course, the road to a completely independent, autonomous fighter bot for the F-16, for instance, is a long one yet. But curiously enough, studies have shown that, even when computers are trained to do exactly the same thing as real humans, people do not trust the computer to handle occurring situations properly. Supporting or monitoring systems, on the other hand, are welcomed without any hesitation, as long as there is a human operator present to make the final decision. Although there are many understandable reasons for this behaviour, computers and software have become increasingly intelligent and may even surpass humans in some areas of control. However, before the computer can take over the cockpit completely, a lot of prejudices and fears people have about computers need to be dealt with. In addition, there is the legal responsibility of the matter. Therefore taking over a cockpit today is only accepted when it is a virtual cockpit in a flight simulator, where plane crashes are not as disastrous as in real life.

Figure 3 An F-16 Fighting Falcon

While there are already automatic pilots available, there are limitations to the workings of these supporting systems. The simplest designs can only hold a predefined heading and altitude. The more advanced will fly a certain path entered by a supervisor. But even in those models, advanced artificial intelligence is not incorporated. When another aircraft is on collision course, the supervisor has to interrupt the automatic pilot to steer clear from the other airplane. Can a system, aided by artificial intelligence, make the required decisions to ensure the safety of the plane? Furthermore, this project has the goal to implement human behaviour and limitations into the flight bot. This opens the way for researchers to study predefined situations and conditions without the need of a human pilot and an expensive simulator. In addition, the research can be repeated as many times as is needed. Also training a pilot may become more realistic in simulators using the *flight bot*.

1.2.2 Research on the decision-making process of a pilot

Before the question whether a system can make the required decisions to ensure the safety of a plane can be answered, a number of things have to be researched. In the first place, a general model has to be designed to create a framework for the decision-making process of a pilot. In other words, how does a pilot come to a certain decision in a particular situation? Secondly, research must be done to translate the designed cognitive model to a computerised pilot. In this stage of the project artificial intelligence kicks in. After the pilot model has been designed, the implementation cycle of the final application starts.

1.3 Project goal and sub-goals

The ultimate goal is to develop an autonomous, computer-driven pilot. Such a pilot, also called a robot or bot, should be capable of flying an aircraft in a simulated environment according to a flight plan. In addition, it should have capability of making in-flight decisions and actions to adjust the actual flight path to reach the destination of the flight plan. The decisions and actions are based on human-like behaviour and constraints.

While the goal to completely automate a pilot is rather unfeasible with the currently available knowledge and processing power, a number of workable sub-goals have been set for this project, of which the first two have the main focus:

- Design a cognitive model of a pilot, which can be used to analyse the flight plan and generate a basic flight script. The tasks and flight procedures that are used must be available to the pilot in an XML knowledge base.
- Design a system that can execute the flight script and perceives and evaluates certain situations during flight and reasons which actions should be executed from a generated list of possible solutions for a specific situation.
- Implement these models in an application, which can acquire, analyse and execute a flight plan in Microsoft Flight Simulator 2002. The architecture of the application must be set up in a way to support future extensions to the project.

1.4 Implementation

The development technique, which has been used for this project, is called prototyping. This manner of developing an application is very useful at the beginning of the project, when the exact specifications are not defined in detail. Prototyping works with a number of stages. Relatively early in the project cycle, a prototype of the main concept of the application will be developed. In later stages, the prototype will be worked out in detail. This way the application will evolve gradually into the end product.

The choice of modelling technique has been made taking into account the other current ongoing projects at the department. The most widely used technique is the Unified Modelling Language (UML), which is further described in Chapter 5 of this report.

The final application of this project can be divided into four tiers. These four stages are described in a simple flow chart in Figure 4.

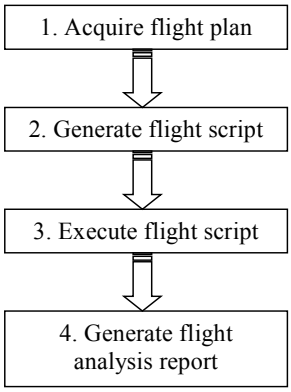

Figure 4 The four stages of the final application

1.4.1 Tier one: Setting up a flight plan

When the autonomous pilot has to fly a specific route, it must be made clear to the computer how it has to fly the chosen path. This knowledge is entered as a flight plan by the supervisor. The flight plan consists of a starting point and a destination (which are both airports) and possibly a number of waypoints the pilot has to fly by. Examples of waypoints are other airport locations, beacons, and GPS coordinates. Also other relevant information can be added to the flight plan.

1.4.2 Tier two: Analyse the flight plan and convert it into a flight script

After the flight plan has been entered, the computer decomposes the plan into the basic stages a typical flight consists of. These stages are then analysed using explicit knowledge about flying an aircraft. During the design stage of this project, a task analysis will be made to provide insight into the behavioural patterns of a real pilot during flight. This model of cognitive tasks can be used to construct the flight script.

In addition, verification will be done during the construction of the script to check consistency, ambiguity and other constraints of the provided flight plan. Examples of these constraints are time constraints (is it possible to reach the next waypoint at the specified time?), altitude constraints (it is definitely undesirable for the airplane to fly into a mountain during flight.) and so forth. Expert knowledge will be used to set these constraints.

1.4.3 Tier three: Use artificial intelligence to fly the flight plan

When the flight script is approved, an autonomous pilot can be instructed to start executing the script in a flight simulator. Often, when flying a real aircraft, in-flight decisions are made to adjust the actual flight plan. The autonomous pilot should also be able to make those decisions based on expert knowledge and situation-awareness. To study these decisions afterwards, a flight log is maintained for the supervisor. This will give him the ability to compare the provided flight plan with the actual flight.

1.4.4 Tier four: Flight analysis

Afterwards, the flight log can be studied to compare the entered flight plan to the actual flight plan. This is useful, for instance, for researchers, who want to study the results of a flight without having the need of a human being as a pilot.

Chapter 2 Designing a cognitive model

When designing an automated bot that must simulate human behaviour, it is necessary to know how people (or rather expert pilots) interpret and respond to data when executing a certain task. A cognitive model describes the way people process information depending on the mental model they have of a certain system. This chapter provides insight into the mental model a pilot has while flying an airplane.

2.1 Aviator abilities

In aviation, there is a huge level of experience and knowledge expected from the person piloting an aircraft. Many tasks must be done unconsciously because of the limited attention resources humans have. For example, when flying a traffic pattern at an airport while the pilot is waiting to land, flying the appropriate pattern at the correct altitude must become an automated skill, because many attention resources of the pilot are focused on the Air Traffic Control messages and the other planes in the vicinity to prevent disastrous collisions from happening.

As Figure 5 points out, the abilities of a (human) pilot can be divided into several layers. The basis for expert flying is the skills and knowledge a pilot obtains during training and flight. Some flight procedures can be done (after training) without the need of allocating many attention resources. These tasks are easy to automate. Much knowledge is based on a pilot's experience during his flight history.

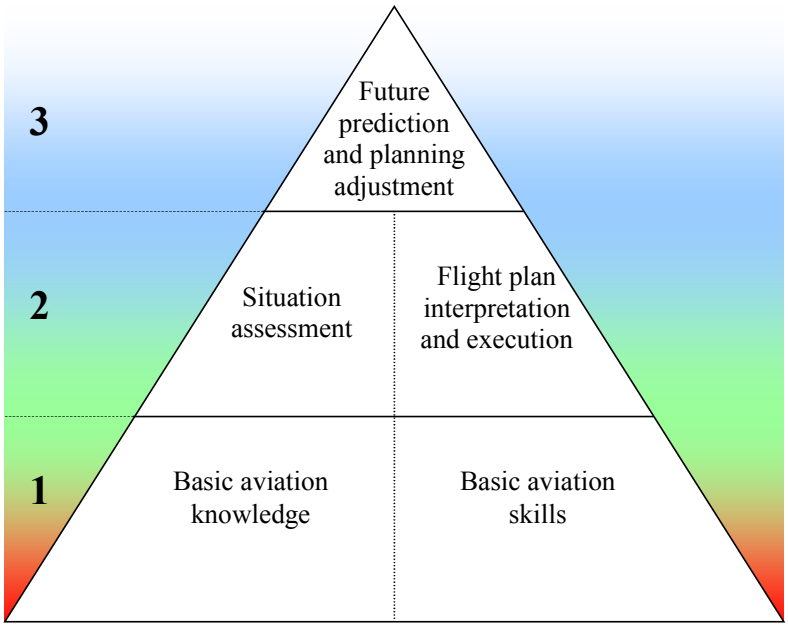

Figure 5 Layered model of the abilities of a human pilot

The second layer depends on the first layer, which is directly underneath the second layer. When a pilot has acquired much experience from earlier flights, he will also train his ability to be aware of potentially dangerous situations. In addition, this gives him the advantage to anticipate to and solve such a situation in an early stage. Usually a pilot wants to fly from airport to airport according to a flight plan. Assuming the pilot has the necessary skills (again depending on the first layer) he can navigate by means of waypoints to get to the appointed destination. Obviously the pilot should study the flight plan carefully before take-off to analyse all aspects of the flight. Not all possible situations can be foreseen. Many flight plan adjustments must be made in-flight because of changing conditions.

This calls for a flexible approach to the flight plan. It is not a flight script that is executed by calling predefined procedures with hard coded and unchangeable variables. For this reason the top layer consists of capabilities of the pilot to predict possible occurring future situations by analysing the flight history and the current airplane and environment status. When the pilot can assess the whole situation, proper measures can be taken to prevent unwanted future situations from happening by changing the flight plan.

2.2 Common cognitive models

Executing certain tasks can require the user to interact with the interface of a system to set or read specific parameters. The way people think about a system depends for the most part on the stimuli the system's interface gives to the user and how the system actually responds to the input users give. While working on the specific task, the user needs to gather information from the system and process it in order to respond appropriately. Obviously, the ability to get the information depends on the knowledge and the mental model the user has of the system and the task.

2.2.1 Information processing

Typically, there are four stages a person has to go through when processing information. This is presented in Figure 6, which describes Wickens' model of human information processing [WiNa88].

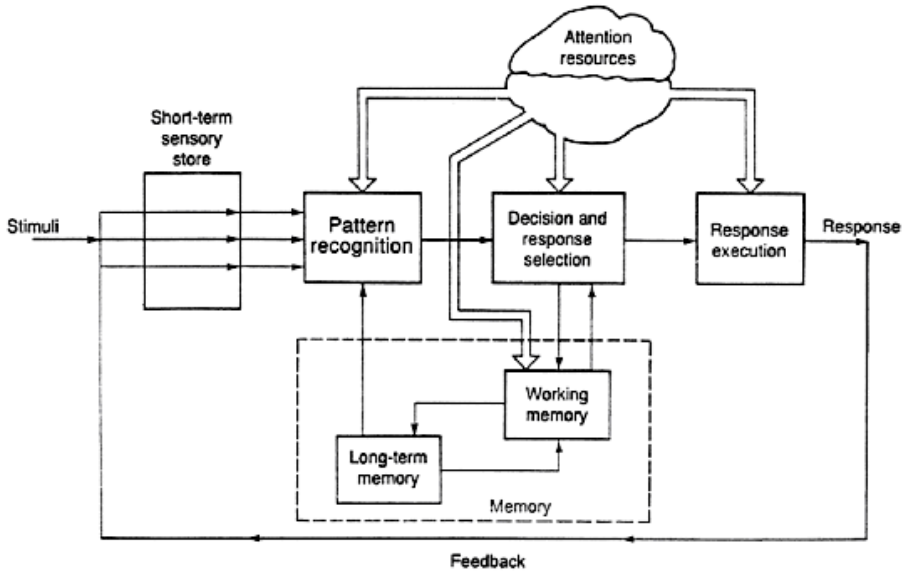

Figure 6 A model of information processing [WiNa88]

The first stage consists of the user getting stimuli from the system's interface or the working environment. A typical human being has five ways of sensing information, of which vision and hearing are the major ones. These sensors use a short-term memory to process the perceived signals briefly before they are send to the brain for further analysis.

After the sensors have converted the (physical) signals into information the brain is able to process, the data is filtered for useful information. This is done in cooperation with the (long-term and short-term) memory, which has useful knowledge about the current task and accumulated information from previous encounters with tasks similar to the one at hand.

Based on the filtered data a decision has to be made how to react to the current situation. During this stage, a selection has to be made from a range of actions, like store the data in memory, initiate a sequence of motor commands, or just stay idle for a certain period. The selection is based upon the priority and predicted (beneficial) result of the response. When a task exceeds a certain level of complexity, the user does not have enough attention resources available to process all the information. In that case, the user will only consume the information, which seems to have the highest priority level in order to accomplish the task.

When a response has been selected, it will be executed. This will often create feedback to the stimuli in the next cycle of information processing.

2.2.2 Levels of behaviour

Looking at the behavioural levels gives another perspective to the manner of information processing of a pilot. In Figure 7, there are three levels of behaviour mentioned in accordance with Rasmussen [Ras86] and Barbarino [KBD97]. The knowledge-based behaviour level is mainly targeted at the decision-making process. One level deeper resides the rule-based behaviour level. The situation awareness and situation assessment is primarily concerned with this level. The basic aviation tasks are processed on the skill-based behaviour level. The tasks at this level are performed automatically, without the need to allocate (many) attention resources by the pilot.

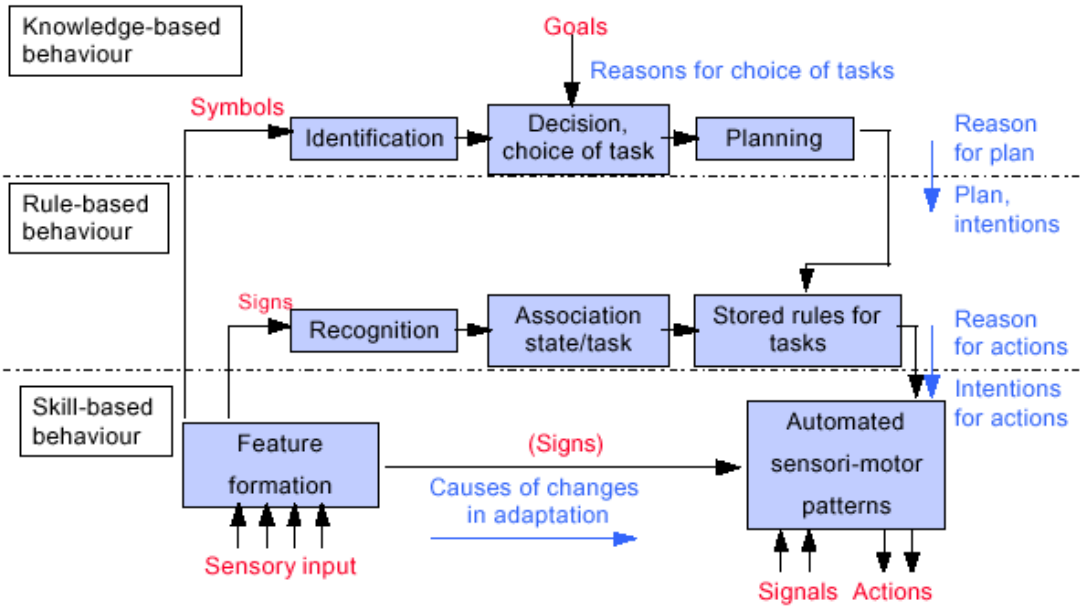

Figure 7 Levels of behaviour ([Ras86] and [KBD97])

2.2.3 Basic aviation knowledge and skills

For the pilot to have a proper understanding of the current state of the world, he has to evaluate several sources of information. As described in the previous section, humans have a limited supply of attention resources. In aviation, these resources are often used to full extent because of the complexity of the decisions and the amount of information the system (in this case the cockpit) has to show the pilot. For this reason the interface of the cockpit has a specific design to accommodate the pilot in obtaining information from the various gauges and instruments. During training, pilots learn to read the instruments in a prescribed scanning pattern in specific situations. For example, Figure 8 and Figure 9 show the instrument-scanning pattern for straight and level flight.

Figure 8 Scanning pattern for straight flight

In the figure above, the pilot needs to know if the flight path is still straight. To put it differently: the pilot has to verify the plane is not deviating from its current heading. The most important instrument in this circumstance is the attitude indicator or also called the artificial horizon. It provides the plane's current bank and pitch angle. When the bank angle of the plane is zero, the airplane is flying straight ahead. When the bank angle is checked, it is also necessary to see if the current heading of the airplane is still correct. It is possible, for example in case of crosswinds, for the aircraft to deviate from the selected course without actually making a turn. To verify the current course the pilot has to scan the heading indicator, which is typically located below the attitude indicator.

Figure 9 Scanning pattern for level flight

When in level flight, the plane is flying at one particular height. To make sure the plane is in level flight, three gauges have to be checked. The first and again most important one is the attitude indicator. As stated before, this instrument provides the pilot with the airplane's pitch information. In ideal circumstances, the altitude will not increase or decrease when the pitch is zero. Nevertheless, many external factors have to be taken into account when checking level flight. The occurrence of wind shear, for example, may influence the altitude of the plane. The altimeter, located to the right of the attitude indicator, provides the means to check the current height. But, unless the altimeter is checked frequently, the climb or descent rate is still unknown at this point. For that reason the Vertical Speed Indicator (VSI) has to be scanned also. If the VSI indicates zero, the plane is not climbing or descending. This completes the scanning cycle and the plane should fly level.

These are only two of many examples of basic tasks in aviation. These procedures are all stored in the long-term memory of the pilot in the form of acquired skills and knowledge. After extensive training and much experience, the basic procedures are performed almost automatically because these skills will be a second nature of the pilot.

2.3 Situation assessment

There is more to flying an aircraft than just looking at instruments, following certain prescribed regulations and procedurally executing motor commands. To know when to initiate specific actions, the pilot needs to be aware of what goes on inside and outside the cockpit. The awareness of certain (possibly harmful) situations already begins before take-off, during the pre-flight briefing.

2.3.1 Pre-flight briefing

Before taking off, a pilot must participate in a pre-flight briefing, which allows him to obtain important information about the intended flight. The most important aspects a pilot should be aware of, according to the Federal Aviation Administration (FAA), are weather condition, type of flight planned, aircraft identification or pilot's name, aircraft type, departure point, route of flight, destination, altitude(s), Expected Time of Departure (ETD) and Estimated Time En route (ETE) [FAA03].

U.S. DEPARTMENT OF TRANSPORTATION
FEDERAL AVIATION ADMINISTRATION

FLIGHT PLAN

(FAA USE ONLY) ☐ PILOT BRIEFING ☐ VNR ☐ STOPOVER | TIME STARTED | SPECIALIST INITIALS

1. TYPE	2. AIRCRAFT IDENTIFICATION	3. AIRCRAFT TYPE/ SPECIAL EQUIPMENT	4. TRUE AIRSPEED	5. DEPARTURE POINT	6. DEPARTURE TIME		7. CRUISING ALTITUDE
VFR / IFR / DVFR			KTS		PROPOSED (Z)	ACTUAL (Z)	

8. ROUTE OF FLIGHT

9. DESTINATION (Name of airport and city)	10. EST. TIME ENROUTE HOURS	MINUTES	11. REMARKS

12. FUEL ON BOARD HOURS	MINUTES	13. ALTERNATE AIRPORT(S)	14. PILOT'S NAME, ADDRESS & TELEPHONE NUMBER & AIRCRAFT HOME BASE	15. NUMBER ABOARD
			17. DESTINATION CONTACT/TELEPHONE (OPTIONAL)	

16. COLOR OF AIRCRAFT

CIVIL AIRCRAFT PILOTS, FAR 91 requires you file an IFR flight plan to operate under instrument flight rules in controlled airspace. Failure to file could result in a civil penalty not to exceed $1,000 for each violation (Section 901 of the Federal Aviation Act of 1958, as amended). Filing of a VFR flight plan is recommended as a good operating practice. See also Part 99 for requirements concerning DVFR flight plans.

FAA Form 7233-1 (8-82) CLOSE VFR FLIGHT PLAN WITH _____ FSS ON ARRIVAL

Figure 10 Example of a flight plan provided by the FAA

Most of these aspects are evident for the pilot. The flight plan, however, needs to be analysed carefully to ensure the pilot does not find himself into hazardous situations that could be prevented beforehand. In addition, when a pilot knows his flight plan well, the stress on the pilot during flight will decrease and the level of situation awareness increases because the pilot has to use (much) less attention resources for the interpretation of the flight plan.
A basic flight plan consists of a departure point, a destination point and the route to fly. Additional information, like the altitude(s) to fly the route, the amount of fuel left or additional waypoints, can also be specified in the flight plan.

2.3.2 In-flight

In order to accurately assess the current situation while flying, the pilot uses different techniques during the various stages of a flight. The stages can be described in a flow chart like the one in Figure 11. When analysing the importance of the instruments and gauges, it appears that in some stages the instruments are more critical, while in other stages the situation is assessed better by visual cues from the outside of the plane. For example, when the aircraft is at cruising altitude, practically all the pilot needs to check to verify the heading and state of the plane is the instrument panel. However, when landing the plane, the pilot is trained to look outside frequently to assess the safety of the situation.

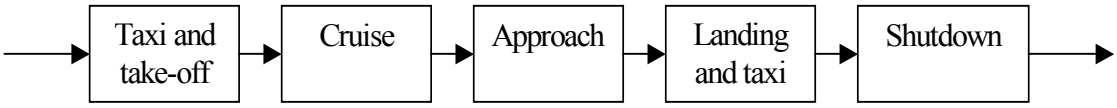

Figure 11 Stages of a typical flight

The importance of the various (instrumental and external) cues is deducted from the mental model the pilot has of the flight and the context of the current situation. A level of priority is given by the pilot to various stimuli, which involves the use of the attained knowledge and skills from training and experience. In other words, the assessment of the current situation depends on the knowledge and skills in the first layer of the model of the abilities of a pilot (see Figure 5). Based on the gathered information, the pilot can formulate a hypothesis about the current situation and take proper actions.

Much of the information a pilot has to assess is probabilistic. For example, before the flight is started weather services may predict a 30% chance of rainfall in the area of the flight. This may be corrected during flight by visual cues (i.e. clouds building in the direction of the flight path) and this may result in the pilot initiating a change of heading. The logic required for the assessment task can also introduce errors in judgment as it is based on probabilistic sources. For that reason special care must be taken when dealing with statistical information, as 98.6 percent of all people know.

2.4 The decision making process

The judgment and decision-making of a pilot gives him the capability to reach the intended destination safely, presuming the pilot has trained these capabilities well. The decision-making process is a complex cognitive task and depends strongly on a flawless mental model of the current state of the world and the assessment of the informational cues gathered from the instrument panel of the plane and the view of the outside world.

2.4.1 Situation recognition and situation awareness

Pilots are trained to make decisions in a procedural way: assess the situation, find all useful respond actions, evaluate these actions based on certain criteria, then execute the best responsive action. This is a relatively simple and effective approach for training, but it takes a lot more experience to do this under real life conditions. Often when flying, a pilot has little or no time to make a decision. Therefore, many decisions are made without assessing all available information.

Humans have an innate capability of recognizing and generalizing resembling patterns or situations. Quite often, the pilot recognizes a certain situation from previous encounters and acts in the same way as before. If the current situation differs (slightly) from the past one, the outcome of the old course of action may lead now to dangerous situations. Therefore, the pilot must change his understanding of the current state of the task and must act accordingly. This is an effective way of making decisions.

There are, however, some disadvantages of this approach. When the pilot does not take all information into account, it is imaginable that in certain circumstances the situation varies in a not detected way from the situation the pilot thinks has occurred. If the difference is too great, the safety of the airplane is in jeopardy, because, although the pilot will take the correct responding action, the decision is based on wrong presumptions.

2.4.2 Risk assessment

The most interesting part is when an unprecedented situation occurs. In such a case, the pilot has to involve his knowledge to come to a decision. This is explained by Wickens in his model of decision-making [WiNa88].

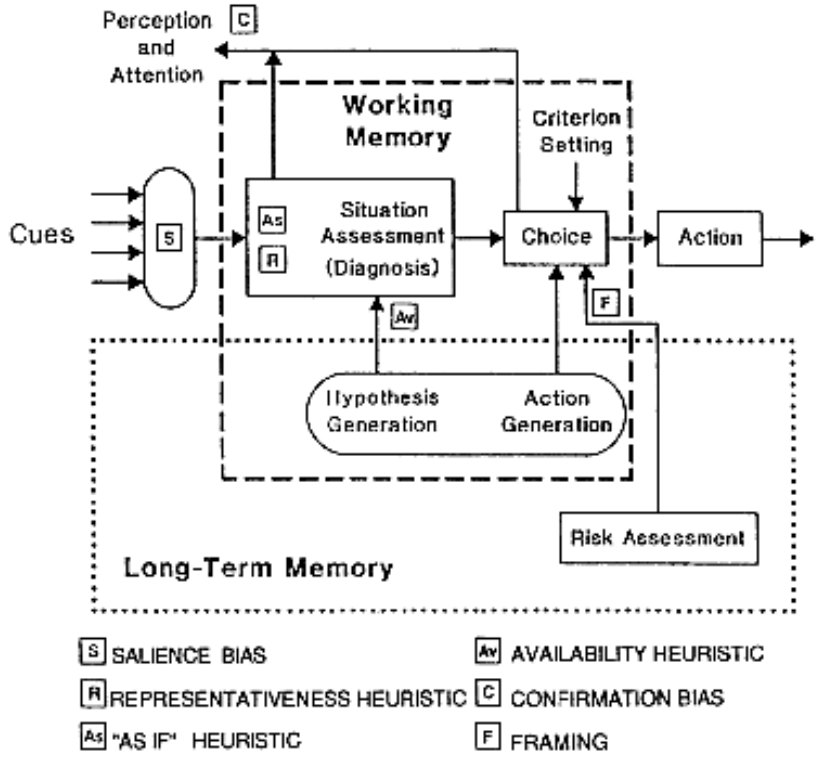

Figure 12 Wickens' model of decision-making [WiNa88]

After the informational cues are processed, the current situation is diagnosed, resulting in a choice of the best course of action to take. In addition, this decision is influenced by the risk assessment, which is made by the pilot to be able to maintain a certain level of safety during the flight.

The risk assessment is the pilot's subjective evaluation of the probability of certain outcomes. The evaluation can differ from person to person, because it depends on many sources that are probabilistic. Firstly, the skill to generate as many problem-solving courses of action as possible is not an innate ability of humans. It requires training and not everyone has the required talent to do this. Secondly, when calculating risks, people have the tendency to overestimate the frequency of very rare positive events. Furthermore, when people have to choose between two losses, the risky (potentially dangerous) loss is seen as a better alternative than a certain (less life-threatening) loss. This may be explained by the fact that people often have overconfidence in their ability to predict near-future events. This can sometimes result in disastrous situations as can be seen in Figure 13. All these facts put together make the work to model a real pilot a tedious and difficult job.

Figure 13 Sometimes judgement errors are made by the pilot

2.5 Conclusion

The information presented in this chapter can be used as a starting point for the *flight bot* that is modelled in the course of this project. It is the basis for the models described in Chapter 4 and Chapter 5. However, this chapter has a certain point of view towards flying an aircraft. It is not meant to explain all relevant aspects of, for instance, choosing certain actions while flying by the *flight bot*. Flying is too complicated to describe in one or two chapters or even for the duration of this project. Implementing these aspects in a consequent manner provides a solid platform for the *flight* bot to start with.

Chapter 3 Analysing the flight plan

3.1 Flight plan requirements

When a pilot has to make a flight, a flight plan is provided to mark all special events during the intended flight. This way the pilot knows beforehand exactly what can be expected to occur while flying. In order to generate a proper flight script for the *flight bot* to execute, the entered flight plan has to be analysed carefully. Because the *flight bot* cannot create a flight plan on its own, it has to be provided by a supervisor.

3.1.1 Content requirements

As is stated in section 2.3.1, a minimal flight plan consists of multiple steer points and additional information about the circumstances of the flight and information about the aircraft. Not all supplied information is required for the *flight bot* to know. In addition, some information is ignored because of the focus of this project. For example, the type of aircraft can be chosen, but only the Cessna 172SP Skyhawk is selectable for the user. In addition, the amount of fuel is considered unlimited, which is not realistically, of course. This has been done to keep the project manageable. A selection has been made to specify what information the supervisor has to enter as a flight plan. The most important information is the route for the bot to fly. It consists of the airport of departure, possibly multiple steer points and a destination airport. Each element has its own attributes. For every steer point, the following attributes are required:

- GPS coordinates (longitude, latitude)
- Time Over Steer point (TOS)
- Type
- Action to take at the steer point
- Airspeed
- Heading
- Altitude

Airports have, in addition to the attributes of a steer point, a name, an abbreviation, a runway number from which to depart or land and the general GPS coordinates of the airport.

Some of the steer point attributes have priority over others. For example, the airspeed is calculated by means of the current weather information. However, when a change in weather occurs, it may be necessary to adjust the airspeed to reach the steer point at the designated TOS time. In addition, the TOS must be checked for feasibility.

The type of the specified steer point can be chosen from an airport location or a GPS coordinate. It is possible to have zero or more navigational points along the flight path. These are specified using GPS coordinates. However, in a flight plan only one airport location may be entered as a start point and one airport location as an end point. These points have additional attributes that tell the bot how to behave near the airport.

3.1.2 Constraints

After acquiring the flight plan, it has to be checked for validity and feasibility. Certain constraints must be set to make the flight plan a reachable goal. The restrictions can be categorised in aircraft, environmental and pilot constraints. Because of the overwhelming amount of constraints in real life, not all restrictions can be examined. A selection of the most important constraints has been made to verify the flight plan for the most obvious errors.

Most aircraft constraints are rather easy to check. When analysing the flight plan, the following list is a set of restrictions, which is imposed on the flight plan by the nature of the aircraft.

- Maximum altitude
- Maximum airspeed and throttle
- Minimal airspeed (also known as stall speed)
- Maximum pitch angle
- Maximum banking angle
- Fuel tank capacity

The aircraft constraints are, naturally, dependent on the type of aircraft used. For this project the Cessna 172SP has been chosen to accomplish the goal. The limits of this particular airplane are listed in Appendix B.

In addition, other restrictions are imposed by the environment in which the flight takes place. In certain circumstances, it is not safe to fly in a certain area. It is preferable to take another route, if the area can be avoided. There are two kinds of environmental constraints, which have to be examined. The first is the weather report. When a storm is approaching at a certain navigational point, the pilot has to stay clear of that area. Right before take-off, a pilot is always briefed about the current weather conditions and forecast. Secondly, some areas are defined as a no-go zone. Special care has to be taken at, for example, the airspace above airports. The air traffic at airports is always lively because of the planes that land and take-off. It is preferred to stay clear of all airports, which are not the destination of the flight plan. There are also areas, which are prohibited by authorities for ‘civilian air traffic’. A well-known and much used example is Area 51 in the Nevada desert.

Figure 14 Area 51 near the shore of Groom Dry Lake

Finally, pilot constraints are typical constraints for all human beings. Examples are the limited attention resources (e.g. not all instruments can be scanned and interpreted at one time) and the limited physical abilities (most humans have, for example, only one pair of eyes). Most of these constraints cannot be checked during the analysis of the flight plan, but only in-flight. This is the case when the pilot suffers from fatigue. Some restrictions may be predicted by analysing the stress at a particular moment in the flight plan. When many actions must be performed at a steer point, the workload on the pilot increases and may exceed the limits of a human pilot. In that case, a warning may be issued beforehand about that particular situation.

3.2 Flight procedures

During analysis of the flight plan, the flight path has to be broken into several smaller pieces, which resemble the stages of a flight as is mentioned in Figure 11. The stages, also called phases of the flight, can in turn be executed in a procedural manner by calling certain predefined basic tasks. These tasks are stored in a knowledge base and are typically situation independent. When dealing with the flight script on a per-stage basis, the constraints set upon a particular stage can differ from the next stage. This way, a distinction can be made between the various stages.

The tasks were defined using the ground school-manual from Microsoft's Flight Simulator 2002 [Mach02]. This manual has been set up to simulate the real thing as best as possible. In addition, it has been written by a real-life pilot instructor, who gives, besides the standard information about flying in Flight Simulator 2002, additional information about the points where Flight Simulator differs from reality.

3.2.1 Basic commands

A layering can be applied to the tasks stored in the knowledge base. The first layer contains the commands that interact directly with the cockpit's interface. These commands may be seen as atoms from which the flight procedures are built. The following list is an enumeration of the basic commands, as detailed in Appendix C.1.

1. Change attitude
2. Change pitch
3. Change power settings
4. Change mixture settings
5. Trim
6. Change flaps
7. Change rudder to keep inclinometer in the central position
8. Set switch or knob of particular instruments

3.2.2 Flight procedures

With the basic commands in mind, complex flight procedures can be built by combining multiple basic commands in a particular order. The enumeration of the flight procedures mentioned below is worked out in Appendix C.2 in the basic steps that have to be executed in chronological order.

1. Take-off
2. Fly level
3. Fly straight
4. Turn
5. Climb (or descent)
6. Approach runway
7. Land

To provide an example, one of these tasks, the "turn" procedure can be worked out as follows:

1. Read out attitude indicator and heading indicator.
2. Calculate difference to the specified target heading. If difference is greater than zero, increase aileron control input value smoothly. Decrease aileron control input value smoothly when difference is smaller than zero.
3. *Change attitude* to +20 degrees if difference is positive. *Change attitude* to –20 degrees if difference is negative.

4. *Change rudder to keep inclinometer in the central position.*
5. Read out attitude indicator and heading indicator.
6. Calculate difference to the specified target heading. If difference is greater than the default margin, go to step 4.
7. *Change attitude* to zero degrees.
8. *Change rudder to keep inclinometer in the central position.*

Based on these procedures, it should be possible to create a flight plan for a plane to take-off from an airport, fly according to a flight path and land at the destination airport, assuming the proper conditions are known. Considering the advanced procedures, tasks that are even more complex can be accomplished by combining them into an even higher-level procedure. Examples of these kinds of procedures are flying a traffic pattern or navigating by VOR (Very high-frequency Omni directional Range) beacons.

3.3 Generation of the flight script

While analysing the flight plan, a script is generated for the *flight bot* to execute. This script consists of a list of advanced procedures to be called in chronological order. In addition, some procedures may be executed concurrently. For instance, while making a right turn, a descent may be initiated. With this enumeration of procedures it should be possible, in theory, to fly the whole flight plan under ideal conditions. The way the script evolves can be compared to a hierarchical structure. While the script is generated before take off using the XML knowledge base, not all details are filled in yet, because some parameters may not yet be available. During flight the script evolves to a more detailed plan, which is read and executed by *flight bot*.

3.3.1 Substitution algorithm

There are many ways to generate a script from a list of tasks. The usability of a certain technique depends on the nature of the actions and on the level of abstraction in which the actions are defined. One solution is to use an algorithm that processes a list of actions and substitutes every action with (possibly multiple) more detailed tasks. This process is repeated to get on the level of flight procedures.

The following example illustrates the way a substitution algorithm works. At the start there is a list of steer points with their individual actions. The flight plan is divided in multiple phases. The actions are used as a seed and extended into complex tasks and procedures with their respective parameters according to the phase the steer point is in. For example, the *land* action consists of an *approach* procedure and a *land* procedure, which are all part of the *land* phase. The list of all the flight procedures is the basis for the flight script and the input for the *flight bot*.

3.3.2 Example of a flight script

To give an example of a flight script, a flight is planned from the airport of Soesterberg (EHSB) to Schiphol (EHAM). While analysing the acquired flight plan, the following script is generated:

Nr	Procedure	Parameters	Task
1	Take-off from EHSB at runway 27	EHSB, runway 27: GPS coordinates 52° 7' 37" N, 5° 17' 56" E, heading 271°	Take-off
2	Climb	Altitude: 3500 feet	
3	Make a right turn	10° bank to heading 300	Cruise
4	Fly level	Altitude: 3500 feet	
4	Fly straight for 21 nautical miles towards airport EHAM	EHAM: GPS coordinates 52° 18' 31.01" N, 4° 45' 50.00" E	
5	Descent	Altitude: 1000 feet	Approach
6	Approach and align with runway 9	EHAM, runway 9: GPS coordinates 52° 19' 6" N, 4° 47' 49" E, heading 269°	
7	Land on runway 9		Land

When feeding this script to the *flight bot*, it will call at every step the appropriate procedure with the proper attributes. In ideal circumstances (e.g. no weather changes or other air traffic nearby), this script would be easy to execute. However, most of the time, the flight conditions are far from ideal. This calls for an agent, which constantly monitors the flight. This agent must be able to correct the flight script when necessary during the process of flying. This process will be explained in the next chapter.

Chapter 4 Designing the flight bot

The cognitive model worked out in Chapter 2 has to be implemented in the bot in order to mimic human behaviour and decision-making in a cockpit environment. Based on what has been stated about real-world pilots, certain aspects can be translated in a cognitive model for the autonomous bot.

4.1 Layered approach

As is indicated in Chapter 2 (see Figure 5), the capabilities of a human pilot can be split up in three layers. The first layer contains the basic knowledge used by the pilot to navigate the airplane. The second layer consists of the actual navigation and the short-term decision-making process to properly follow the flight plan. The third layer is the ability to anticipate to events that may occur in the future.

When applying the model from section 2.1 to an automatic *flight bot*, the three layers form a basic framework for the implementation the bot. The red layer is the part of the flight bot that is the knowledge base of flight procedures and regulations. The segment of the model in Figure 15, which is inside the box with the dashed lines, represents the part of the application that actually steers the airplane (the *flight bot*) and another part that anticipates on future events and changes the flight script accordingly (the *co pilot*).

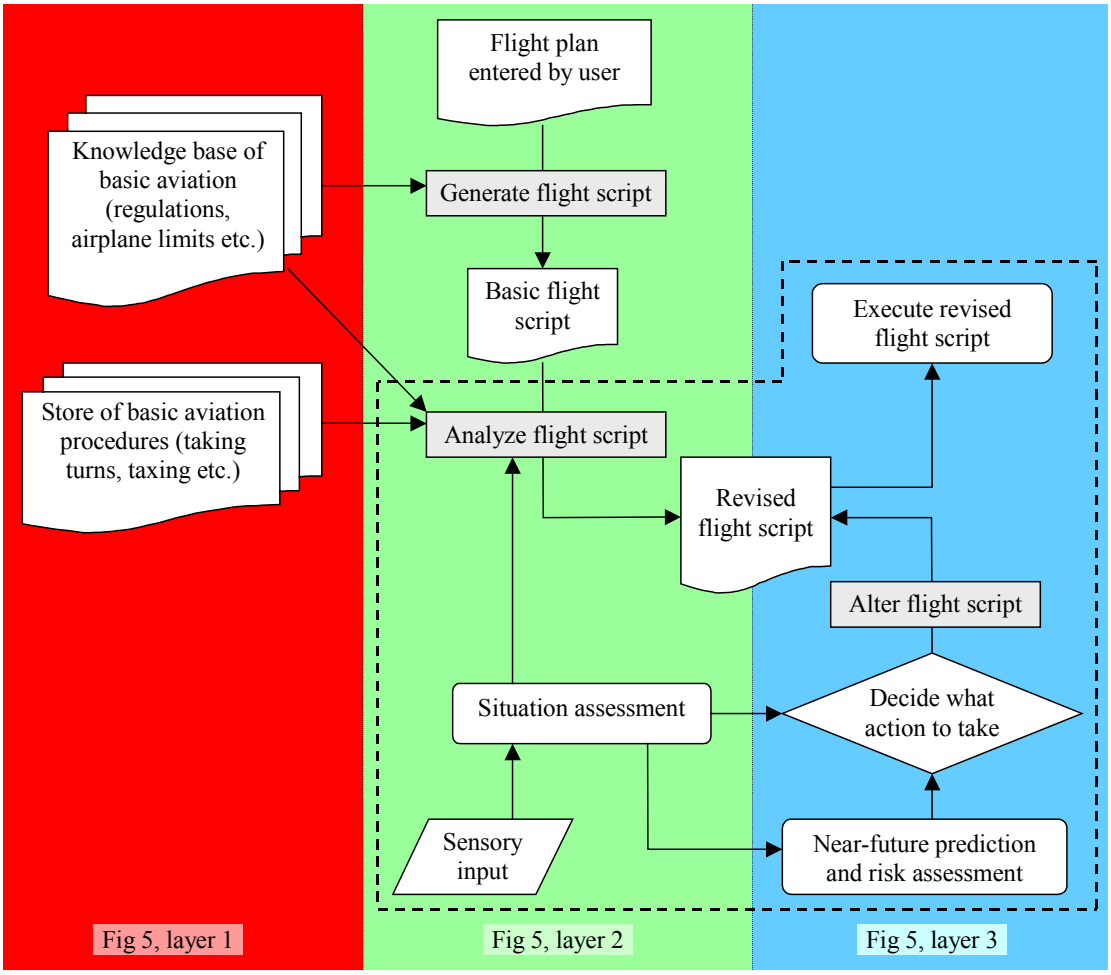

Figure 15 Design of the flight bot using a three-layered approach

4.1.1 Expert system and procedures

The basis for proper flying is the use of advanced skill and knowledge about procedures and regulations in aviation. This is not that complicated to implement when using an expert system with a knowledge base. The knowledge base consists of the standard aviation regulations. For example, when flying according to the Visual Flight Rules (VFR), there are rules about the minimal and maximal altitude to maintain. In addition, the skills can be automated by using generic flight procedures, which can be called during flight to execute a certain flight manoeuvre. These scripted procedures have to be generic, because of the many parameters such a procedure depends on. A simple manoeuvre as taking a turn already needs several parameters, like the banking angle, the throttle setting, the climb or descent rate and so forth. Special notion has to be taken of the interdependency between parameters. For example, to maintain airspeed while climbing, the throttle has to be increased.

4.1.2 Flight plan analysis and situation assessment

The knowledge base from the first layer can be used to analyse the flight plan. The analysis can take place in two stages. Firstly, the basic flight script should be generated according to the standard flight path, which is stated in the flight plan. After this, the flight path should be reviewed for hazardous situations or obstacles, which are in the flight path. If such a situation occurs, a change should be made to the flight script to anticipate to these circumstances. When an adjustment has been made, the whole process of reviewing the script should be repeated.

Figure 16 A Cessna flying over normal ground traffic

After the analysis, the bot can fly the flight script. While doing this, the bot should be aware of the current situation to accomplish the goal of the flight plan. When a condition arises where the bot has to take action, it should be able to properly formulate a hypothesis about the current situation. This will pave the way for taking the correct measures in the top layer, because it is based on correct information from the second layer.

4.1.3 Near-future prediction and planning adjustment

When certain conditions are met that can trigger a possible harmful situation, there are in general several courses of action possible to prevent the situation from happening. These conditions should be checked at a regular interval by the bot. Choosing which action to execute depends on many factors, like urgency of the situation, complexity of the task at hand, regulations, commands from external sources (for example Air Traffic Control), weather conditions and airplane safety.

It is beyond the scope of this study to model all possible alternatives in every situation, taking all conditions into account. Real-life pilots fly for a major part on experience from previous flight. The bot should also maintain the flight history for this reason. It must be able to perceive if the current situation is comparable to previous ones and verify if the outcome was satisfactory. If so, the *flight bot* will reason which course of action can be taken in the current situation depending on the list of comparable situations.

Risk assessment is another important criterion for selecting the appropriate course of action. The current situation can be diagnosed, based on a knowledgebase of fuzzy or probabilistic information about the risk of taking a certain action in the current situation.

4.2 Human limitations for the flight bot

For the situation and risk assessment it is necessary to process the cues the instrument panel provides about the current situation of the airplane. While a computer can read and process information very fast, humans cannot perform this task as effective. Because this project has the goal set for the bot to resemble the flying behaviour of human pilots, the bot has to be programmed to act and respond generally the same way real pilots do. This has certain implications for the bot.

- It has to acquire instrument readings the same way pilots do with instrument scanning patterns, also taking into account the speed and accuracy pilots use when reading instrument panels.
- The bot has a limited amount of attention resources, like humans do, and must allocate them for a period to particular tasks.
- Some kind of memory has to be implemented to recognize patterns and situations from previous encounters.
- The knowledge stored in the (long-term) memory can also be used to prioritise or filter certain informational cues while processing them.
- When risk is involved, generate a set of solutions or alternatives, limited to the skill of an expert pilot.

Chapter 5 Application design and implementation

To successfully create an application, a design must be made to give a clear notion of the workings of the intended application. As an overview of the total application is presented, a better understanding is available to the programmer of the amount of time and resources needed for the task of implementing the application. For this reason several techniques have been invented.

5.1 Unified Modelling Language

The Unified Modelling Language (UML) is a methodology created by Grady Booch, James Rumbaugh, and Ivar Jacobson to define the inner workings of an application. It uses models to make an abstract representation of a design or a system from a particular point of view. These models are often visualised in a diagram [PoSt99]. There are various diagrams defined in the Unified Modelling Language.

- *Class diagrams* are used to document the static structure of the system. The system consists of multiple objects. A class describes a set of objects with an equivalent role or roles in a system. In the class diagram, every class is defined and the relation between the objects is displayed. Classes can be seen as the nouns in the description of the system in natural language. Associations, or relations, correspond to the verbs in the same description.
- *Use case diagrams* document the behaviour of the system from the user's point of view. The user is an external entity, which interacts with the system.
- *State diagrams* show how an object in a system responds to a received message, or other event. It uses states, transitions and events to visualise the workings of the objects.
- *Collaboration diagrams* are the diagrams that show the interaction between the classes from the underlying class model. It allows detailed recording of the way objects interact to perform a certain task.
- *Sequence diagrams* show the flow of objects and actors that take part in a certain task. Every object has a lifeline, which is the time the object is activated for a certain procedure.
- *Activity diagrams* can be used to describe the way in which a procedure should be implemented and coordinated. The essential dependencies between operations can be modelled more clearly in this kind of diagram.

There are even more types of diagrams in the Unified Modelling Language that are more obscure, but not all diagrams are necessary to be used. The most important criterion why a certain kind of diagram should be used is the question if using that particular diagram enhances the view on the design of the entire system.

5.2 Application architecture

While the structure of the *flight bot* itself has been dealt with in Chapter 4, the application as a whole is more than just the *flight bot*. It has been briefly mentioned in the introduction that the application is divided into four separate stages. These tiers will be worked out in the following sections. An overview of the application is given to clarify the design of the program in Figure 17.

The class diagram speaks mostly for itself, but a few things are worth noting. The *flight planner GUI* is the graphical user interface (GUI) of the application. The supervisor will interact with this part of the program. The *flight planner GUI* will also pass commands of the person supervising the flight bot to

the rest of the application. Furthermore, the *flight plan consistency checker* is drawn in grey, which means it is not implemented yet in this project. Only a shell for the module is implemented, which always returns *true* even if the flight plan is not consistent. This part of the program is left for future development. This is also the case for the *co pilot*, which monitors the execution of the flight and adjusts the flight script real-time to reach the appointed goal. The *flight bot* executes the actual flight procedures described in the flight script. Finally, the flight plan and the flight script are special classes, because they are in essence only an XML document.

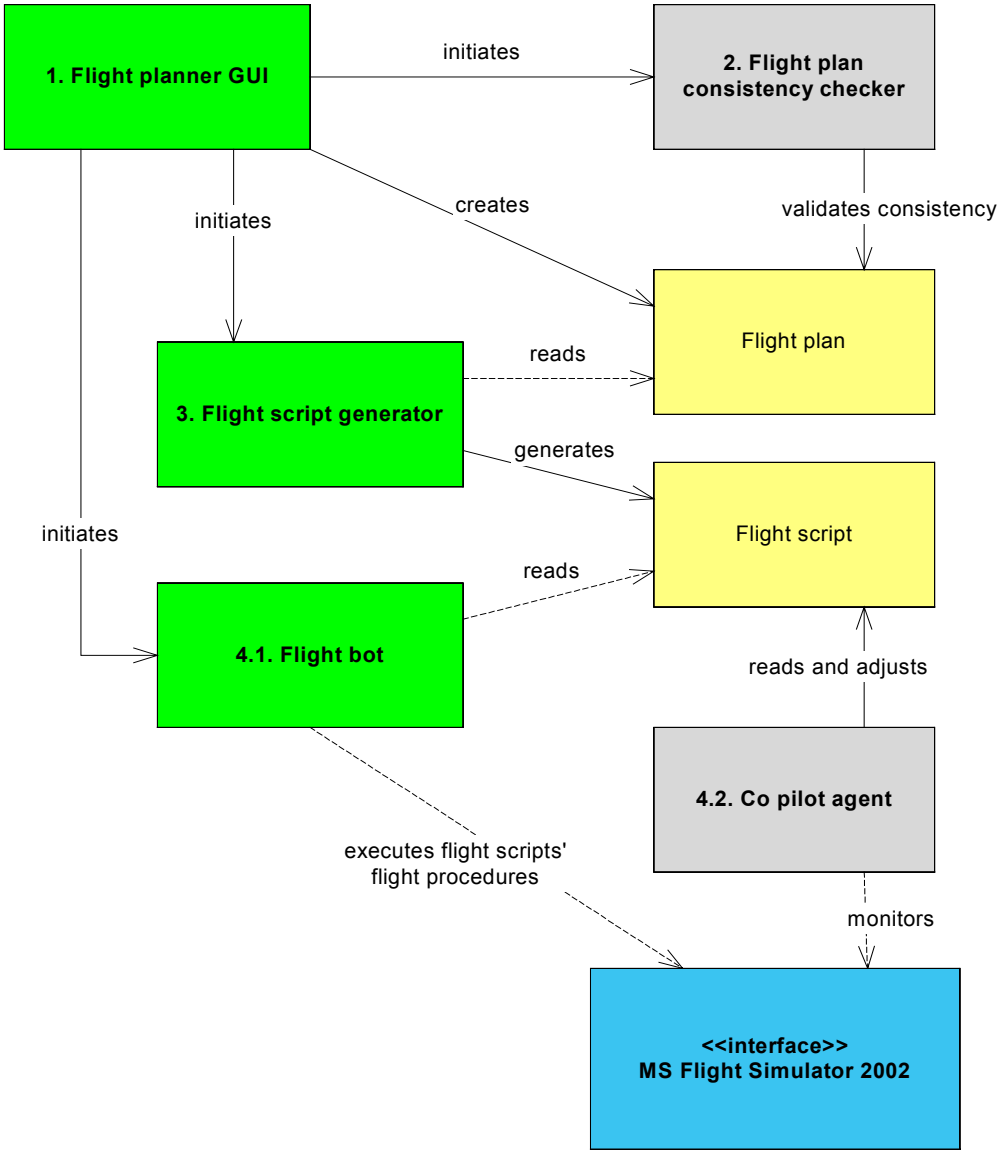

Figure 17 A class diagram of the complete application

The interface with Microsoft Flight Simulator 2002 is a special module called FSUIPC, written by Peter Dowson [Dow03], which is installed in Microsoft Flight Simulator. The module supports reading and writing variables from an external application by a method called Inter-Process Communication.

5.2.1 Flight planner GUI

The first part of the application is also the part that interacts with the supervisor. It is called the graphical user interface (GUI) of the application. The simplest way to model the specifications of a user interface in UML is by using *use case* and *sequence* diagrams.

Figure 18 describes the use cases of the *flight planner GUI*. There are four actions a supervisor can do in the interface. Firstly, the supervisor can enter the flight plan into the *flight planner*. After this has been done a consistency check must be done to validate the entered flight plan. When the flight plan has been proven to be correct, the supervisor can initiate the flight script generation. Then, the supervisor can give the command to feed the flight script into the *flight bot*, which executes the flight actions in the flight simulator.

It is apparent that these four use cases depend on each other in a strong fashion. The *flight bot* cannot execute any manoeuvres while the flight script has not been generated. A condition for the generation of the flight script is the fact that the flight plan must be consistent. And finally, when a flight plan has not yet been entered, it cannot be validated by the application.

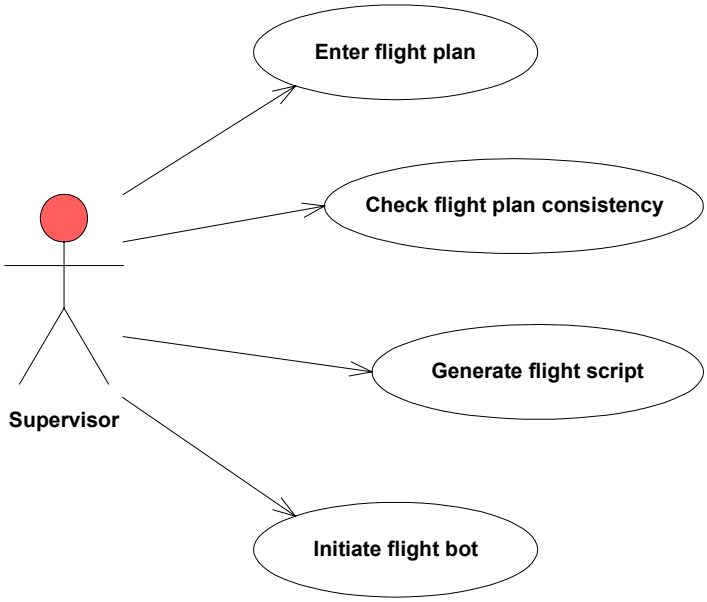

Figure 18 Use case diagram of the flight planner GUI

The available use cases are worked out in the next sections in sequence diagrams, which denote the sequence of the available actions the user can perform. One use case, namely *initiate flight bot*, has not been worked out because of the straightforwardness of the use case. When the application has generated the flight script, the user can initiate the *flight bot* by giving one command. The *flight bot* itself will begin flying the flight script and there will not be any interaction with the *flight planner GUI*. In that case, a new stage of the application has been entered and the user will have feedback directly from the *flight bot* itself. The flight bot itself has been worked out in a separate chapter.

5.2.1.1 Enter flight plan

For a user to be able to enter a flight plan, several features must be provided by the user interface. In the first place, a way to actually collect the steer points must be implemented. This is implemented in the application in the shape of a display for so-called NOS/GEO sectional charts. Sectional charts are

used in the aeronautical domain to visualise the flight route. The format of the files in which the charts are stored is called NOS/GEO. The user can open such a chart in the application and add steer points to the flight plan by pointing out the various coordinates on the map.

When the supervisor has added the steer points, he can choose to save the flight plan to be able to retrieve the flight plan on a later date. Changing or removing steer points are also features the supervisor must have to correct possible mistakes while entering the steer points.

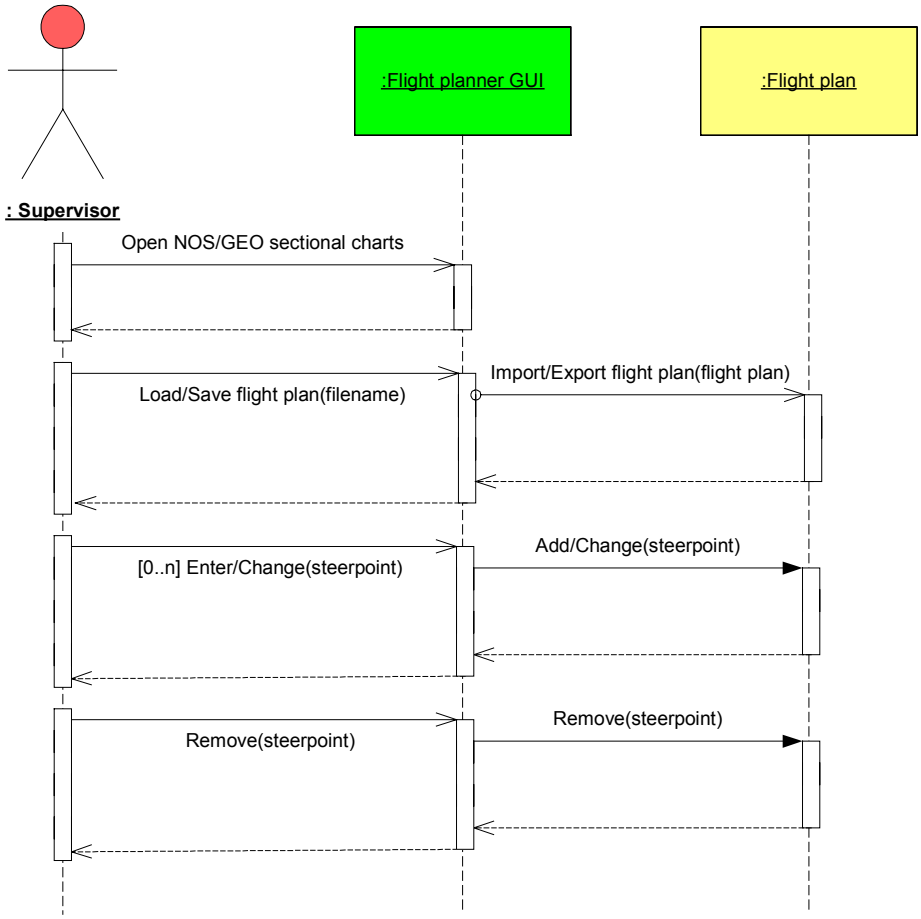

Figure 19 Sequence diagram of use case Enter flight plan

5.2.1.2 Check flight plan consistency

The sequence diagram (Figure 20) of the use case *Check flight plan consistency* is fairly obvious. The primary functionality of this module is located at the message whether the “Flight plan is valid”. To answer this question, many aspects of the flight plan must be researched by the application. Because of the time constraints of this project, this part of the application has only been implemented to return a positive answer. A shell or shell of this module has been made for future work to provide the possibility to extend this feature. This module already incorporates an XML database with the specifications (or limitations) of the aircraft used by the *flight bot*. In addition, the general manner the module is intended to work is described in section 5.2.2.

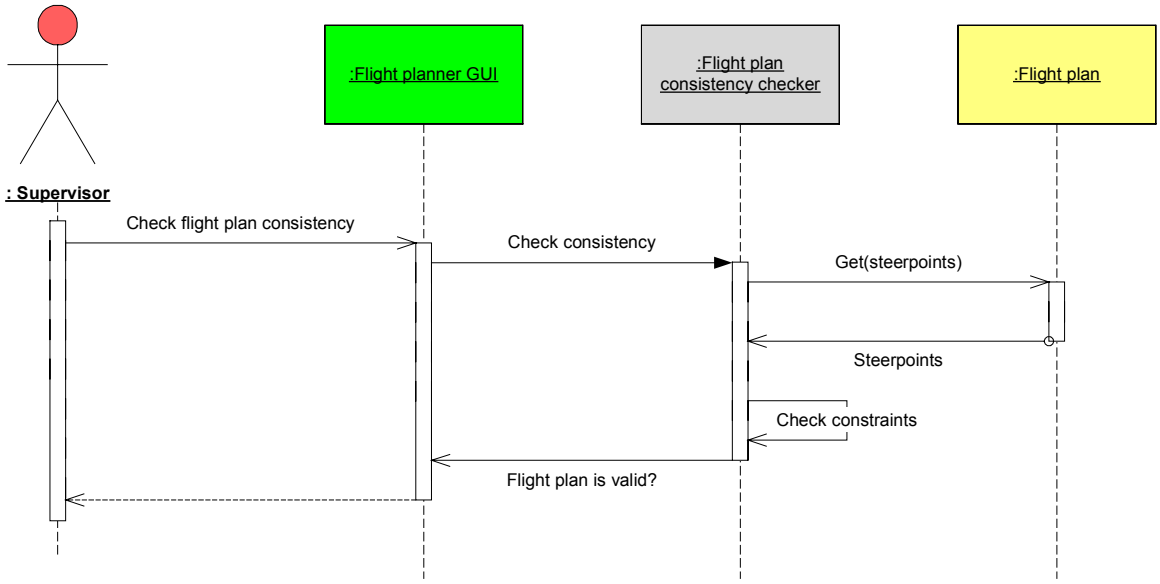

Figure 20 Sequence diagram of use case Check flight plan consistency

5.2.1.3 Generate flight script

The *Flight Script Generator* goes through several phases before the flight script is assembled. Firstly, the application makes sure the flight plan has already been validated. If this is not the case, the possibility exists the flight plan is inconsistent and thus a flight script cannot be created from this flight plan.

A valid flight plan has been given at every steer point at least one task. For every task in the flight plan, the flight script knowledge base is called to search for a corresponding task in the database. The task of the knowledge base has been defined using flight procedures, which are the base procedures the *flight bot* executes chronologically. Finally, when the flight procedures are added, the parameters for the procedures are extracted from the flight plan and appended to each procedure.

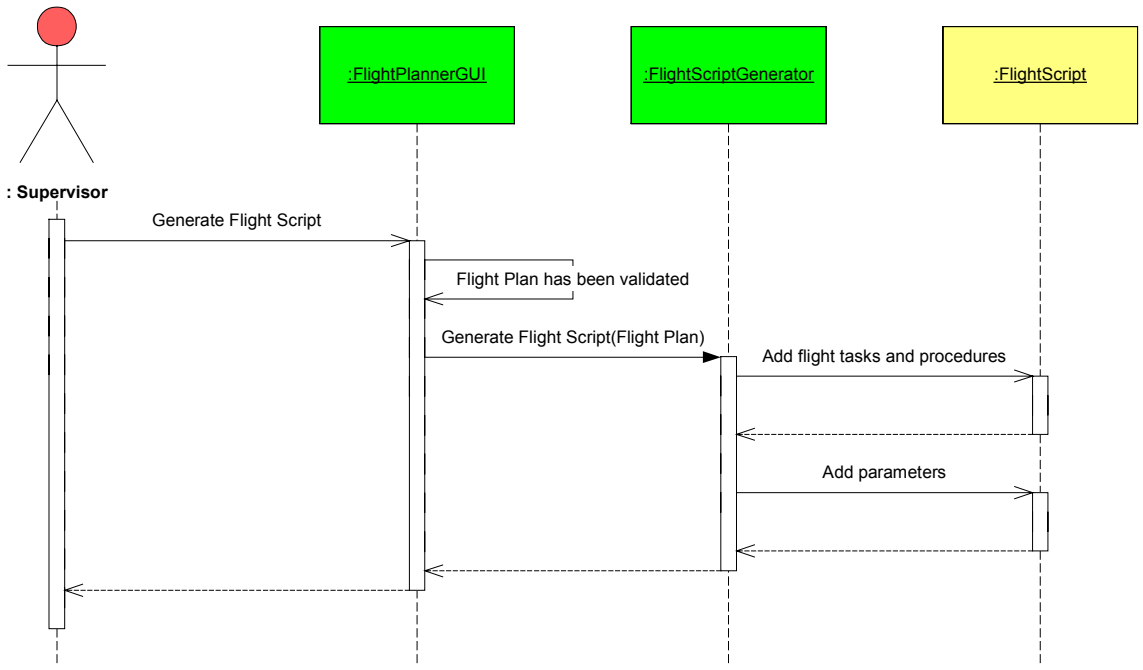

Figure 21 Sequence diagram of use case Generate flight script

5.2.2 Flight plan consistency checker

Although the *flight plan consistency checker* will not be fully implemented in the scope of this project, a principal design of this module is given in Figure 22. See section 3.1.2 for more information about the types of constraints.

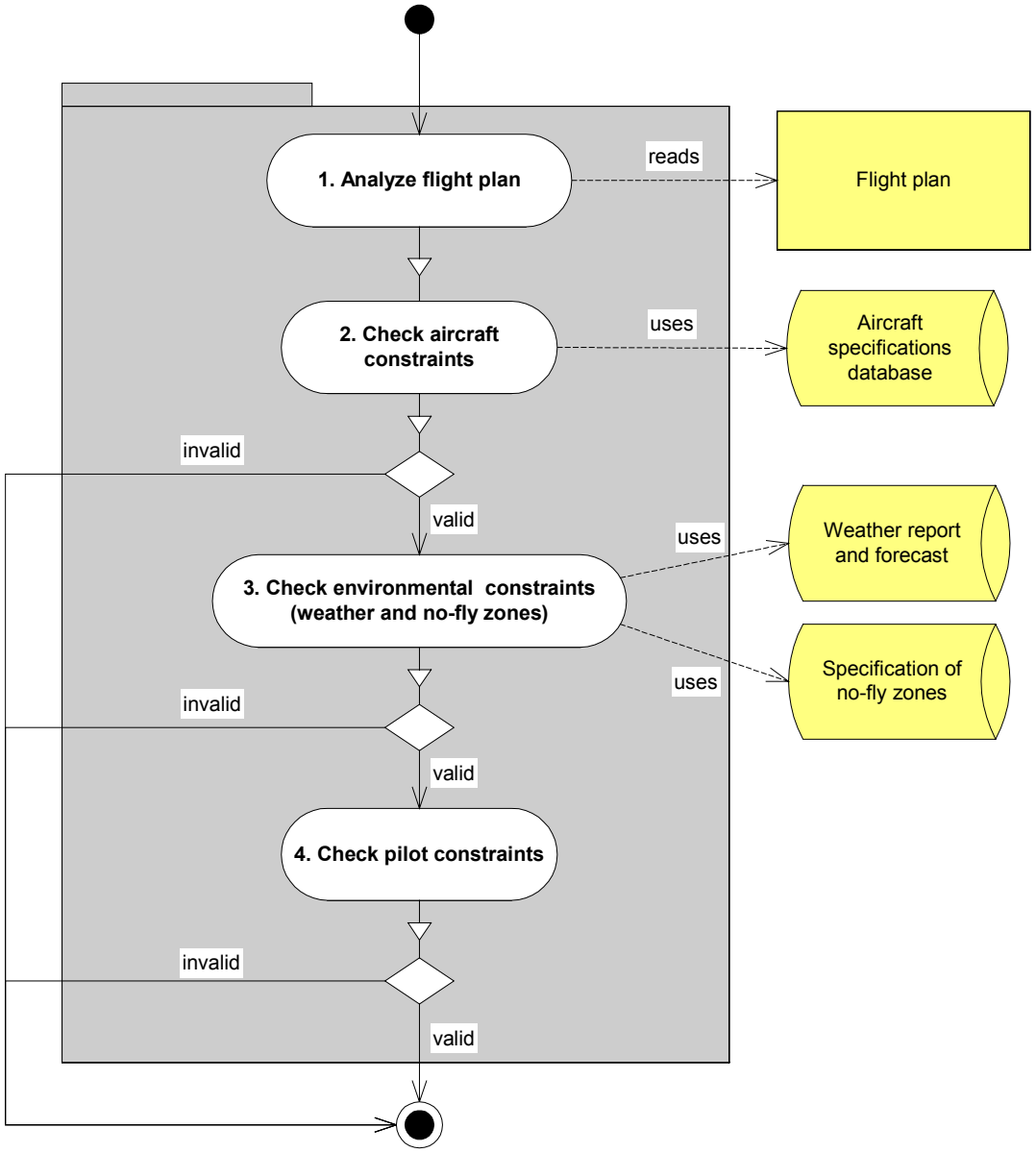

Figure 22 Activity diagram of the flight plan consistency checker

5.2.3 Flight script generator

The following figure is a model that represents the internal activities of the *flight script generator* module. Before the *flight script generator* starts, it makes sure the flight plan is checked for validity. Then the module begins with reading the flight plan and the flight script tasks and procedures database. The database contains the standard tasks the *flight bot* must be able to perform. Every task is defined by a chronological list of flight procedures.

The generator adds all tasks that are mentioned in the flight plan into the flight script and appends the corresponding flight procedures for every task. After this, the parameters are added to the list of flight procedures in the flight script. When this is done, the flight script is ready for use by the *flight bot*.

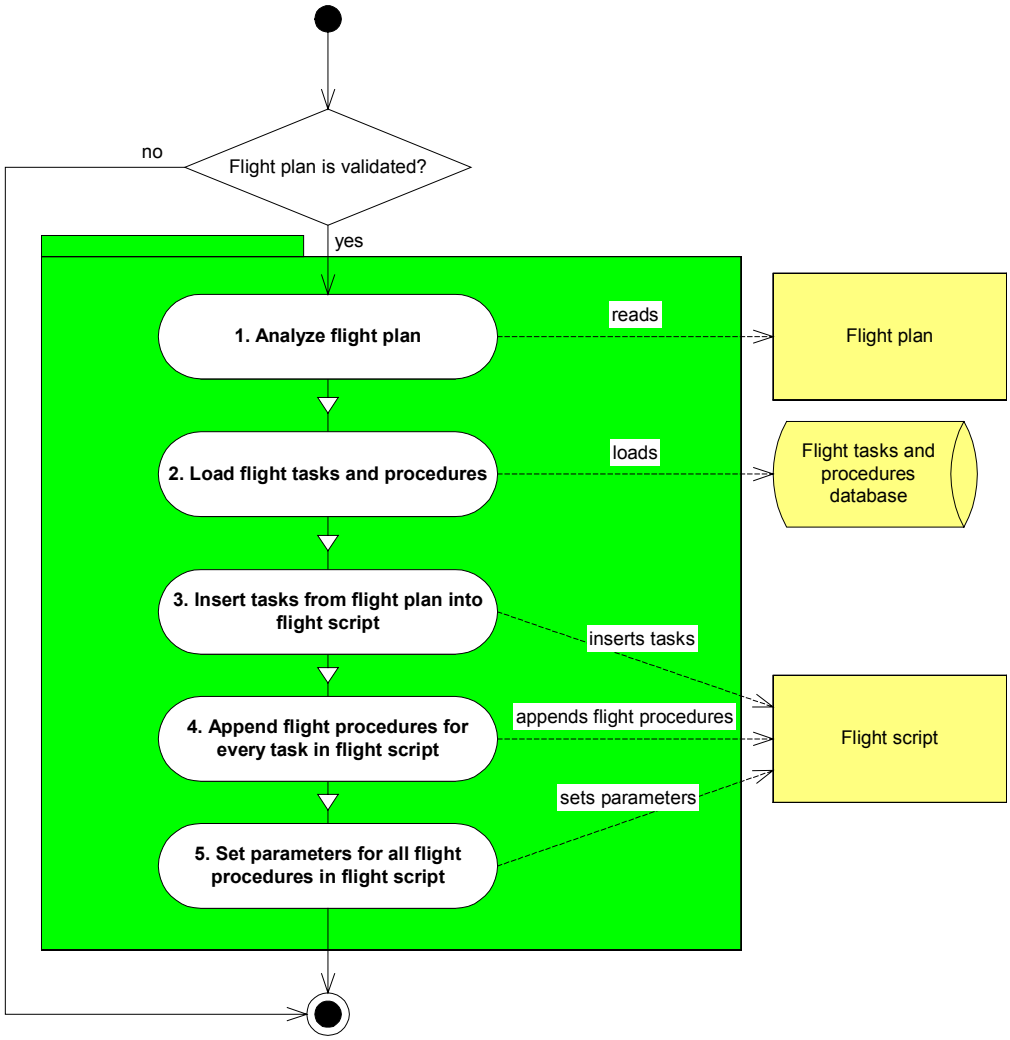

Figure 23 Activity diagram of the flight script generator

5.2.4 Flight bot and co pilot

In Chapter 4 a design is proposed for the *flight bot*. This design gives a general impression of the inner workings of the bot. The next figure is a procedural flow diagram of the *flight bot*, taking into account the points noted in Chapter 2 and Chapter 4.

The *flight bot* and the *co pilot* can be seen as the second and the third layer of Figure 5. The second layer of this figure can be described as the layer where all the action is. Here, the pilot executes the basic actions to fly an airplane and this is the place where the pilot's training is being used. In the third layer all the predictions about the future are made. Now, the experience of the pilot from previous flights is used. The pilot has built up in his long-term memory a vast knowledge base about what actions to take in certain (hazardous) situation and how to anticipate on those situations.

The *flight bot* works on a short-term basis. It reads the flight script and extracts the task and flight procedure that should be executed at the current time. This should be done periodically (i.e. not just in one try at the start of the flight) because the *co pilot* may have altered the flight script during the current flight. Hence, to keep track of those changes, the *flight bot* should always get the most up-to-date information available from the flight script.

At the beginning of a flight, the aircraft must be positioned at a certain location, which is mentioned in the flight script as the runway or airport from where the take off must take place. Then it enters a loop in which the bot will consecutively execute all flight procedures that are read from the flight script. Every flight procedure has a (short-term) goal, which must be reached to start with the next task or procedure. The loop is exited when the *flight bot* has accomplished all goals or when a fatal event occurs after which the bot is unable to go on.

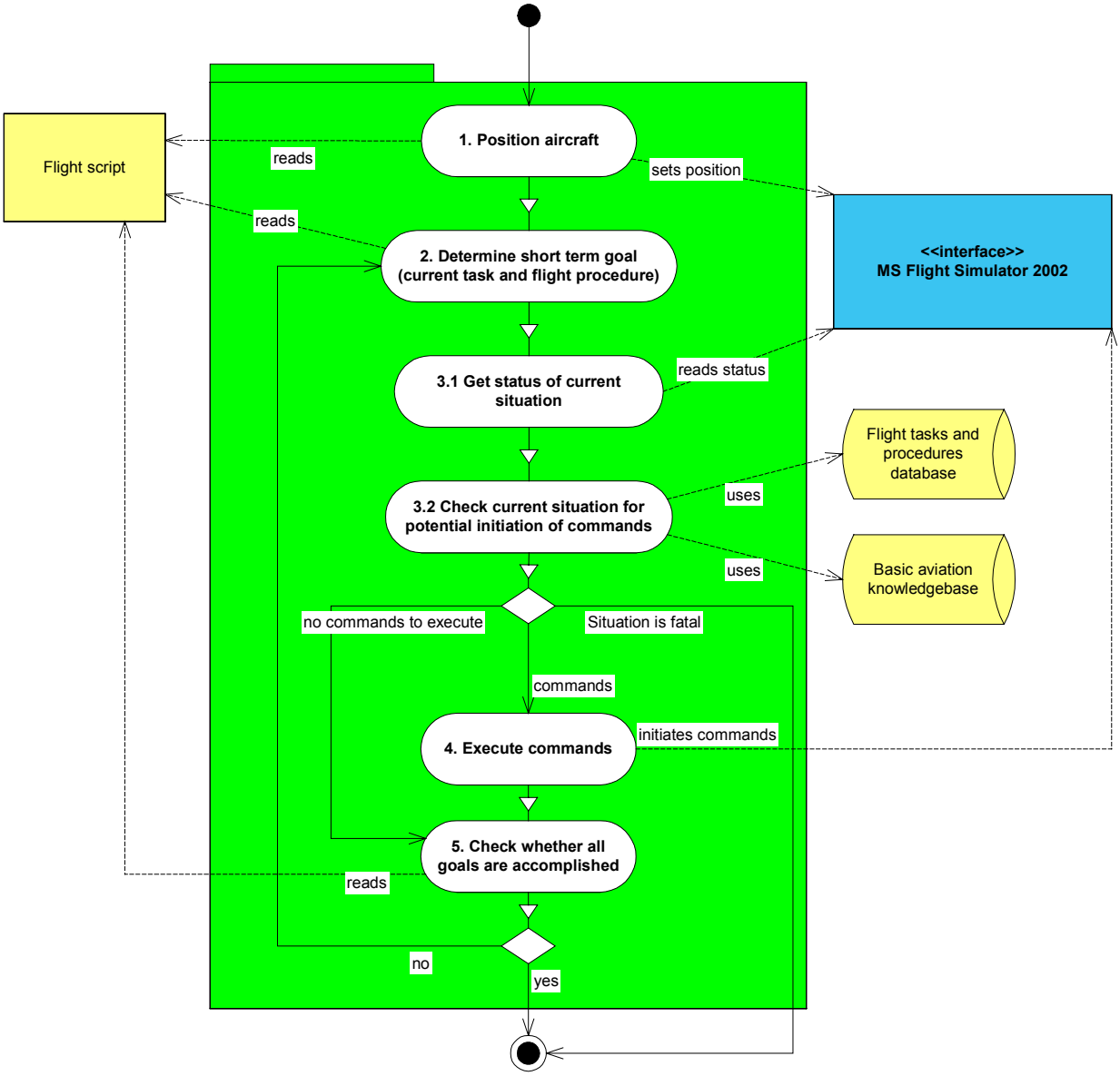

Figure 24 Activity diagram of the flight bot

In a concurrent thread, the *co pilot* is running to determine whether the flight script needs adjustments to accomplish the tasks appointed by the supervisor in the flight plan. The *co pilot*, in contrast to the *flight bot*, works with long-term goals. These goals are determined by analysing the task at hand.

The *co pilot* must assess the current situation to determine if the goal is still feasible or whether a fatal occurrence of events may happen. When a situation occurs where the goal cannot be accomplished or a critical situation is predicted, the *co pilot* will try to create a number of solutions (a solution is defined as a set of flight procedures) for this situation, which enables the *flight bot* to reach the appointed long-term goal and avoid any (predicted) hazardous situations. The number of solutions generated must be limited to the available attention resources that are not in use for other means. This should be implemented to mimic the human behaviour and limitations as is described in section 0.

With this set of solutions available, the *co pilot* must pick the best solution from the set and adjust the Flight Script accordingly. A check is then done to verify if the thread of the *flight bot* is still running. When the *flight bot* has finished, stopped or crashed, there is also no need for the *co pilot* to resume its duties.

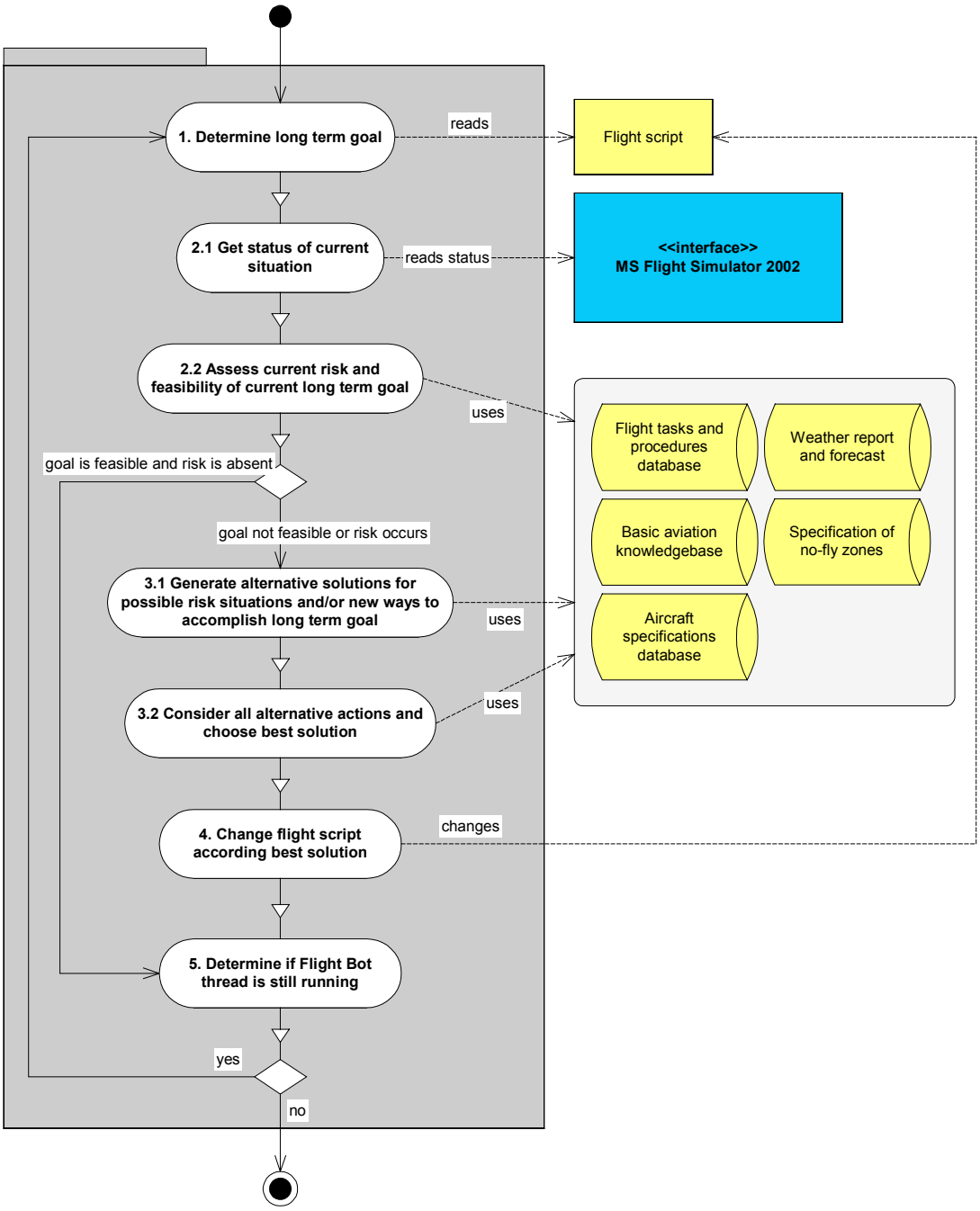

Figure 25 Activity diagram of co pilot

Chapter 6 Data storage and exchange

When there is an interactive application, there is also the need for data exchange and storage. In this project there are multiple instances where there is the need to design a data structure to define the way in which the information is stored. This chapter will elaborate on the choices that have been made regarding the method and manner in which the data is passed on.

6.1 Extensible Markup Language

In the past five years, a markup language has been developed by the World Wide Web Consortium [W3C03] to describe data. The Extensible Markup Language, abbreviated as XML, has originally been developed to exchange information on the Internet. Currently, because of its success and (relative) simplicity, it is widely used to represent data on various platforms running different kinds of applications.

The concept is an extension of the much older Standard Generalized Markup Language (SGML). The main advantages of XML are its extensibility and portability. The language is based on self-describing tags, which must be defined by the user. There are multiple ways to define the tags. Two of them are commonly used, namely by Document Type Definitions (DTDs) and by XML Schemas. This opens the possibility to exchange many kinds of information in an unambiguous and elegant manner.

6.1.1 Basics of storing data in XML

In XML, using so-called tags formats the data. These tags resemble the ones used in SGML and it's successor: HTML (HyperText Markup Language). All data is stored in the tags as basic strings of characters. This ensures the portability of the XML data over the various available platforms.

To give an example of the workings of XML, the following code is designed to catalogue books.

```
<?xml version="1.0" encoding="UTF-8"?>

<book isbn="0586217835">
 <title>
    Magician
 </title>
 <author>Raymond E. Feist</author>
 <character>
    <name>Pug</name>
    <friend-of>Tomas</friend-of>
    <birth-town>unknown</birth-town>
    <trade>Magician</trade>
 </character>
 <character>
    <name>Tomas</name>
    <birth-town>Crydee</birth-town>
    <trade>Swordfighter</trade>
 </character>
</book>
```

When examining this code closely, a few points can be noted concerning the nature of XML. Firstly, it stores the data in a hierarchical fashion. At the root of the data resides the book-tag. A title, an author and an enumeration of characters describe the book. In addition, the book has an attribute "isbn" inside the book-tag. All tags must have an equivalent closing-tag, except if the tag only holds attributes and

no data. The definition of an element is everything between a tag and it's closing tag, also known as the main building blocks of XML. The attributes give extra information about an element in particular.

Another question may rise when inspecting the previous XML code. Why is data at one instance entered as an attribute and elsewhere it is put in a child element? For example, the name of a character can also be added as an attribute to the element "character". As a matter of fact, both options are valid. However, there are a number of reasons to prefer child elements above attributes [W3S03]:

- Attributes cannot contain multiple values (child elements can)
- Attributes are not easily expandable (for future changes)
- Attributes cannot describe structures (child elements can)
- Attributes are more difficult to manipulate by program code

Also, validating attribute values against a Document Type Definition (DTD) is not easily done.

6.1.2 Document Type Definitions and XML Schemas

At the beginning of the XML era, the structure of the XML data was defined by a Document Type Definition (DTD). The DTD is a fairly simple and straightforward way to define the data structure of the XML data. The main disadvantage of a DTD with regard to XML Schemas is the lack of flexibility and functionality. With the development of the XML Schemas a new approach has been taken and many shortcomings in the design of the DTD have been removed. The support for data types is the primary strength of the XML Schema. It has also been made easier to define the format and restrictions on data, in addition to the possibility to validate the correctness of the data. Despite of the readability of a DTD, which is better than for XML Schemas, the choice has been made for this project to use XML Schemas to define the structure of the data.

The best way to show the capabilities of the method of XML Schemas is to give a real example. Therefore, the following schema has been made to describe the XML code about the classification of books from the previous section:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

 <!-- definition of simple type elements -->
 <xs:element name="title" type="xs:string"/>
 <xs:element name="author" type="xs:string"/>
 <xs:element name="name" type="xs:string"/>
 <xs:element name="friend-of" type="xs:string"/>
 <xs:element name="birth-town" type="xs:string"/>
 <xs:element name="trade" type="xs:string"/>

  <xs:simpleType name="isbnType">
    <xs:restriction base="xs:string">
      <xs:pattern value="[0-9]{10}"/>
    </xs:restriction>
  </xs:simpleType>

 <!-- definition of complex type elements -->
 <xs:element name="character">
    <xs:complexType>
       <xs:sequence>
          <xs:element ref="name"/>
          <xs:element ref="friend-of" minOccurs="0" maxOccurs="unbounded"/>
          <xs:element ref="birth-town"/>
          <xs:element ref="trade"/>
       </xs:sequence>
    </xs:complexType>
 </xs:element>
```

```
 <xs:element name="book">
    <xs:complexType>
       <xs:sequence>
          <xs:element ref="title"/>
          <xs:element ref="author"/>
          <xs:element ref="character" minOccurs="0" maxOccurs="unbounded"/>
       </xs:sequence>
       <xs:attribute name="isbn" type="isbnType" use="required"/>
    </xs:complexType>
 </xs:element>
</xs:schema>
```

A few things can be noted when sifting through this code. The whole structure is build up from elements called “simple type elements”. These elements are data types that hold only (character string) values and no other element or attribute children. The more complex elements (of type “complexType”) may have attributes and non-text children (e.g. other complex elements or simple elements). It is also allowed to reference to earlier declared elements inside another element.

In addition, the values of some elements are constrained by certain attributes. An example of this is the simpleType element “isbnType”. In this character string, only the characters ‘0’ through ‘9’ are permitted and it is required to enter ten characters. This way, data can always be tested for validity and it is possible to verify the stored data structure complies with the standard data structure the user has designed.

6.2 Use of XML in the application

Now that the concept of XML is explained, the question arises where this knowledge can be used in the *flight bot*. There are several occasions where data is gathered and exchanged. The data structures used in those situations are worked out in the following sections.

6.2.1 Exchanging the flight plan

The data structure of a flight plan has already been defined in section 3.1.1. The next step is to translate the definition to a correct XML schema. There are many ways to do this. The most common approach is to define the data structure as a hierarchical tree and create an element for every tree node, starting at the leaves of the tree. The main reason for starting at the leaves of the tree is that these elements are so-called simple type elements. By using references to the child elements, the relationship between the elements is set. The following schema uses this method to create the data structure.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" ↵
                 xmlns:fp="urn:flightplan">

 <!-- definition of the flight plan (root of the document) -->
 <xs:element name="flightplan">
    <xs:complexType>
        <xs:sequence>
          <xs:element name="steerpoint" type="fp:steerpoint" minOccurs="2" ↵
                 maxOccurs="unbounded"/>
          </xs:sequence>
    </xs:complexType>
 </xs:element>

 <!-- definition of a steer point -->
 <xs:complexType name="steerpoint">
    <xs:sequence>
       <xs:element name="GPSCoordinate" type="fp:GPSCoord"/>
```

```
      <xs:element name="timeOverSteerPoint" type="xs:string"/>
      <xs:element name="task" type="fp:task" maxOccurs="unbounded"/>
      <xs:choice id="steerpointType">
         <xs:element name="airport" type="fp:airportType"/>
         <xs:element name="GPS" type="fp:GPSType"/>
      </xs:choice>
   </xs:sequence>
   <xs:attribute name="phase" type="fp:flightPhaseList" use="required"/>
</xs:complexType>

<!-- define the various types of steer points. Don't forget to mention ↵
                them in the choice section above! -->
<xs:complexType name="airportType">
   <xs:sequence>
      <xs:element name="runway" type="xs:string"/>
      <xs:element name="runwayMagnHeading" type="fp:heading"/>
      <xs:element name="runwayMagnVar" type="xs:float"/>
      <xs:element name="runwayElevation" type="xs:float"/>
      <xs:element name="GPSRunway" type="fp:GPSCoord"/>
   </xs:sequence>
   <xs:attribute name="shortname" type="xs:string" use="required"/>
   <xs:attribute name="name" type="xs:string" use="required"/>
</xs:complexType>
<xs:complexType name="GPSType">
   <xs:sequence>
      <xs:element name="airspeed" type="xs:positiveInteger"/>
      <xs:element name="heading" type="fp:heading"/>
      <xs:element name="altitude" type="xs:float"/>
   </xs:sequence>
</xs:complexType>

<!-- definition of the phase list for each type of steerpoint -->
<xs:simpleType name="flightPhaseList">
   <xs:restriction base="xs:string">
      <xs:enumeration value="takeoff"/>
      <xs:enumeration value="cruise"/>
      <xs:enumeration value="land"/>
   </xs:restriction>
</xs:simpleType>

<!-- define restrictions of task regarding phase and steer point type in ↵
                fsdb.xml -->
<xs:simpleType name="task">
   <xs:restriction base="xs:string">
      <xs:enumeration value="takeoff"/>
      <xs:enumeration value="cruise"/>
      <xs:enumeration value="approach"/>
      <xs:enumeration value="land"/>
   </xs:restriction>
</xs:simpleType>

<!-- define a GPS coordinate -->
<xs:complexType name="GPSCoord">
   <xs:sequence>
      <xs:element name="longitude" type="fp:longitude"/>
         <xs:element name="latitude" type="fp:latitude"/>
   </xs:sequence>
   </xs:complexType>

<!-- define other simple type elements -->
<xs:simpleType name="longitude">
   <xs:restriction base="xs:float">
      <xs:minInclusive value="-180.0"/>
      <xs:maxInclusive value="180.0"/>
   </xs:restriction>
</xs:simpleType>
<xs:simpleType name="latitude">
   <xs:restriction base="xs:float">
```

```
      <xs:minInclusive value="-90.0"/>
      <xs:maxInclusive value="90.0"/>
   </xs:restriction>
 </xs:simpleType>
 <xs:simpleType name="heading">
   <xs:restriction base="xs:float">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="360"/>
   </xs:restriction>
 </xs:simpleType>
</xs:schema>
```

6.2.2 Composing a flight script

The data structure of the flight script is somewhat harder to define because of the complexity of the flight procedures and the associated parameters. The common manner in which a flight script is constructed is explained in section 3.3. In addition, the next section defines in meta-language the structure of a general flight script to make clear from what pieces it is composed.

```
<?xml version="1.0" encoding="UTF-8"?>
<flightscript>
 <phase name="takeoff|cruise|land|...">
   <task name="takeoff|cruise|..." steerpoint="n">
      <procedure name="takeoff|..." sequence="t">
         <parameter a/>
         <parameter b/>
         <GPSRunway>
            <longitude>52.1271667</longitude>
            <latitude>5.2911667</latitude>
         </GPSRunway>
         <runwayHeading>274</runwayHeading>
         <runwayElevation>21.7</runwayElevation>
      </procedure>
   </task>
   <task name="takeoff|cruise|..." steerpoint="n+1">
      <procedure name="approach|..." sequence="t">
         <parameter x/>
      </procedure>
      <procedure name="land|..." sequence="t+1">
         <parameter y/>
      </procedure>
      ...
   </task>
   ...
 </phase>
 ...
</flightscript>
```

The following figure is extracted from the example above. It depicts the relations between the various objects of which the flight script consists. A flight script contains (flight) phases. These phases contain tasks to be executed in a chronological sequence. The tasks are built up from (flight) procedures, which contain certain parameters to specify the way a procedure should be executed.

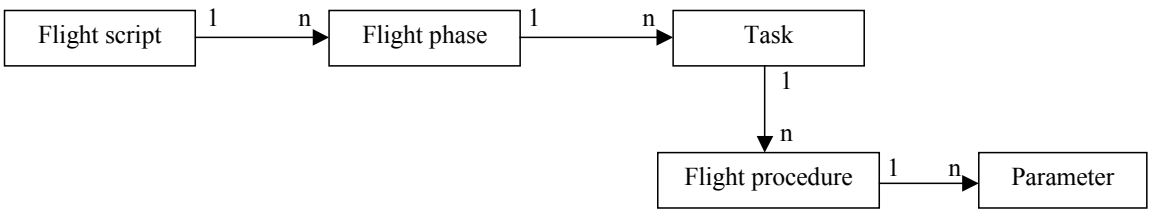

Figure 26 Relations inside the flight script

Chapter 7 Project results

During the period set for this project, much has happened and much has been developed. This chapter discusses the final application and other project results and how they correspond to the goals set for the project.

7.1 The cognitive model and the system for decision making

The main project goal was the design of "a cognitive model of a pilot, which can be used to analyse the flight plan and generate a basic flight script". This model has been discussed extensively in Chapter 2 and Chapter 4 of this report. In addition, it was required to design "a system that can execute the flight script and perceives and evaluates certain situations during flight and reasons which actions should be executed from a generated list of possible solutions for a specific situation". Chapter 3 and Chapter 4 discuss the system designed for this purpose. In Chapter 5, the design is made more practical by presenting the application design. This way the models and methods become more applicable to the programmer.

7.2 The application

The project has a third goal set to implement "these models in an application, which can acquire, analyse and execute a flight plan". The part of the application, which can acquire and analyse the flight plan, is called the *Flight Planner* application, while the part that executes the flight plan has been named *flight bot*. During the implementation of the application, many ideas came up and where worked out alongside the main goal.

7.2.1 Acquiring the flight plan

First of all, a way had to be devised to acquire the flight plan from the supervisor. After doing a brainstorm with the persons related to the project, it became apparent the flight plan had to be entered visually, i.e. by a point-and-click method on a displayed map. As the screenshot in Figure 27 shows, this has been implemented using maps from AeroPlanner [AER03]. It is possible to open multiple maps at once, while the correct flight path is drawn on every map.

When the supervisor wants to add a steer point to the flight plan, he or she double-clicks on the map at the exact place the steer point is located. After that, a dialog box pops up asking the supervisor for more details about the steer point. The first dialog box determines the general parameters for the steer point, as is depicted in Figure 28. The longitude and latitude coordinate of the steer point is determined by the point the user has clicked on the map.

The dialog box is used to enter information about the GPS coordinate and the type (i.e. airport or GPS coordinate) of the current steer point. Additionally the Time Over Steer point (TOS) is entered. It is not imperative to enter a TOS at this time. If it is not entered, the *flight bot* will ignore the TOS during flight. Finally, the dialog box asks to enter the phase of the flight at the current steer point.

Two types of steer points are implemented. When the steer point is meant as a point from which the *flight bot* should take off or land, the type should be "airport". All other steer points are of type "GPS coordinate". This type has a longitude and latitude coordinate at which point a task should be executed.

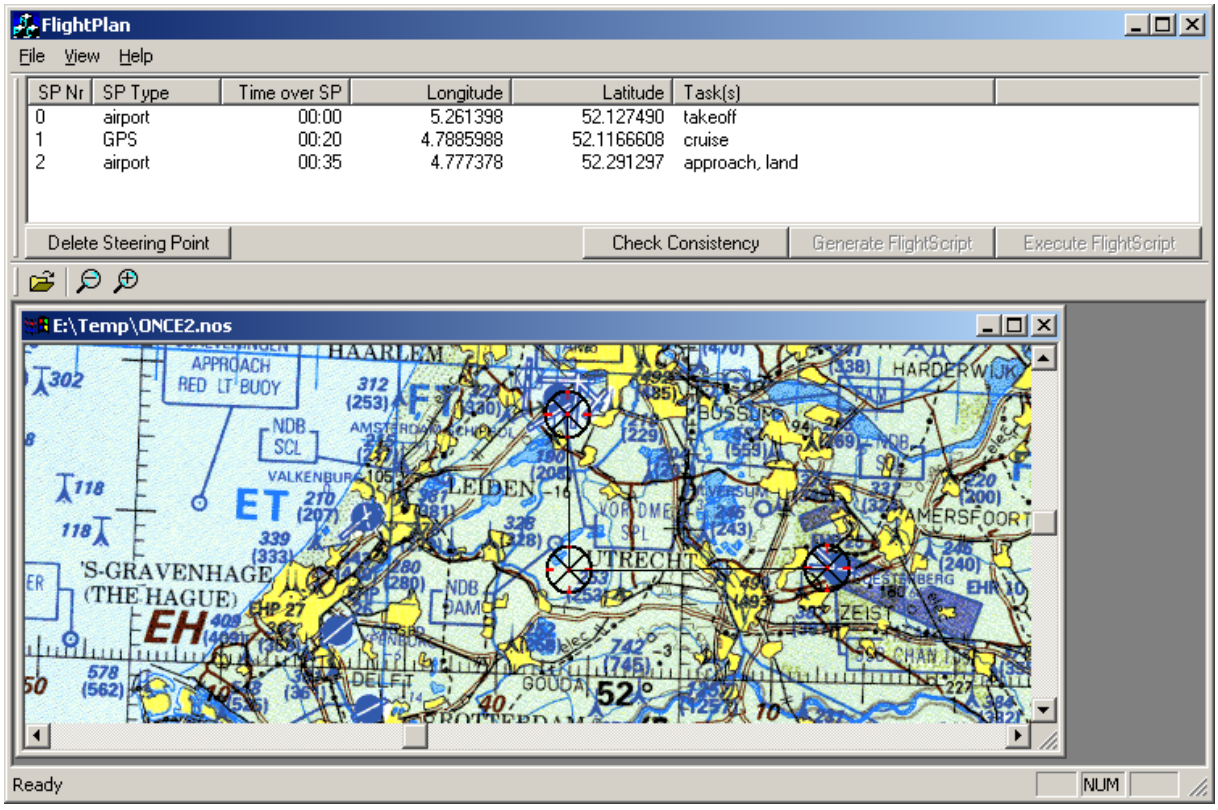

Figure 27 FlightPlanner uses maps to acquire the flight plan from supervisor

It is also important to select the correct flight phase from the first dialog box, because the task that should be executed at the current steer point depends on the flight phase. For example, the *flight bot* can only execute the task "land" during the "land" phase of the flight. If the "take off" phase is selected, the "land" task is not available for the supervisor to select.

After clicking on the "Next" button, the supervisor is asked to enter additional information about the steer point. The layout of the second dialog box depends on the type of the steer point and the flight phase, which the supervisor has entered in the first dialog box.

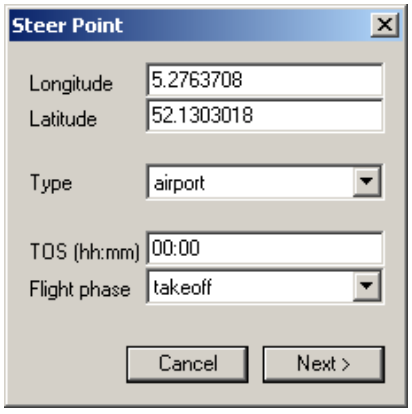

Figure 28 The dialog box for entering a steer point

When the type is "airport", the second dialog resembles Figure 29. Here, the application looks up the nearest airport in an airport database and populates the dialog box with the information about the airport (runways and ICAO name). Additionally, all airports within 10 degrees longitude and latitude are added to the airport combo box. Now, the supervisor can select quickly the intended airport and runway.

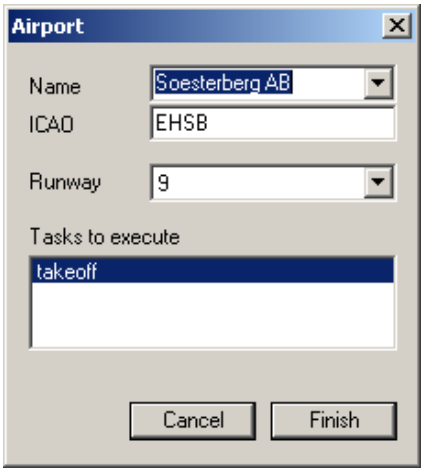

Figure 29 Airport specific dialog box

The other type of steer point that is implemented is the "GPS coordinate". Usually, a flight plan consists of more than only a point from which to take off and an airport to land at. The supervisor can specify the route to fly with GPS coordinates. The way to do this is showed in Figure 30.

Figure 30 GPS coordinate specific dialog box

With this point-and-click method it is easy for the supervisor to enter a flight plan. An example of an entered flight plan is displayed in Figure 27. The *flight bot* takes off from airport Soesterberg. It will gain altitude and begin cruising to the GPS coordinate defined in the second steer point. When the second steer point is reached, the bot turns about 90 degrees to the right to get on course to Schiphol. Finally, the bot will align with the designated runway and land at Schiphol airport.

7.2.2 Analysing the flight plan

With the flight plan entered, the supervisor must first initiate the consistency check by clicking the "Check Consistency" button. The other buttons ("Generate Flight Script" and "Execute Flight Script") are still disabled while the flight plan has not been verified. After the verification has proven the flight plan to be correct, the button "Generate Flight Script" will become enabled.

Clicking on the "Check Consistency" button will pop up a new dialog box with messages about the verification process. Here, the supervisor can see what is wrong with the flight plan if it is not approved by the application. If the flight plan is approved, the supervisor will be notified and the dialog box can be closed.

At this time, the consistency check module has not yet been implemented, as is stated in section 5.2.2. The current implementation will always assume the flight plan is correct and hence will always approve the flight plan. This module is left over for future projects to implement.

7.2.3 Execute a flight plan

With the flight plan approved, the way is paved for the *flight bot* to begin the journey through the skies. Only one thing must be done before flight execution. Although the *flight bot* can be build in a way it understands the flight plan as it resides in memory, it is easier for the bot to read a list of tasks, which is chronologically ordered. Therefore a flight script must be generated. The supervisor can do this by clicking the "Generate Flight Script" button.

The dialog box used by the Flight Script Generator has the same appearance as the one used by the Consistency Check module. Figure 31 gives an example of this dialog box.

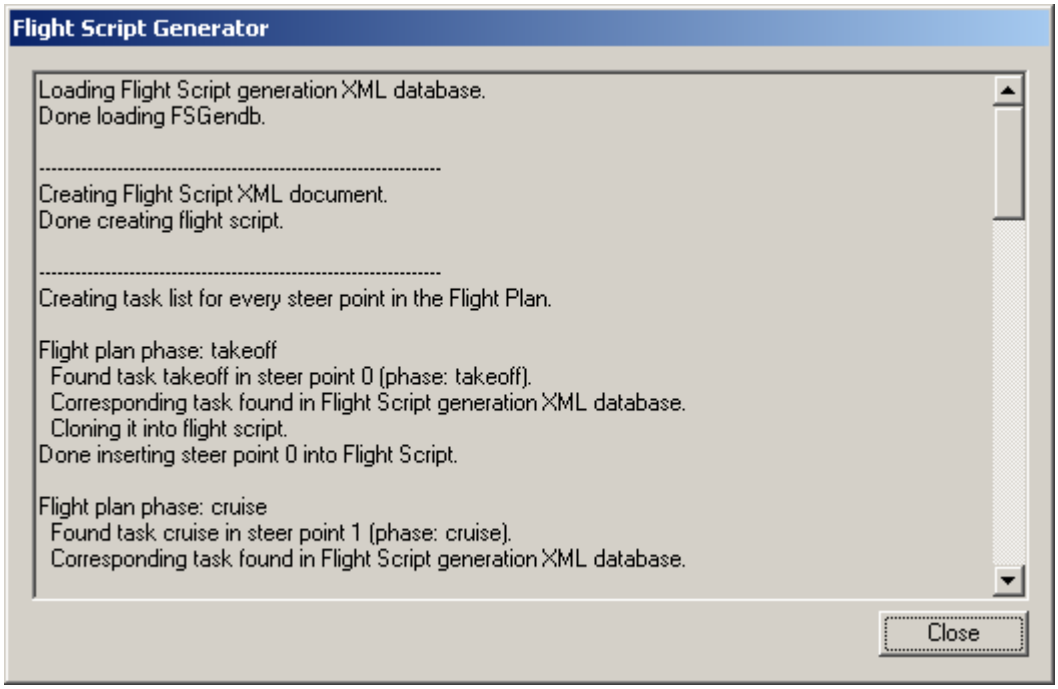

Figure 31 Dialog box of the Flight Script Generator

After the flight script has been constructed, the *flight bot* has to be initiated to execute the tasks. The supervisor does this by clicking the "Execute Flight Script" button. At this point, the application assumes Microsoft Flight Simulator 2002 is already running and the airplane is positioned at the location mentioned in the first steer point.

At the beginning, the *flight bot* will try to make a connection to Flight Simulator by using the FSUIPC module created by Peter Dowson [Dow03]. This module enables external applications to exchange data with Microsoft Flight Simulator by means of a method called Inter-Process Communication. This enables processes (or applications) that reside concurrently in memory to communicate. It is required that the FSUIPC module is installed in advance. The *flight bot* tries to connect to the FSUIPC module. If this fails, a warning is displayed resembling Figure 32. After closing this message box, the application returns to the main window, which gives the supervisor time to start Flight Simulator 2002 and retry the initiation of the *flight bot*. When the connection is successfully opened, the *flight bot* will begin executing the flight script task by task.

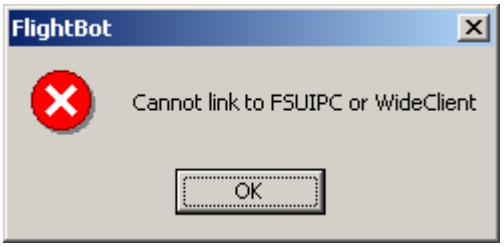

Figure 32 Warning generated when FSUIPC is not installed or FS2002 is not running

7.2.4 Extensibility of the application architecture

Another condition of this project is to construct an application architecture, which supports future extensions to the project. To accomplish this goal, the application has been set up in multiple modules. Every module that is compiled generates a separate piece of the application. These pieces are called dynamic link libraries, or simply DLLs. The main application (the FlightPlanner) calls the various modules when the supervisor initiates those commands by clicking on a button. One advantage of this approach is the fact that when one module is updated, it is not necessary to recompile the complete application. Only compiling the module suffices. In addition, it gives clarity about the architecture of the total application.

Chapter 8 Conclusions and recommendations

Now that first prototype of the *flight bot* has been implemented, a number of conclusions can be drawn from the course of the project and the results. This will be done in the first section of this chapter. Additionally, in the next section recommendations are given about possible extensions to be implemented in future projects.

8.1 What has been accomplished

Coming at the end of the project, it is a good time to look back and see what has been finished of the goals and sub-goals set for this graduation project.

8.1.1 Research and design

In the course of this project we have done research to model a flight bot that resembles a human pilot in behaviour. We have set up a number of rules and limitations the *flight bot* should keep while executing a flight plan to mimic the limitations human pilots have while flying. In addition, the flight procedures and tasks for the *flight bot* to fly an airplane have been described in an XML knowledge base.

8.1.2 The first prototype

Furthermore, we have implemented a first prototype of the *flight bot*. This application currently has three tiers. The *Flight Planner* acquires the flight plan from the supervisor by giving the supervisor the ability to visually enter the flight plan. The second tier analyses the flight plan and converts it to a list of tasks and flight procedures in a language the *flight bot* can easily understand. The third part of the application executes the flight script in Microsoft Flight Simulator 2002.

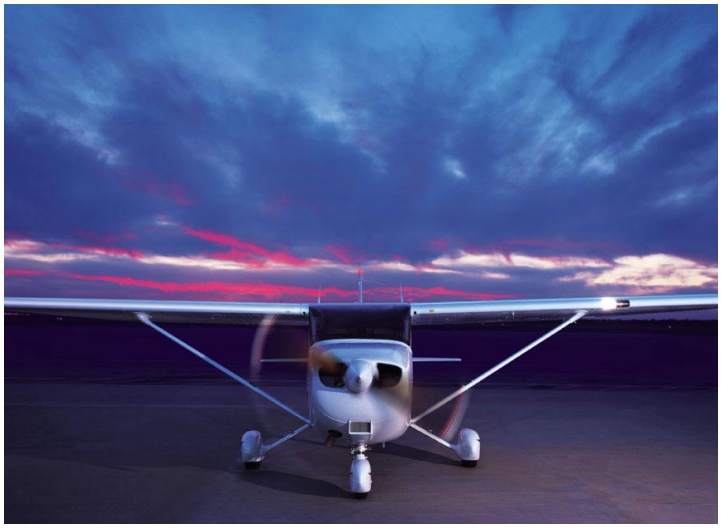

Figure 33 A Skyhawk waiting for take off at an airport

8.1.3 Please fasten your seat belts: we are taking off!

The basic aviation procedures and tasks are implemented in the flight bot. Currently the implementation of the *flight bot* does not incorporate any artificial intelligence yet. It is still based on procedures and (constant) parameters. This means it cannot react to the surroundings yet. Any unexpected and possibly harmful event (i.e. a collision course with another aircraft, a high cliff or just very bad weather) will result in a fatal crash of the airplane. It is for future projects to implement the avoidance or handling of such situations.

8.1.4 Further remarks

When looking back on the project, it appears this project has been mainly focussed on the preparations for the implementation of an intelligent flight bot. With the first prototype all requirements are met to start flying for the *flight bot*. At the start of this project it was not yet known how time consuming the implementation of an intelligent bot is. This project is estimated to last a few more years, before a really intelligent bot is build. However, it was a satisfying experience and much fun to be working on this project. We want to encourage any person who wants to spend time on this project to extend the application and enhance the intelligence of the *flight bot*.

8.2 Recommendations for future projects

Because the application is just a first prototype, there is much to improve yet to extend the *flight bot* application. From the programmer's point of view a number of items are still not yet implemented. This refers to the *flight plan consistency checker* and the *co pilot* module. The final tier of the application, the part in which the supervisor can compare the entered flight plan and the actual flight, is also not yet implemented. Another module should be built to log the current flight. It is advisable to use the Flight Recorder inside Microsoft Flight Simulator 2002 to do this to reduce the time spent on such a module.

The *flight planner* can also be extended on a number of points. For example, the flight path is now drawn directly from one steer point to the next. It would be interesting and useful to implement a module that calculates the flight path and avoids certain zones in which flying is prohibited. In addition, this new module could take the weather forecast into account while planning the flight.

Furthermore, the *flight script generator* is not yet as flexible as it is supposed to be. It is imaginable to add artificial intelligence also to this module to make presumptions about certain parameters that are not clearly defined in the flight plan.

Much work has to be done in the *flight bot* itself. Currently the *flight bot* is only following the flight script without using any artificial intelligence. To mimic human behaviour, a certain level of intelligence has to be built into the *flight bot* and the c*o pilot* module. Another problem arises in mountainous terrain. When a cliff looms up directly in front of the aircraft, it will crash into the mountain wall because the *flight bot* does not yet look out the window of the cockpit. In addition, many more tasks and flight procedures could be added to the knowledge base to enhance the flying capabilities of the *flight bot*.

Finally, it would be interesting to combine this project with other subprojects of the ICE environment to see the outcome. For instance, the *situation awareness project* can be started alongside the *flight bot* to compare whether the results of both projects are consistent and similar. Another fascinating situation would be the combination of multiple flight bots flying in one environment. Microsoft Flight Simulator 2002 supports multiple users to log on to one instance of Flight Simulator. This way, for instance, the training of a human pilot becomes more realistic to the pilot.

Appendix A Preliminary research

A study was done in the interest of the project to determine which software could be used best to accomplish the goal of building an autonomous flight bot. The primary source for the comparison of the available software was the documentation and resources for developers who want to add new features to the various available simulators. Most of the documentation was found on the Internet.

A.1 Available simulators

For the ICE project there are currently a number of simulators in use for research and development purposes. These simulators come in three flavours:

- Microsoft Flight Simulator 2002 Professional Edition
- Flightgear 0.9.1
- Microprose Falcon 4.0

The last one, however, is not an option for this particular project because of the lack of extensibility. Falcon 4.0 is an extremely realistic F-16 combat simulator, but it does not have the possibility to add or customize features to the simulator in any way. Therefore only two options are still available for use in this project.

A.2 Comparison

The most important advantages and disadvantages of the two main candidates for use are shown in the table below.

	MS Flight Simulator 2002 Pro	**Flight Gear 0.9.1**
Advantages:	• Simulator is relatively stable and already has been through several extensive implementation cycles • Well-documented Software Development Kit (SDK) • Flight Instructor mode for easy in-flight adjustments • Many plugins, extra aircrafts, etc. available for download on the internet	• Freely available and customisable source code (open source) • All implemented models (environment, aircraft, etc.) can be checked and customized for realism • Available for multiple platforms (Linux, Windows) • Developers are supported by mailing lists, forums, etc.
Disadvantages:	• Less ‘enthusiastic’ support. (Microsoft does not officially support the SDK, although support is always available in the community of the developers.) • Not available as open source software • Only supported by MS Windows 2000, XP or 98. External plugins can always be implemented platform independently, presuming the platform supports sockets.	• Not as stable on all platforms • Still in the beginning of the implementation cycle, thus it is not as advanced in many areas, e.g. simulation of the environment, aircraft dynamics, multiplayer options, etc.

A.3 Recommendation

After considering the arguments mentioned in the comparison of the two simulators, the best choice is to use Microsoft Flight Simulator 2002 Professional Edition. The main reasons for this decision are listed below.

- Quality considerations: the main advantage of Flightgear is the availability by open source standards. Every nook and cranny in the simulator can be checked for realism and can in case of an unrealistic situation, be adjusted to make the simulator even better. This cannot be done in Flight Simulator 2002, because the source is not freely available. However, because of the age of Flight Simulator, compared to Flightgear, the developers at Microsoft have had more time (and resources) to develop and improve Flight Simulator 2002.
- Developer support considerations: the main disadvantage of Microsoft Flight Simulator 2002 is the lack of support for developers who want to implement new features in Flight Simulator. However, this is compensated by the community of developers who have experience with implementing new plugins and extending current functionality in Flight Simulator.

Appendix B Aircraft specifications for the Cessna 172SP Skyhawk

All estimated performance data are based on airplane weights at 2,550 pounds; standard atmospheric conditions; level, hard-surface, dry runways; and no wind. They are calculated values derived from flight tests conducted by The Cessna Aircraft Company under carefully documented conditions and will vary with individual airplanes and numerous factors affecting flight performance [CES03].

Performance data	English	Metric
Cruise * 45% power at 10,000 ft		
range	638nm	(1,182 km)
time	6.72 hr	
Cruise * 75% power at 8,500 ft		
range	518nm	(959 km)
time	4.26 hr	
Engine: Textron Lycoming IO-360-L2A		
180 BHP at 2,700 RPM		
Landing Performance		
Ground roll	575 ft	(175 m)
Total distance over 50 ft obstacle	1,335 ft	(407 m)
Power Loading	14.2 lb/hp	(6.4 kg/hp)
Propeller: 2-Bladed Fixed Pitch, Diameter	76 in	(1.93 m)
Rate of Climb at Sea Level	730 fpm	(223 mpm)
Service Ceiling	14,000 ft	(4,267 m)
Speed		
Maximum at sea level	126 kts	(233 kph)
Cruise, 75% power at 8,500 ft	124 kts	(230 kph)
Stall Speed, CAS		
Flaps up, power off	53 knots	(98 kph)
Flaps down, power off	48 knots	(89 kph)
Takeoff Performance		
Ground roll	960 ft	(293 m)
Total distance over 50 ft obstacle	1,630 ft	(497 m)
Wing Loading	14.7 lb/sq ft	(71.8kg/sqm)

Appendix C Flight task knowledge base

This part describes the basic flight procedures and the advanced tasks used by the *flight bot* to execute a flight script in a standard Cessna 172SP Skyhawk. The advanced tasks are composed of basic flight procedures. The procedures are defined in the next section. The basis of these procedures and tasks has been taken from the ground school manual Microsoft provides with Flight Simulator 2002 [Mach02].

C.1 Basic flight procedures

When a standard margin is mentioned, a default margin should be defined for the action in question.

- **Change attitude**
  Description: Increase or decrease the bank angle of the aircraft.
  Parameters: Goal attitude angle (degrees).
  Actions:
  1. Read out current attitude and calculate difference with goal attitude angle.
  2. If difference is greater than the standard margin then smoothly increase or decrease the aileron control input value depending on whether the difference is positive or negative.
  3. Read out new attitude angle and calculate difference with goal attitude. Go to point 1 if the difference is greater than the standard margin.
  4. Smoothly set aileron control input value to zero.

- **Change pitch**
  Description: Increase or decrease the pitch angle of the aircraft.
  Parameters: Goal pitch angle (degrees).
  Actions:
  1. Read out current pitch and calculate difference with goal pitch angle.
  2. If difference is greater than the standard margin then smoothly increase or decrease the elevator control input depending on whether the difference is positive or negative.
  3. Read out new pitch angle and calculate difference with goal pitch angle. Go to point 1 if the difference is greater than the standard margin.

- **Change power settings**
  Description: Increase or decrease the throttle of the aircraft.
  Parameters: Goal throttle setting (percentage||speed in knots).
  Actions:
  1. Read out current throttle setting and calculate difference with goal setting.
  2. Gently increase or decrease throttle to goal setting.

- **Change mixture settings**
  Description: Set fuel mixture level of the engine.
  Parameters: Goal mixture setting (percentage).
  Actions:
  1. Read out current mixture setting and calculate difference with goal setting.
  2. Gently increase or decrease mixture to goal setting.

- **Trim**
  Description: Set trim button to adjust elevator trim tab.
  Parameters: Up trim or down trim (up||down).
  Actions:
    1. Increase trim level by one unit if aircraft needs a down trim, decrease trim level by one unit if aircraft needs an up trim.

- **Change flaps**
  Description: Set flaps indicator to adjust flaps position.
  Parameters: Extend or retract flaps (up||down).
  Actions:
    1. Increase flaps postition by one unit if flaps need to be lowered (extended), decrease flaps level by one unit if flaps need to be raised (retracted).

- **Change rudder to keep inclinometer in the central position**
  Description: During a turn, the rudder is used to avoid slipping.
  Parameters: None.
  Actions:
    1. Read out inclinometer and calculate the difference to the normal position.
    2. If difference is greater than the standard margin, smoothly increase the pressure on the left rudder pedal if the plane makes a left turn; smoothly increase the pressure on the right rudder pedal if the plane makes a right turn.
    3. Read out new inclinometer value and calculate difference with the normal position. Go to point 1 if the difference is greater than the standard margin.

- **Change rudder during take-off to keep aligned with runway**
  Description: During take-off, the rudder is used to keep the aircraft on the runway using the centreline of the runway as a guide.
  Parameters: None.
  Actions:
    1. Calculate deviation from centreline of the runway.
    2. If difference is greater than the standard margin, smoothly increase the pressure on the left rudder pedal if the plane deviates to the right; smoothly increase the pressure on the right rudder pedal if the plane deviates to the left.
    3. Calculate new deviation. If difference is greater than the standard margin, go to point 1.

- **Set switch or knob of particular instruments**
  Description: This procedure sets one of the many other switches and knobs of the instrument panel that have not yet been described in other flight procedures.
  Parameters: Button or knob identification (number) and intended value (number).
  Action:
    1. Set knob or button to the intended value.

C.2 Advanced tasks

- **Take-off**
  Description: The aircraft starts at a runway of an airport. It will make speed and take off at a certain velocity. Then the plane gains altitude until cruise altitude (or a predefined altitude) has been reached.
  Preconditions:
  - The aircraft is positioned at the start of a runway.
  - All pre-take off checklists are done.
  - The throttle is set to zero and the plane is ready for take off.

  Postconditions:
  - The aircraft is airborne and has reached the correct altitude.
  - Throttle is greater than the idle state.
  - Ground speed is greater than zero.

  Actions:
  1. Release any brakes (use *Set a particular instrument's switch or knob*).
  2. *Change mixture settings* to 100%.
  3. *Change power settings* to 100%.
  4. *Change rudder during take-off to keep aligned with runway.*
  5. Read out Indicated Airspeed indicator. If IAS is smaller than 55 knots, go to point 4.
  6. *Change rudder to keep inclinometer in the central position.*
  7. *Change pitch* to +11 degrees.
  8. Read out altimeter. If altitude is smaller than 500 feet, go to point 8.
  9. *Change power settings* to 85 knots.
  10. Read out altimeter. If altitude is smaller than defined altitude, go to point 10.
  11. *Change pitch* to 0 degrees.

- **Fly level**
  Description: Gets the plane flying without gaining or losing altitude.
  Preconditions:
  - The aircraft is airborne.

  Postconditions:
  - The aircraft flies level.

  Parameters:
  - None.

  Actions:
  1. Read out altimeter, Vertical Speed Indicator and attitude indicator.
  2. If VSI is greater than zero, *change pitch* to –3 degrees.
     If VSI is smaller than zero, *change pitch* to +3 degrees.
  3. Read out Vertical Speed Indicator. If VSI is still diverging from zero, *increase pitch* with another (+/-)3 degrees. Go to step 3.
  4. Read out value of elevator control unit. If value is greater or smaller than zero, decrease value one unit towards zero.
  5. Read out Vertical Speed Indicator. If VSI is still converging to zero go to point 5.
  6. Read out value of elevator control unit. If value is greater than zero, *trim down*. If value is smaller than zero, *trim up*. If value is not zero, go to point 5.

- **Fly straight**
  Description: Gets the plane flying without turning left or right.
  Preconditions:
  - The aircraft is airborne.

  Postconditions:
  - The aircraft flies in a straight line.

  Parameters:
  - None.

  Actions:
  1. *Change attitude* to zero.

- **Turn**
  Description: Makes a turn to a specific heading.
  Preconditions:
  - The aircraft is airborne.

  Postconditions:
  - The aircraft flies in a straight line at the specified heading.

  Parameters:
  - Target heading (degrees)
  - Steep turn (true || false)

  Actions:
  1. Read out attitude indicator and heading indicator.
  2. Calculate difference to the specified target heading. If difference is positive, increase aileron control input value smoothly. Decrease aileron control input value smoothly if difference is negative.
  3. *Change attitude* to +20 degrees (in case of steep turn: +50 degrees) if difference is positive. *Change attitude* to –20 degrees (in case of steep turn: -50 degrees) if difference is negative.
  4. *Change rudder to keep inclinometer in the central position.*
  5. Read out attitude indicator and heading indicator.
  6. Calculate difference to the specified target heading. If difference is greater than default margin, go to step 4.
  7. *Change attitude* to zero degrees.
  8. *Change rudder to keep inclinometer in the central position.*

- **Climb (or descent)**
  Description: Gets the plane flying at the specified altitude.
  Preconditions:
  - The aircraft is airborne.

  Postconditions:
  - The aircraft flies level at the specified altitude.

  Parameters:
  - Target altitude (meter).

  Actions:
  1. Read out altimeter, Vertical Speed Indicator and attitude indicator.
  2. *Change power settings*: increase if plane should climb, decrease if plane should descent.
  3. Read out altimeter, Vertical Speed Indicator and attitude indicator. If VSI is smaller than 75 knots during climb, go to step 2. If VSI is greater than –75 knots during descent, go to step 2.
  4. If difference between current altitude and goal altitude is greater than the standard margin, go to step 4.

5. If VSI is greater than zero, *change pitch* to –3 degrees.
   If VSI is smaller than zero, *change pitch* to +3 degrees.
6. Read out Vertical Speed Indicator. If VSI is still diverging form zero, *increase pitch* with another (+/-)3 degrees. Go to step 3.
7. Read out Vertical Speed Indicator. If VSI is greater than the standard margin, go to point 4.
8. Read out value of elevator control unit. If value is greater or smaller than zero, decrease value one unit towards zero.
9. Read out Vertical Speed Indicator. If VSI is still converging to zero go to point 5.
10. Read out value of elevator control unit. If value is greater than zero, *trim down*. If value is smaller than zero, *trim up*. If value is not zero, go to point 5.

- **Approach runway**
  Description: Gets the plane aligned with the runway.
  Preconditions:
  - The aircraft is airborne.
  - The aircraft is within range of the specified airport.

  Postconditions:
  - The aircraft flies aligned with the specified runway.

  Parameters:
  - Target airport and runway. (ICAO name of the airport, GPS coordinates of start and end of runway)

  Actions:
  1. Calculate the heading of the runway with the start and end GPS coordinates.
  2. *Turn* to the calculated runway heading.
  3. Read out the current position of the aircraft. (GPS coordinates)
  4. Calculate the offset from the airplane perpendicular to the line intersecting the start and the end of the runway.
  5. If the offset is positive, *turn* +90 degrees. If the offset is negative, *turn* –90 degrees.
  6. Calculate the offset to the line intersecting the start and the end of the runway. Until the offset is within the default margin, go to step 5.
  7. *Turn* to the calculated runway heading.

- **Land**

  Description: Gets the plane on the ground at the specified runway and stops the airplane.

  Preconditions:
  - The aircraft is airborne.
  - The aircraft is within range of the specified airport.
  - The aircraft is aligned with the specified runway.

  Postconditions:
  - The aircraft has landed on the specified runway and stands idle.

  Parameters:
  - Target airport and runway. (ICAO name of the airport, GPS coordinates of start and end of runway)

  Actions:
  1. Read out the current position of the aircraft (GPS coordinates and ground altitude).
  2. Calculate the descent rate for the aircraft at which the start GPS coordinates of the runway is reached with the current position.
  3. *Change pitch* to –14 degrees.
  4. Read out Vertical Speed Indicator. If VSI is greater than the calculated descent rate, *change power settings* to a lower value. If VSI is lower than the calculated descent rate, *change power settings* to a higher value.
  5. Read out altimeter and Indicated Airspeed. If IAS is greater than 65 knots, *change pitch* to a higher value. If IAS is lower than 65 knots, *change pitch* to a lower value.
  6. If altitude is greater than 20 feet, go to step 4.
  7. *Change pitch* to +10 degrees.
  8. Read out altimeter. If altitude is greater than zero, go to step 8.
  9. *Change pitch* to 0 degrees.
  10. Use brakes and read out Indicated Airspeed. If IAS is greater than 0, go to step 10.
  11. Set parking brakes.

Appendix D Bibliography

D.1 Literature

[Ber00] Berndt, J.S. (2000) JSBSIM: *A configurable flight dynamics model* http://jsbsim.sourceforge.net/JSBSim.PDF

[KBD97] Kallus, K.W., Barbarino, M., Damme, D. Van (1997) *Model of the Cognitive Aspects of Air Traffic Control*, EUROCONTROL, http://www.eurocontrol.int/humanfactors/docs/HF7-HUM.ET1.ST01.1000-REP-02.pdf

[Mach02] Machado, R. (2002) *Rod Machado's Ground School*, Microsoft (bundled with Microsoft Flight Simulator 2002)

[Mou03] Mouthaan, Q.M. (2003) *Flying an F16: A knowledge base describing the situations an F16 pilot may encounter*, Technical Report DKS03-03 / ICE 03, Data and Knowledge Systems group, Faculty of Information Technology and Systems, Delft University of Technology, http://www.kbs.twi.tudelft.nl/Publications/Report/2003-Mouthaan-DKS03-03.html

[PoSt99] Pooley, R.J., Stevens, P. (1999) *Using UML: software engineering with objects and components*, Addison Wesley Longman Ltd, England

[Ras86] Rasmussen, J. (1986) *Information processing and human-machine interaction: An approach to cognitive engineering*, North-Holland, New York (NY), USA

[SaAm00] Sarter, N.B., Amalberti, R. (2000) *Cognitive engineering in the aviation domain*, Lawrence Erlbaum Associates, Inc., New Jersey, USA

[SRK97] Seamster, T.L., Redding, R.E., Kaempf, G.L. (1997) *Applied cognitive task analysis in aviation*, Ashgate Publishing Ltd, England

[WiNa88] Wiener, E.L., Nagel, D.C. (1988) *Human factors in aviation*, Academic Press, San Diego, USA

D.2 Websites

[AER03] Imaps (2003) *AeroPlanner.com* http://www.aeroplanner.com

[CES03] Cessna Aircraft Company (2003) *Cessna SkyhawkSP. Specification and description* http://skyhawksp.cessna.com/spec_perf.chtml

[Dow03] Dowson, P. (2003) Creator of the FSUIPC module. http://www.schiratti.com/dowson.html

[FAA03] Federal Aviation Administration (FAA) (2003) *Flight services 7110.10P*,

http://www2.faa.gov/atpubs/FSS/INDEX.HTM

[Mack02] Mackie, R. (2002) Microsoft Flight Simulator and Microprose Falcon 4.0 resources
http://website.lineone.net/~rmackie/flightsim.htm

[MSFS03] Microsoft Flight Simulator 2002 Homepage (2003)
http://www.microsoft.com/games/fs2002/

[Mur98] Murr, D.L. (1998) *Flightgear Flight Simulator project design proposal*
http://www.flightgear.org/proposal-3.0.1

[Oli03] Olivares, T. (2003) *FlightGear: A Flight Simulator For Linux!*
http://www.simflight.com/news5/modules.php?name=News&file=article&sid=2511

[Ols02] Olsen, C.G. (2002) *Flightgear Flight Simulator Overview*
http://www.flightgear.org/Overview/

[SDD02] Selig, M.S., Deters, R., Dimock, G. (2002) *Aircraft Dynamics Models for Use with FlightGear.*
http://www.aae.uiuc.edu/m-selig/apasim/Aircraft-uiuc.html

[W3C03] World Wide Web Consortium homepage (2003)
http://www.w3.org/

[W3S03] W3Schools (2003) *W3Schools online web tutorials*
http://www.w3schools.com/

[Wil03] Williams, E. (2003) *Aviation Formulary V1.37*
http://williams.best.vwh.net/avform.htm

