TOWARDS AN INTELLIGENT COCKPIT ENVIRONMENT:

a probabilistic approach to situation recognition in an F-16

Mouthaan, Quint

Technical Report DKS-03-03 / ICE 03
Version 1.1, May, 2003

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

TU Delft

Graduation Committee:

Dr. drs. L.J.M. Rothkrantz (supervisor),
Prof. dr. H. Koppelaar (chairman),
Prof. dr. ir. E.J.H. Kerckhoffs.

Mouthaan, Quint M. (quintmm@hotmail.com)

"Towards an intelligent cockpit environment: a probabilistic approach to situation recognition in an F-16"

Technical Report DKS-03-03 / ICE 03
Version 1.1, May, 2003

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

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

Keywords: ICE project, knowledge based system, F-16, situation recognition, situation awareness, probabilistic, Bayesian, expert system.

Abstract

The ability to fly has become more and more important since the invention of the airplane. Not only the importance of commercial aviation has grown, but especially that of military aviation. There is a continuous demand for better, faster and more manoeuvrable airplanes. As a result, flying an airplane has become more and more complex and the danger of a pilot making a mistake has become bigger and bigger. To prevent a pilot from making such mistakes during a flight cockpits have been designed as pilot friendly as possible and a lot of the systems in the cockpit have been automated. Research is being done constantly to new cockpit systems. That research has focused on intelligent cockpit environments. Such environments usually they filter the information for the pilot and present him with only the most relevant information for the current situation to keep the workload of the pilot as low as possible. But they can also monitor the pilot and inform him if he makes a mistake.

This report tries to answer a question that is relevant for most of those intelligent systems: *Is it possible to design a system that recognizes, in real time, the situation a pilot is in based on the actions of the pilot, the state of the airplane and information from the environment?*

The system that is described focuses on detecting the current situation when flying an F-16, but it is designed to be able to work with other airplanes as well.

In this report three possible models for the system are described: a finite state model, a probabilistic model and a causal model. A prototype has been built and tested which is based on the probabilistic model. This model uses a knowledge base containing data that describes a number of situations that the pilot might encounter. The system will convert the knowledge in the knowledge base to a set of rules that compare the information in the knowledge base with the state of the real world at a certain moment. Based on this comparison the rules will generate probabilities that the situations given in the knowledge base are occurring. These probabilities are combined using Bayesian belief networks, which will produce for every situation the probability that it is occurring and the probability that it is not occurring. The system will then decide which of the situations is most likely to be occurring.

The prototype was tested using Microsoft Flight Simulator 2002 to simulate a flight with an F-16. The program performed very well. During the tests it became clear though that the program still made some mistakes incidentally. After analysing those mistakes the conclusion was that the main cause for these mistakes was that the probabilistic model could not account for the relationships between actions or events over a period of time.

The final conclusion is that a situation recognition system based on a probabilistic model performs very well, but that the performance of the system would be even better if it was based on a causal model.

Preface

This thesis has been written as a result of a research project that I have worked on during my graduation at the Knowledge Based Systems group of the faculty of Information Technology and Systems at the Delft University of Technology. The project was done as part of the ICE (Intelligent Cockpit Environment) project. The goal of the ICE project is to do research to new techniques and technology that can be used to assist the crew of an airplane during a flight.

Project overview

The goal of this particular project was to determine if a system could be built that could detect the current situation during a flight with an F-16. Because at the start of the project we did not have any knowledge about how to fly an F-16, a preliminary research project was done in which this knowledge was gathered. After that project had been completed and a knowledge base with knowledge about flying an F1-6 had been created, the best way to implement a situation recognizer was investigated. After a model of the recognition process had been created a prototype was implemented and tested. The testing was done with a flight simulator on a normal PC. The flight simulator that was used was the Microsoft Flight Simulator 2002. To test the situation recognizer that had been created a cockpit of an F-16 was created for the flight simulator and a number of test flights were recorded. The program was tested on these recorded flights.

Acknowledgements

During this project I was supervised by Dr. drs. Leon Rothkrantz, who helped me to put all the results of this research project in a structured way in this report. I want to thank him for all his help. I also had help from ir. Patrick Ehlert, who gave constructive criticism when it was necessary. Furthermore I would like to thank Gideon Reisel who is a former F-16 pilot and who gave us a lot of information about how to fly an F-16. I would also like to thank prof. dr. Henk Koppelaar for bringing me into contact with Gideon Reisel.

The structure of the report

The report starts with an introduction that describes the project and its goals. In chapter 2 some of the knowledge that has been gathered about the F-16 is described. Chapter 3 contains explanations of some artificial intelligence techniques that could be used for the situation recognition process. The system architecture is described in chapter 4. In this chapter we also describe three models that could be used to implement the reasoning process. Then in chapter 5 the architecture of the system and the probabilistic and the causal models are explained in more detail. We also make a choice between the possible artificial intelligence techniques and explain how the chosen techniques could be used to implement the probabilistic and causal models.

After the models have been described we will describe the design of the prototype that was implemented in chapter 6. In chapter 7 the contents of the rule base are described.

Then in chapter 8 we will describe the test scenarios that have been flown and the results of the prototype when it was run on those scenarios. After the test results have been described we will draw some conclusions about the performance of the program and given recommendations for future work in chapter 9.

The appendices of this report describe the structure of the knowledge base, the way in which the Microsoft Flight Simulator 2002 is used to test the program and what the cockpit of an F-16 looks like. We have also included a paper that is based on this report and that has been submitted to the 15th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'03) in October 2003 in Nijmegen in The Netherlands.

Quint Mouthaan,
Delft, May 2003.

Contents

List of Figures

Part I

The system design

Chapter 1

Introduction

A lot of boys dream of becoming a pilot and flying an airplane like the F-16. Some of those boys keep dreaming of this while they grow older and apply for a job as a pilot. Very few of those are selected to get the training necessary to become a pilot. One could wonder why this selection process is so hard. Two reasons can be identified for this. First of all there are only a limited number of positions available, secondly a pilot that flies in an F-16 has to be in perfect shape and must be able to function at his best even when he is under a lot of stress. There are but a few people who are capable of this to such an extent that they are capable of flying an F-16 and even they can still make mistakes. Making a mistake while flying can be deadly, especially in an F-16. To help prevent these mistakes systems have been developed to assist a pilot in performing his task. Such systems have not only been developed for the F-16, but also for pilots of other airplanes, who have to cope with a lot of the same problems. To understand what these systems do and why they do it we have to know what kind of mistakes pilots make and what the cause is for those mistakes. To be able to do this we need to know something about the cognitive process of a pilot during a flight. Therefore a pilot model of this cognitive process has been created and is described in section 1.1.

1.1 A cognitive pilot model

Flying an airplane is not an easy thing, especially when it comes to a military aircraft like an F-16. From time to time we can hear on the news that an airplane has crashed. Sometimes this is caused by a malfunction in the airplane, other times the cause of the crash is a pilot error. One of the reasons that flying is so difficult is that a pilot has to process a large amount of information in a small period of time and has to take important actions based on that information. This process is depicted in figure 1.1.

The different phases of this model are explained next. To give an idea of what can go wrong in this process some of the mistakes a pilot can make and some possible causes of those mistakes are described as well.

Information: The information is everything that is presented to the pilot in the aircraft. This is the data from the flight instruments in the cockpit and the information the pilot gets from the environment. The data the pilot gets from the flight instruments can be erroneous. This might be caused by a defect in the aircraft instruments, for example a needle that is stuck or a light that is broken.

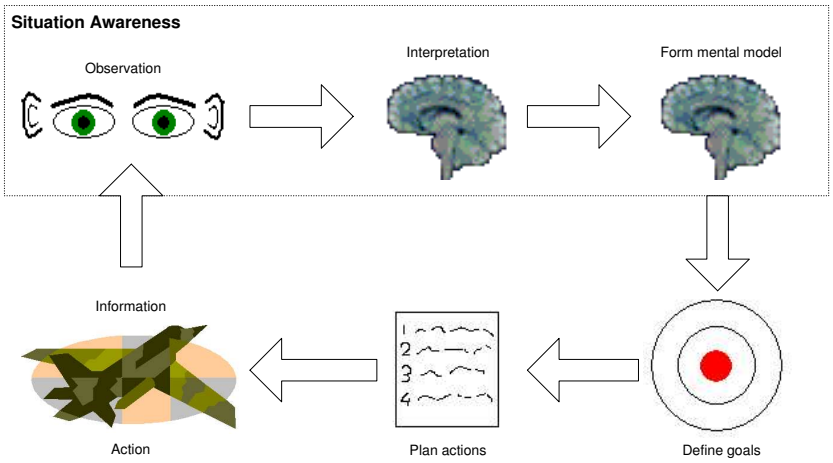

Figure 1.1: A cognitive pilot model.

Observation: The pilot observes (some of) the information he gets from the cockpit instruments and the environment. This can be visual as well as auditory information. The pilot may make an error in his observations. He might miss some of the information that is presented to him for example. This can be because the pilot is not paying enough attention or because the pilot has to absorb so much information that there is an information overload.

Interpretation: When the pilot has observed the information he will interpret it. This means he will try to determine what the information means. He will always do this in a specific context. For example in the F-16 there is a light that will be switched on during taxiing, but that will also be switched on during air refueling when the air refuel door is open. The meaning of this light can only be determined if the pilot knows which situation he is in. The pilot can make a mistake in interpreting the information he receives. This can happen when the pilot has had insufficient training or when his awareness drops, for example because he is tired.

Form mental model: After the pilot has interpreted the information he will try to form a mental model of what is going on. A possible mistake in this situation is that the pilot draws the wrong conclusion about what is going on. This can be caused by a lack of experience or by a lack of training. For example one common mistake that has been made by pilots in the past is when they encountered a wind shear. In the centre of such a wind shear the airflow goes down and the airplane will lose lift and will fall if it does not have enough airspeed, but on the outsides the airflow goes up and the airplane will rise. When a pilot flies into such a wind shear the airplane will rise at first. If the pilot decreases the power and lowers the nose in that situation and enters the centre of the wind shear with a low airspeed and the nose lowered the airplane might stall and fall down, which might lead to accidents. This is a typical example of how the wrong mental model of a pilot about a situation can cause serious accidents during a flight.

Define goals: Based on the mental model of the situation that the pilot is in he will define goals that he wants to achieve. If the pilot has not been trained well enough he might set the wrong goals.

Plan actions: Once the pilot has defined his goals he will plan which actions to perform to reach those goals. If the pilot is inexperienced or is untrained he might choose the wrong actions and he will not reach his goals.

Perform actions: After the pilot has planned the actions, he will execute them. These actions will have consequences on the state of the aircraft and/or the environment. This will result in a change in the information that is presented to the pilot and then we have come full circle and the process will repeat itself. To perform the right action it is essential for the pilot to have had the right training.

As can be seen from the cognitive model of the pilot, situation awareness consists of three parts. First the pilot observes information, then he interprets the information so that he understands what it means and finally he places the meaning of the information in a wider context and forms a mental picture of the situation that is occurring.

The cognitive model of a pilot has a resemblance to the BDI (Belief-Desire-Intention) model that has been developed for agents as described in [24]. In the phase called Situation awareness the pilot tries to form *beliefs* about what is going on based on the information he gets from the airplane sensors. He then defines a set of goals or *desires* based on those beliefs. With those goals in mind he plans and performs actions (*intentions*) to achieve those goals.

The process as it is depicted in figure 1.1 is a continuous process. The pilot never stops monitoring the system. This means that the elements that have been described can be performed in parallel. It might be that the pilot during one of the elements of the process sees some very important information and might stop the process and begin the process anew based on the new information.

1.2 The ICE (Intelligent Cockpit Environment) project

Situation awareness is usually considered to be one of the most important aspects when flying an airplane. Most research shows that many human errors in aviation are caused by a lack of situational awareness [6]. A number of systems have been developed to help prevent mistakes while flying by increasing the situation awareness of the pilot. These systems filter the information for the pilot and provide him with only relevant information based on the current situation that the pilot is in and the workload of the pilot. Some of them can also tell the pilot when he forgets to do something or when he is doing something wrong.

The ICE project is a project that has been set up to research the usefulness and effectiveness of intelligent cockpit systems in military as well as other airplanes. A number of research projects are being executed or have been executed in the ICE project. For example a system has been developed that tried to detect what the pilot was doing using a neural network. The neural network was trained on data that was generated by flying a number of flights with the open source flight simulator Flightgear. The results were encouraging, but it became clear that it was very difficult to train the neural network completely because it was not easy to gather enough training and test data. Furthermore a system has been developed that tries to detect the current situation during a flight using a rule base with if-then rules that looks for predefined transition conditions between situations. If such conditions were satisfied the program considered the next situation to be occurring. This system worked fine as long as the pilot did what he was supposed to do, but it was not very flexible, since it did not work correctly when the pilot made a mistake. Projects have been started to make this system more flexible.

Next to the research to such situation recognition systems some research is being done to autonomous flight bots. These flight bots are being designed to imitate the behaviour of a human pilot. One of those projects has as goal to design a system that tries to manoeuvre an F-16 airplane behind an enemy airplane during a dogfight. This system makes decisions based on decision trees that have been defined during the project. Another project is aimed at developing a system that generates and executes a flight script based on a flight plan for a Cessna.

1.3 Related projects

Intelligent systems that try to assist the crew in a cockpit are also called Crew-Assistant Systems (CAS). The ICE project is not the only project that does research to such systems. We will now give a short description of some related research projects. For a more detailed description we refer to [6].

Pilot's Associate

The Pilot's Associate (PA) project has as goal to enhance a military fighter pilots situation awareness, enabling him to make better decisions. The project was started by the United States Air Forces Wright Laboratory and the Defense Advanced Research Projects Agency (DARPA). The architecture of the system consists of knowledge-based subsystems that can communicate with each other. It manages uncertainty using General Electrics Reasoning with Uncertainty Module (RUM). The system adapts the information that is shown to the pilot based on the current situation. It also has the capability to recognize pilot intention and detect pilot errors.

Because the system was developed by different partners integration of the different parts of the system was problematic. Eventually two demonstrations were held in which the system showed that it had difficulties performing in real time. There were also problems with knowledge acquisition and validation.

Rotorcraft Pilot's Associate

The Rotorcraft Pilot's Associate (RPA) project is an extension of the Pilot's Associate project. The project has been started by the US Army Aviation Applied Technology Directorate. It's goal is to adapt the PA project so that it can be applied to an attack/scout helicopter instead of a jet aircraft. When the RPA system was tested it turned out that although the system was not always correct the pilots preferred to fly with the system switched on.

CASSY

The purpose of CASSY is to guide the pilot with the objectively most urgent task or sub task, while avoiding overload of the crew in planning, decision-making and execution. It is developed by the Flight Mechanics and Flight Guidance group of the University of the German Armed Forces in Munich Germany in cooperation with the Dornier company. CASSY primarily focuses on flight planning and pilot error detection in civil (transport) aircraft under instrumental flight rules (IFR).

A working prototype of CASSY has been tested in-flight in a civil transport airplane. The responses of the test pilots were very positive, even though only a part of the system was functional.

CAMA

CAMA is a project that extends the CASSY system to military transport aircrafts. The project was initiated in 1992 by the German Ministry of Defense. CAMA uses a state-based approach (Petri-nets) to model what a pilot should do during a flight. Case-based reasoning is used to evaluate the pilots actual actions and adapt state-transitions so individual pilots differences can be learned.

1.3.1 Evaluation of the related projects

What all the systems that have been described have in common is a situation assessment module that tries to detect what the current situation is during a flight. This is necessary to understand the goal of the pilot, so that the pilot can be assisted as effectively as possible. This situation assessment is not so straightforward because there is a lot of information that has to be processed and the system must be able to detect the current situation even if the pilot fails to assess the situation correctly. To give an example, when a pilot is flying a mission with an F-16 in enemy territory and a missile is fired at him and is detected by the aircraft sensors the pilot will be informed by those sensors. When this happens the pilot might not see the information that those sensors give him and might keep on flying as if nothing has happened. The system must then still be able to conclude that the pilot is under attack even if the pilot does not act as if he is. In other words the system must not only look at the actions of the pilot to determine what the current situation is, but must also look at other sources of information, because the actions of the pilot might not be appropriate for the current situation.

Another thing that the projects that were described have in common is that they were designed for one specific aircraft. This makes the systems inflexible. Ideally the recognition process of the system would work based on a knowledge base that is created for a specific type of aircraft and that can be changed if the system is applied to another type of aircraft. This way the recognition system would not have to be adapted, only the knowledge base would have to be replaced.

1.4 Project goal

The goal of this project is to develop a situation recognition system, such as described above, for the F-16, that can be used in a number of different applications. We can define the following problem description:

Is it possible to build a system that can detect in real time the current situation (e.g. taking off, dogfight, air refueling) during a flight with an F-16 airplane, based on the state of the airplane, some information about the environment and the actions the pilot performs?

Such a system that tries to determine what the current situation is during a flight can be used for a number of intelligent systems in aviation, not only in Crew Assistance Systems. One can think for example that the system can also be used as part of an AI-bot that tries to simulate human pilot behaviour.

Although we focus in this project on the F-16, we will try to design the system so that it will be able to work with other airplanes as well.

From the problem definition and the project description we can deduce the following system requirements:

1. The system should be able to come up with a conclusion based on the information that is available through the sensors of the airplane. This can be information about the state of the aircraft, like the airspeed, about the environment, like the fact that a missile has been fired at the airplane, or about actions of the pilot, like raising the landing gear.
2. The system must be able to function in real time.
3. The system must be reliable. This means that it should be able to detect the correct situation even if the pilot makes a mistake and that it must be able to correct its own mistakes.
4. It should not be too difficult to integrate the system in a larger application like an AI-bot for example.
5. It should not take too much effort to adapt the system to make it work with other airplanes than the F-16.

Chapter 2

Flying an F-16

2.1 Introduction

In this chapter we will describe what an F-16 looks like and what an F-16 pilot does. Furthermore we will describe the situations a pilot might encounter. Because some information about the F-16 is confidential and therefore not publicly available it was difficult to gather enough information. The main sources of information were an official manual of the US Air Force ([2]) and the manual of the flight simulator game Falcon 4 ([14]). Furthermore an interview was conducted with Gideon Reisel, a former F-16 pilot for the Dutch Royal Air Force. The report of this interview can be found in appendix B.

2.2 The F-16

The F-16 is (one of) the most advanced and versatile fighter jets that is being used nowadays. Its dogfighting skills are exceptional, due to its combination of power and manoeuvrability. But that is not all, it also has a good reputation when it comes to performing air-to-ground attacks. The F-16 is relatively easy to fly (some say it is easier to fly than a Cessna [14]), but this does not mean the pilot can just sit back and enjoy! Although the cockpit of an F-16 is designed to give the pilot as small a workload as possible the pilot still has to process a lot of information. To give an idea of the type and quantity of information the pilot has to process the cockpit of an F-16 is described in section 2.2.2.

2.2.1 The different F-16 versions

Because the F-16 is used by many countries, most of which have made adjustments to their F-16's there are a lot of different versions of the F-16. There are four main types though: the F-16A, the F-16B, the F-16C and the F-16D. Of these four types the F-16B and F-16D are two persons airplanes, which are mainly used for training purposes. The F-16A and F-16C are one person airplanes. The main difference between the first two versions (the F-16A and F-16B) and the last two versions (the F-16C and F-16D) is that the engine and radar system were upgraded. The Dutch Royal Air Force has F-16's of the type F-16A and F-16B, which were upgraded in the 1990's in the Mid Life Update (MLU) program to resemble the F-16C and F-16D.

At the start of the project a choice had to be made between these different versions of the F-16. Because Falcon 4 is modelled after the F-16C and because the F-16's that were updated in the MLU program resemble the F-16C, the decision was made to focus our research on the F-16C.

2.2.2 The F-16 cockpit

To give an idea of the information that an F-16 pilot has to process, we will show what an F-16 cockpit looks like (figure 2.1). The pictures that are shown in this section are not pictures of a real F-16 cockpit, but of simulators that resemble the F-16 cockpit as much as possible. The pictures shown here are pictures of the cockpit of an F-16C.

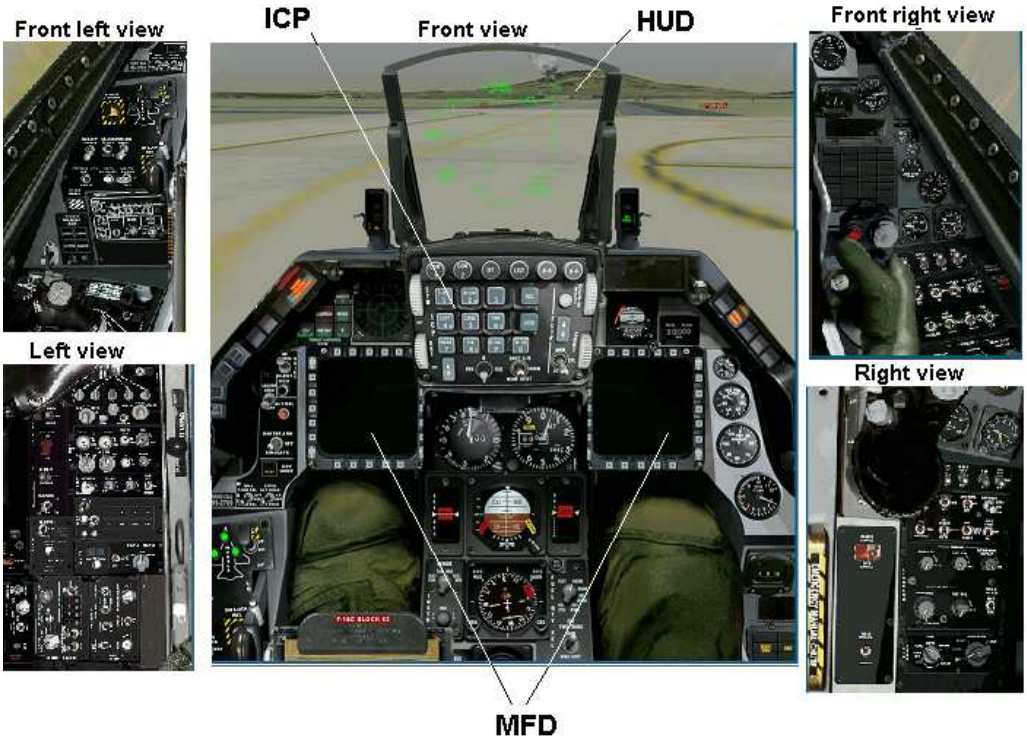

Figure 2.1: The cockpit of an F-16C.

We will now describe some of the most important instruments in the cockpit. The main source of information for an F-16 pilot is the Heads Up Display (the HUD). The HUD can display a lot of different information, depending on the settings of some of the instruments. Next to the HUD the pilot has two Multi Function Displays (MFD's). These MFD's, like the HUD, can display lots of information. The MFD's can display the same information. In the F-16A there was only one MFD. The reason that there are two in the F-16C, is because in some situations it is necessary for the pilot to have access to different kinds of information that can not be shown on a single screen. The pages that can be displayed on the MFD are described in appendix D. Between the MFD's the standard flight instruments are placed that show information like altitude, attitude, airspeed, heading and vertical velocity. These are the instruments that give the most important information and that will be checked by the pilot most often. But as can

be seen in the pictures, they are by far not the only instruments in the cockpit. Of all the instruments that are mentioned in this report the function and their position in the cockpit are described in appendix D.

2.3 A flight with an F-16

During a flight with an F-16 a number of different situations can be distinguished. To give a global idea of what a pilot has to do in those different situations we give a short description of those situations in section 2.3.1. First the relations between the situations will be described, so that it becomes clear in what order the situation might occur during a flight. The order in which the situation might occur is illustrated in the figures 2.2 and 2.3. In figure 2.2 all situations have been plotted against the time.

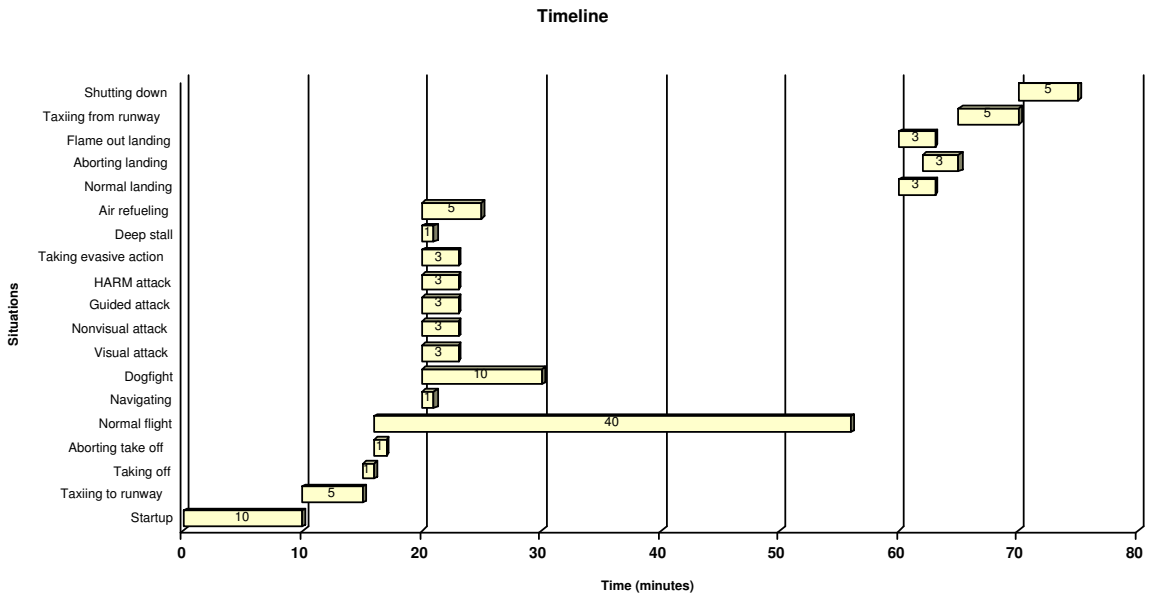

Figure 2.2: A timeline with all possible situations.

The situations after taking off and before landing are not situations that all start at the same time, but they are situations that can occur in random order. The durations of the situations as they are displayed in the figure are not exact durations of the situations, but they are maximum durations, called the time windows of the situations. The time windows are estimates that were given by Gideon Reisel during our interview (see appendix B).

The starts and ends of the situation are not always clear. One situation may lead to another situation without there being an exact moment at which one can say that the first situations has finished and the second situation has started. This means that some of the situations may overlap and that there are moments at which it is not sure which situation is occurring.

To further clarify the chronological order of the situations and the transitions from one situation to another a state diagram has been created and is included as figure 2.3.

Because we can not clearly define the conditions for which we move from one state to another the transitions between the states (or situations) have not been described in the diagram, we only indicate that there might be a transition from one state to another at some point during a flight.

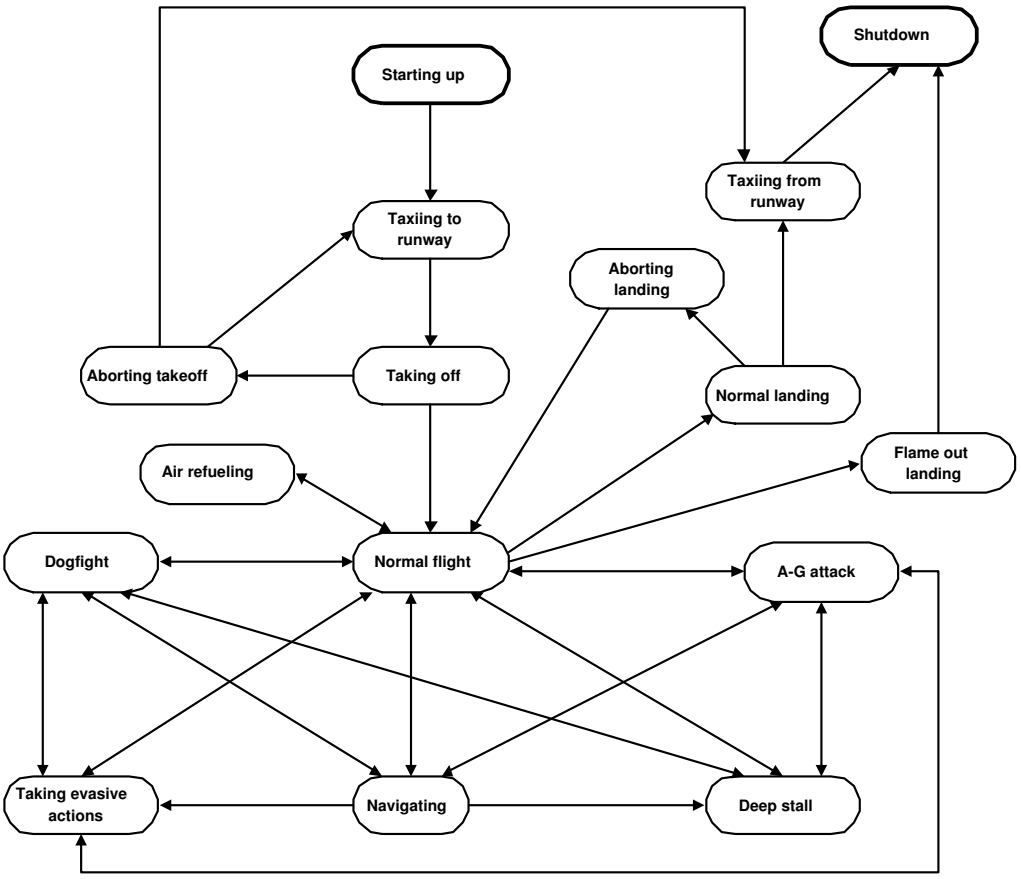

Figure 2.3: A state diagram describing the situations and their transitions.

2.3.1 The situations a pilot might encounter

We will now give short descriptions of the situations that the pilot might encounter during a flight. For more extensive descriptions see [15].

Startup: During startup the pilot has to make sure the airplane is ready for takeoff. He has to check all the systems and configure the aircraft for the mission he is going to fly. It is possible for pilots to perform the startup in less than four minutes, provided that the airplane has been prepared in advance (which is done in situations in which pilots might have to be in the air very fast), but normally the startup will take approximately 10 minutes.

Taxiing to runway: During this phase the pilot moves the airplane to the runway which he will takeoff from.

Taking off: During the takeoff the pilot will get the airplane in the air. The pilot might have to abort the takeoff which is considered a separate situation.

Aborting takeoff: A takeoff can be aborted for various reasons. The pilot can abort a takeoff as long as he has not reached the calculated Refusal Speed. This speed is determined during the briefing and depends among other things on the weight of the aircraft and the length of the runway. Depending on the reason the takeoff was aborted the pilot might choose to try to takeoff again. The only limiting factor is the temperature of the wheel brakes. They might need to cool down before the landing gear is raised.

Normal flight: In the situation normal flight the pilot is flying towards a steerpoint. A steerpoint is a geographic location that is defined in the flight plan of the flight. The pilot has to fly to all steerpoints in the flight plan. The last steerpoint is the airport the pilot will land at. The situation normal flight is the situation in which the pilot will find himself most of the time.

Navigating: In the navigating situation the pilot can either be selecting a new steerpoint or he can check his course for the currently selected steerpoint. The pilot usually follows the INS steering in the HUD to stay on the track to the next steerpoint. The heading indicator can not be used to stay on this track, because the heading depends on the wind. The stronger the wind the more the nose has to be turned into the wind to fly a straight path.

Dogfight: When the pilot is engaged with an enemy fighter his survival depends on his piloting skills. This is one of the situations in which there is a danger of information overload of the pilot. The attention of the pilot will be mainly directed to the outside environment during a dogfight because he will be trying to keep his eyes on the enemy fighter. Therefore the pilot will give little attention to the instruments in the cockpit and will not want to be distracted by those instruments.

Visual attack: A visual attack is an attack on a ground target which the pilot can see with the naked eye. There are four different kinds of visual attacks. These are:

- The strafe attack: In this attack the pilot shoots at the ground target with his air-to-ground gun.
- A rockets attack: In this attack the pilot tries to hit the target using the rockets.
- The CCIP (Continuously Computed Impact Point) attack: In this attack the pilot tries to bomb the target using the CCIP HUD symbology.
- The DTOS (Dive Toss) attack: In this attack the pilot tries to hit the target in a dive using the DTOS HUD symbology.

Nonvisual attack: A nonvisual attack is an attack at a ground target that the pilot can not see with the naked eye. A nonvisual attack is performed using one of the following three methods:

- The CCRP (Continuously Computed Release Point) method: In this method the FCC (Fire Control Computer) continuously computes the point at which the bombs should be released. This point is indicated in the HUD. The pilot has several options. He can either drop the bombs in a dive, he can drop the bombs in level flight or he can climb and loft the bombs on the target.

- The LADD (Low Altitude Drogue Delivery) method: In LADD the pilot does not have a choice of how to release the bombs, the only way to drop the bombs is to climb and loft the bombs on the target in a 45 degrees angle.
- The BCN (BeaCoN) delivery method: This method is used when the target's position is only known relative to a ground-based radar beacon.

Guided attack: A guided attack is an attack with a laser guided or heat seeking bomb. An example of a heat seeking bomb is the Maverick. There are several kinds of LGB's (Laser Guided Bombs). The HUD symbology for an LGB attack is similar to the CCRP HUD symbology.

HARM (High-speed Anti-Radiation Missile) attack: The HARM is specially designed to be used against targets that emit radar waves. For this weapon there is a special radar mode, the HTS (Harm Targeting System) mode. The HARM is a very flexible missile that is able to shoot a target that is located behind the aircraft. This reduces the missiles range however, because it will lose a lot of energy in making the turn. It is recommended that the pilot turns the aircraft to aim at the target if that is possible.

Taking evasive action: In this situation the pilot is alerted of an incoming missile by the RWR (Radar Warning Receiver). The RWR is only able to detect radar guided missiles, so heat seeking missiles are not detected and should be spotted by the pilot himself. A good way for a pilot to get away from installations that do not shoot missiles but bullets is to change a dimension every few seconds. This simply means the pilot has to manoeuvre a lot which makes it a lot more difficult for the enemy installations to predict the next position of the airplane, which makes aiming the bullets much more complicated.

Deep stall: Deep stall is a special kind of stall. When the airplane threatens to enter a deep stall a loud horn will sound to warn the pilot. If this occurs the airplane is said to be in the "dog house". Figure 2.4 shows the combinations of airspeed and AOA that form the dog house. A normal stall is a lack of energy from which the pilot

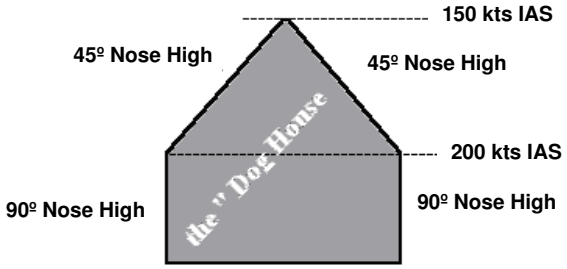

Figure 2.4: The Dog House.

can recover by lowering the nose and increasing the throttle. In such situations in which the airplane is difficult to control the FLCS (Flight Control System) will try to take control of the airplane and the pilot will be shut out of the control loop. However the FLCS is usually not able to get the airplane out of a deep stall. Just

lowering the nose will not help in a deep stall. The following actions should be taken by the pilot to get out of a deep stall:

- **Release controls:** By releasing the controls the pilot gives the airplane the chance to recover by itself. He gives the FLCS free reign to try to get out of the deep stall.
- **Throttle to idle.**
- **Set rudder in opposite yaw direction:** This is only necessary if the airplane is inverted and spinning. The rudder should be used to try to stop the spinning.
- **Switch the MPO:** This should be done if the FLCS is not successful in getting the airplane out of the deep stall. By switching the MPO (Manual Pitch Override) to OVERRIDE the pilot overrides the FLCS and regains control of the flight controls.
- **Cycle in phase:** The nose of the airplane will be going up and down. The pilot should now take the controls and try to get in phase with this bobbing. So when the nose comes up the pilot should pull (if the airplane is upright) and when the nose goes down he should push. The pilot will know he is out of the deep stall if the nose stays below the horizon and does not try to come up again. At this point the pilot should keep the nose down and increase speed until he has reached 200 knots.

Air refueling: Air refueling is for most beginning pilots a difficult manoeuvre because it requires very precise steering. It is up to the pilot of the F-16 to approach the tanker so that the tanker boom can be connected to the refuel door of the F-16. When the pilot has finished refueling he should inform the tanker he wants to disconnect. After the boom operator has disconnected and the boom is well away from the airplane the pilot can continue his mission.

Landing: The F-16 has a system that tries to detect when the pilot is trying to land the airplane. This detection is based on the angle of attack and the airspeed of the airplane. When the airplane detects the pilot is trying to land it will give hints as to whether the pitch angle is good or that the pilot should lift or lower the nose. This built in detection system is useful, but easily misguided, it might for example think the pilot is landing when he is trying to make a slow turn.

A landing can be either a normal landing or an emergency landing. The landing procedures for an emergency landing are generally almost the same as for a normal landing. The only difference is that the pilot should jettison the stores and might deploy the hook or the emergency parachute during an emergency landing.

Flame out landing: A flameout landing is a special kind of emergency landing. A flameout landing is a landing with no engines. If the engines of an F-16 shut down the pilot can try to glide to a nearby airfield to land the F-16. The F-16 is capable of covering 1 NM (Nautical Mile) for every 1000 feet altitude, depending on wind and aircraft configuration (weight and drag). The pilot should try to maintain the optimum glide speed during the landing to cover as much distance as possible. It is impossible to correct a shortage of energy while an excess of energy can be corrected easily using the speed brakes.

Aborting a landing: Aborting a landing in an F-16 is not different from aborting a landing in another airplane. The pilot should add power, close the speed brakes and raise the landing gear.

Taxiing from runway: After the pilot has successfully landed the airplane he will taxi the airplane to the hangar, where he will shut down all the systems.

Shutting down: This is the phase in which the pilot shuts down all systems.

Chapter 3

Artificial intelligence techniques

3.1 Introduction

In this chapter we will describe some techniques that are used in the field of artificial intelligence. It is not possible to discuss all techniques here, so we have made a selection of the most interesting techniques for our problem. From these techniques we have to select those that will fit our purposes best. The choice we make will determine the performance of our system. In the next sections short descriptions will be given of the possible techniques with some of the advantages and disadvantages of each technique.

3.2 Bayesian belief networks

Bayesian belief networks are often used in reasoning systems. They have proven themselves in a number of applications. In a Bayesian belief network one can indicate the effect an event has on another event. Bayesian networks can be visualized as directed graphs. Given some evidence (some facts should be specified as being either true or false with a certain probability) the probability of the events one is interested in (the query events) can be calculated. One can also state that it is not completely certain an event has happened, but that it happened with a certain probability. This probability is then taken into account when the probability of the queried event is calculated. In the following section we will give an example of a Bayesian belief network.

3.2.1 Inference in Bayesian belief networks

To illustrate how we use Bayesian belief networks to combine probabilities and allow for uncertainty in the input data we will give a small example. Suppose we want to know what the probability is that a pilot is landing at a given moment. To determine this probability we will look at two things. One is the probability that the pilot has contact with the air traffic control tower and the other is the probability that the pilot is performing an approach. Suppose we know the probability that the pilot has radio contact with the tower based on the fact that the radio channel is set to the tower channel ($P_{radio} = 0.6$) and that we have to calculate the probability that the pilot is performing the approach ($P_{approach}$) from the fact that he has set the throttle to IDLE

and that he has lowered the landing gear. Furthermore we know the separate conditional probabilities that the pilot is performing an approach given that he has lowered the landing gear ($P(approach|gear) = 0.8$) and that he is performing an approach given that he has set the throttle to IDLE ($P(approach|throttle) = 0.5$). Suppose that we do not know for sure that the pilot has set the throttle to IDLE, for example because he has set it to a little bit above IDLE and we are not sure if he meant to set it all the way to IDLE. Thus the throttle is set to IDLE with a probability lower than 1 ($P_{throttle} = 0.9$). This would result in the Bayesian belief network shown in figure 3.1.

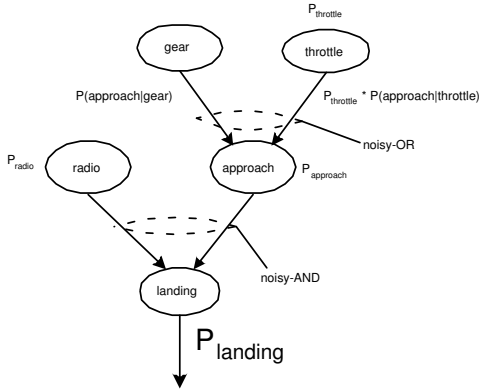

Figure 3.1: An example of inference in a Bayesian belief network.

Now there are two tools that we can use to calculate the probability that the pilot is landing. These are the noisy-OR and the noisy-AND inference methods that are described in [12]. The mathematical formula of the noisy-OR model is equivalent to Bernoulli's rule of combination which is a special case of Dempster's rule of combination (see [27]). Using these methods would lead to the following calculation if it is given that the pilot has lowered the landing gear, set the throttle to IDLE and the airplane is descending:

$$\begin{aligned} P_{approach} &= 1 - ((1 - P_{throttle} * P(approach|throttle)) * (1 - P_{gear} * P(approach|gear))) \\ &= 1 - ((1 - 0.9 * 0.5) * (1 - 1 * 0.8)) \\ &= 0.89 \\ P_{landing} &= P_{radio} * P_{approach} \\ &= 0.6 * 0.89 \\ &= 0.534 \end{aligned}$$

We will now discuss some of the advantages and some of the disadvantages of Bayesian belief networks.

Advantages: A lot of research has been done on Bayesian belief networks and they are being used in a lot of applications.

Disadvantages: Bayesian belief networks can be very time consuming and probability inference in complicated belief networks has been shown to be NP-hard [5]. A lot of research on real-time inference algorithms has been done though and a couple of approximation algorithms have been developed.

Another disadvantage is that the probabilities used in the Bayesian belief network should be very precise and should in the ideal case be derived from data sets. This can be a problem for our application because we do not have any experience with flying an F-16.

3.3 Certainty factors

The certainty factor model, which was introduced by Shortliffe and Buchanan, is a model that can be applied to expert system using a rule base for the reasoning process. Certainty factors can be used if one wants to state that for a rule *"if fact then conclusion"* the conclusion is not entirely certain, but only with a certain probability or certainty. That probability is called the certainty factor. This might seem a very interesting model, but there is one drawback. The certainty factor model was very popular in expert systems in the 1980's but has gotten so much criticism over the years that it has been abandoned. The criticism was based on the its ad-hoc nature of the model, its mathematical shortcomings and in general the unsatisfactory theoretical justification (see [31]). Nowadays it is only considered to be interesting from a historical point of view [12]. In most applications the certainty factor model has been replaced by Bayesian belief networks, since the two models have a lot in common [12].

3.4 Dempster-Shafer theory

The Dempster-Shafer theory is a generalization of the Bayesian belief model. It is a mathematical model to model a person's belief in a fact. It provides a method to combine measures of belief in a fact induced by different pieces of evidence, the so called Dempster's rule of combination. In the Dempster-Shafer theory the probabilities of the facts do not all have to be specified exactly. So it is possible to reason with only partial information.

Advantages: One of the advantages of the Dempster-Shafer theory is that the degrees of belief for a fact can be obtained from the probability distribution for another fact [28]. Also the Dempster-Shafer theory has a very thorough mathematical basis and is widely used to reason with uncertainty.

Disadvantages: Because the Dempster-Shafer theory is a mathematical model it can be quite complicated and in contrast to Bayesian belief networks the reasoning process is not very intuitive. The calculations can also be very time consuming.

3.5 Fuzzy logic

Fuzzy logic can also be used to represent uncertainty. Fuzzy logic makes it possible to convert a crisp value to a fuzzy value and reason with that fuzzy value to come to a fuzzy conclusion. This fuzzy conclusion can then be defuzzified to get a crisp conclusion. For example, we can define the fuzzy set for airspeed as A={FAST, NORMAL, SLOW} and throttle setting T={HIGH, MEDIUM, LOW} as visualized in figure 3.2. Suppose we have two rules in our knowledge base:

if the airspeed is SLOW then the throttle should be set to HIGH
if the airspeed is NORMAL then the throttle should be set to MEDIUM

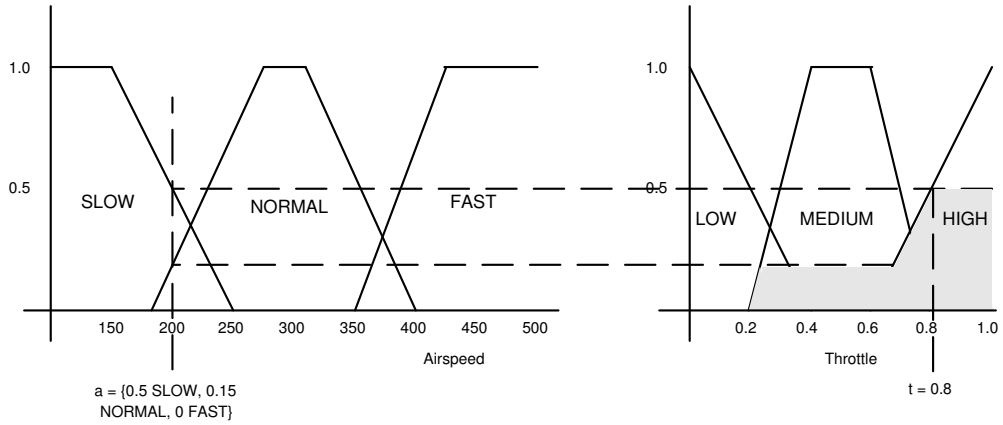

Figure 3.2: An example of fuzzy reasoning.

And suppose we have an airspeed of 200 knots, what should the throttle be set to? To calculate the answer to that question we would use the reasoning method shown in figure 3.2. The crisp airspeed value of 200 can be fuzzified as a={0 FAST, 0.15 NORMAL, 0.5 SLOW}. Now we can reason with this fuzzy value and the rules in the rule base to get a fuzzy value for the throttle setting as shown in the figure. This fuzzy value would have to be defuzzified, which can be done in several ways. The method that is shown in the figure is called the First Maximum defuzzification method. This method results in a throttle setting of 0.8, as shown in the figure.

Advantages: Fuzzy logic is a nice method because it is intuitive and it makes it possible to define the rules in normal language.

Disadvantages: There is no completeness in the inference formalism and the mathematical basis for fuzzy logic is rather coarse [19].

3.6 Markov models

Markov models are statistical models of sequential data that can be used to recognize a pattern in a sequence of data in some time series. A Markov model can be visualized using a Bayesian network. An example is given in figure 3.3.

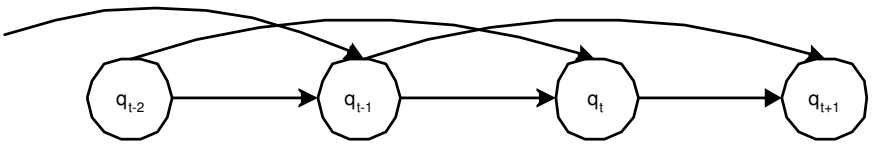

Figure 3.3: A Bayesian network representing a Markov model of order 2.

Markov models have successfully been applied to speech recognition and pattern recognition applications. They are particularly useful to analyse a sequence of data that can be characterized as a signal. Markov models can be characterized by the following

equation ([3]):

$$P(q_1^T) = P(q_1^k) \prod_{t=k+1}^{T} P(q_t | q_{t-1}^{t-k})$$

Where q_t is a state variable, k is the order of the Markov model and q_{t-1}^{t-k} characterizes all relevant past information. This equation means that the probability that the state q_1 is reached depends on the probabilities that the previous k states have been reached and the effect that reaching those states have had on reaching the state q_1.

Advantages: Markov models have successfully been applied in a number of pattern recognition applications.

Disadvantages: To use Markov models one needs precise information about the probabilities of the state transitions. This is not always easy to obtain.

Chapter 4

The architecture of the system

4.1 Introduction

As mentioned in the introduction of this report the goal of the project is to create a system that detects the current situation during a flight with an F-16. In this chapter we will describe the recognition process for such a system. In section 4.2 we will give an overview of how the system will come to a conclusion about the situation that is occurring. In section 4.3 we will define the requirements of the system. These are not the complete software requirements, but are performance requirements for the system. Then in section 4.4 an overview of the architecture of the system will be given and the elements of the architecture will be discussed.

4.2 Overview of the reasoning process

As mentioned before the information we get from the airplane sensors is information about the environment (for example a missile that has been launched at the airplane), the state of the airplane (for example the airspeed) and the actions of the pilot (for example lowering the landing gear). Our system will use all this information to determine which one of a list of possible situations is occurring. This list of situations is defined in a knowledge base. For every situation in the knowledge base the state of the airplane, the possible actions of the pilot and some events that might happen during the situation have been defined. This information describes the situations. The change of a variable of the airplane (because of a change in state, an action or an event) can concern more than one situation at a time. This is depicted in figure 4.1. During a flight the system will compare the information it receives from the sensors with the information in the knowledge base and it will try to determine which situation is occurring.

When we look back at figure 1.1 we can compare this approach with the way a pilot tries to assess the situation. We see that the pilot forms a mental model of the current situation based on the information he gets from the airplane sensors about the airplane state and the information he gets from the environment. Although our program is limited in the amount of information it gets from the environment it has an advantage that the pilot does not have. The system can look at the actions of the pilot to try to determine the pilot's goal. The pilot will have a different goal for every situation and thus if the system knows what the pilot's goal is it will know the current situation.

But there is a danger in this approach and that is that if the pilot forms the wrong

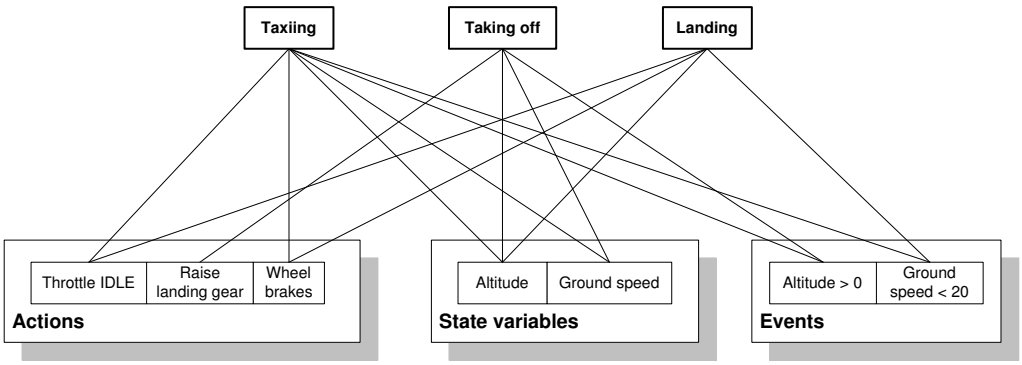

Figure 4.1: A tree that shows how a situation can be described by the airplane variables.

mental picture and performs the wrong actions or forgets to perform some actions, the system might come to the same conclusion as the pilot, which is the wrong conclusion. To prevent this from happening we need to define some characteristic events for the situations from which we can conclude that the situation is happening or is not happening independent of the actions of the pilot. For example if a missile warning is received and the pilot keeps on flying as if he is in a normal flight the system should still conclude that the airplane is under attack even if the pilot does not act as if it is.

This does not mean that if the pilot performs an action that he is not supposed to do, that action should be ignored. To give an example, the pilot might set the throttle to full during a landing if he has to abort the landing for some reason. This is not an action that he should do during a normal landing, but this does not mean it is a wrong action. If the pilot performs more actions to abort the landing like raising the landing gear and retracting the speed brakes, the system should conclude that the pilot is aborting the landing. If these actions would be ignored the system would never be able to detect that a pilot is aborting a landing. To be able to discriminate between these possibilities the system will have to look at a lot of variables and has to be critical about the actions of the pilot.

4.3 The requirements

In this section the requirements that have been defined in the introduction will be extended. The system will eventually be tested to see if it fulfills these requirements. The requirements are the following:

1. There must be a way to extract data about the state of the airplane, the actions of the pilot and information about the environment from the flight simulator.

2. The system must be able to function in real time. This means that the system must reach a conclusion as fast as possible and that the pilot can keep flying. It is very important that the flight simulator is not slowed down significantly by the system to keep the flying as close to the real thing as possible.

3. The system must be reliable. The reliability can be split up into two elements:

   (a) The system must be able to correct its own mistakes. It is unrealistic to assume that the system will be infallible. It might not detect that a situation has

started or ended. In that case the system must not get stuck in the situation it is currently in. This means among other things that the chronological order of the situation as it is given in figure 2.3 should not be a strict order for the system, it must be able to go from one situation to another one that is not one of the following situations.

(b) It must be able to cope with pilot mistakes. This does not mean the system should be able to detect these mistakes and report them to the pilot. It means that if a pilot for example forgets to perform one of the actions he should have performed the system should still be able to come up with a good conclusion about the current situation. Detecting pilot errors is something that might be the subject of a future research project.

4. The system should be designed in such a way that it can easily be incorporated with a larger system like an AI-bot. This means that the conclusion that is drawn by the system must be understandable for other systems or it that it should at least take little effort to make it understandable.

5. The system must be able to work with other airplanes than the F-16. If a knowledge base for a Cessna is created and the pilot flies a Cessna in the flight simulator, the system should be able to work as well as for the F-16. This means that all of the knowledge should be defined in the knowledge base and none of it in the implementation.

4.4 The system architecture

Figure 4.2 shows an overview of the architecture of the system that will be created to detect the situation the pilot is in.

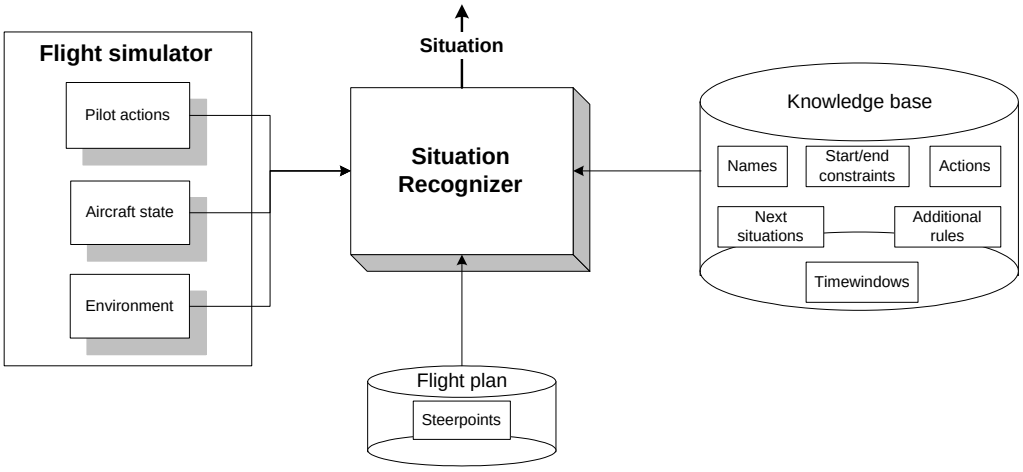

Figure 4.2: The system architecture.

4.4.1 The flight simulator

Because we do not have a real F-16 simulator available we will use a flight simulator on a PC to test the system. This flight simulator will be used to monitor the pilot actions, the state of the aircraft and the environment. Because not every flight simulator provides the possibility to retrieve this data from the simulator with an external program, the choice of the flight simulator was an important one. After analysing several possibilities, among which the open source flight simulator Flightgear and the F-16 simulator Falcon 4, we have eventually chosen to use the Microsoft Flight Simulator 2002. The reasons for this choice and how this simulator was used is explained in appendix A.

4.4.2 The knowledge base

The knowledge about how to fly an F-16 has been gathered and stored in a knowledge base in a preliminary research project (see [15]). The contents of the knowledge base will be used to determine which situation is most likely to be occurring at a certain moment. The knowledge base has been stored as an XML file of which the schema, which is the definition of the structure of the knowledge base, is included in appendix C. This schema can be used to create knowledge bases for other airplanes than the F-16. These knowledge bases can then be used to make the Situation Recognizer work with those other airplanes.

The knowledge base is divided according to the situations that have been described in chapter 2. The system will only be able to recognize those situations for which information has been stored in the knowledge base. The knowledge base will contain the following information for the situations:

Names: These are unique names for the situations.

Actions: A situation has a list of actions that should or could be performed by the pilot in that situation. Some of these actions should be performed in a specified chronological order, while others can be performed at any time during the situation. By monitoring the actions of the pilot the system can establish what the goals of the pilot are. If the pilot performs a lot of actions that belong to a certain situation the probability that the pilot is trying to achieve the goal that belongs to the situation and thus that the situation is occurring will become higher.

An action always has its effect on a control or instrument of the airplane. For example if the pilot moves the stick forward the positions of the elevators will change. For simplicity, when looking at the actions of the pilot we will look only at the effect that those actions have for the state of the airplane. We assume that there will be no malfunctions in the aircraft controls that will cause discrepancies between the actions of the pilot and the state of the airplane. We will also assume that the time between the action of the pilot and the change of the aircraft state will be so small that it can be neglected. An example of such a time difference is when the pilot lowers the landing gear, because it may take a few seconds before the landing gear is lowered completely.

Because of this direct relationship between action and airplane state we have chosen to represent an action in the knowledge base by a variable or an instrument getting a certain value. For every action the following information is stored:

The name: This is the name of the instrument or variable whose value changes as a result of the action.

The value: This is the value that the instrument or variable will have after the action has been performed.

A priority value: The priority value is an indication of the importance of the action for the situation it belongs to. The priority values are values between 0 and 1. A priority value below 0.5 means that the action might never be performed during the situation. A priority value of 0.5 or above means that the action is compulsory. A priority value of 1 means that the action is vital for the successful completion of the manoeuvre.

A probability value: The probability value states the likeliness that a situation is occurring when the variable or instrument that corresponds to the action is set to a specific value. There could be some confusion about the difference between a priority value and a probability value for an action. This can be prevented by remembering that the priority value says something about the importance of the action, while the probability value says something about the effect of the action. To give an example of the difference between these two values, suppose the pilot is aborting a landing. The pilot is not obligated to lower the hook when he aborts a landing, which means the *priority* value of the action lowering the hook is 0, but if the pilot does lower the hook the *probability* that he is aborting a takeoff is high.

Because we do not have any experience with flying an F-16 we can not know the exact values that the probabilities that have to be specified in the knowledge base should have. Therefore these probabilities are not given as crisp values but as general indications of the probabilities. The following values are used (from highest probability to the lowest): VBP (Very Big Probability), BP (Big Probability), MP (Medium Probability), SP (Small Probability), VSP (Very Small Probability). For the eventual implementation these values will have to be converted to a crisp value between 0 and 1 and the best values will have to be determined by experimentation.

Start/end constraints: The start and end constraints describe the state of the airplane at the start and at the end of the situation. For example, at the start of the taking off situation the altitude must be 0 and at the end of the takeoff it must be greater than 0. These are conditions that will always be satisfied at the start or end of the situation, even if the pilot forgets to perform some actions. The conditions apply to the same controls and instruments as the actions. The following information is stored for the conditions:

The names: These are the names of the instruments or variables that have a starting and/or ending constraint.

Start and end conditions: The conditions for the instruments or variables values that have to be satisfied at the start or end of the situation. These conditions can be single values, a set of values combined with a logical operator (NOT, OR or AND) or a range of values.

A start probability: This probability indicates how big the chance is that the situation has actually started when all start conditions are satisfied. This value is one of the values of the set that is described in the actions sections above.

An end probability: This probability indicates how big the chance is that the situation has actually ended when all end conditions are satisfied. This value is also one of the values of the set that is described in the actions sections above.

Additional rules: One can imagine that during a flight events can occur from which can be concluded immediately that it is very likely that a situation is occurring or that a situation has ended and another one has started. For example if the pilot is in normal flight and the altitude becomes lower than the minimum altitude it is likely that the pilot has started a landing and that the normal flight situation has ended. This kind of information is stored in the knowledge base as additional rules. The additional information says something about the probability that a situation is or is not occurring given the value of a single variable. Therefore this additional information looks a bit like the start and end constraints. But where the constraints also say something about the *possibility* that a situation has started or ended given a *set* of variable values, the additional rules only say something about the *probability* that a situation has started or ended given the value of a *single* variable. The additional knowledge is further described in [15].

An additional rule has the following attributes:

The name: This is the name of the instrument or variable.

A start or end attribute: If the rule says something about the probability that the situation has started and is occurring it will have a start attribute, if it says something about the probability that the situation has ended or is not occurring it has an end attribute. The value of the attributes is the value that the variable has to have if the rule is to fire.

A probability value: This is the probability that a situation is or is not occurring if the rule fires. If the probability value is 1, the situation will certainly be occurring or will certainly not be occurring.

Time windows: The time window of a situation is the maximum duration of a situations. The time windows of all the situations are displayed in figure 2.2. How the time windows are used to reason about the situations is explained in chapter 5.

Next situations: As was explained in section 2.3 and shown in figure 2.3 situations do not occur in a random order. After one situation has finished there is only a limited amount of other situations that can follow it. This information is stored in the knowledge base by storing for every situation all the situations that can follow it. Our system should not consider this order a strict one, but only an indication of what to expect. If the system would consider it a strict order and it would fail to detect a situation it would get stuck in one situation and might never get out of it again.

4.4.3 The flight plan

The order in which the situations may occur that is depicted in the state diagram in section 2.3 is an order that always applies for all flights. But when an F-16 pilot is on a mission he will have a flight plan that he should follow and that specifies the steerpoints he has to track. Such a flight plan could be used by our system as an indication of the order in which the situations might occur. This way the system can predict the situations that will come and will be able to check that predicted situation with the situation it detects itself.

A flight plan can be specified as an XML file. The structure that the flight plan should have and an example of a flight plan are shown in appendix C.

A flight plan is a sequence of steerpoints the pilot will have to fly to. A flight plan contains the following information for a steerpoint:

Coordinates: These are the coordinates of the steerpoint in longitude and latitude.

Airspeed: This is the airspeed that the pilot should fly at to reach the steerpoint in time.

Heading: This is the heading at which the pilot should fly to reach the steerpoint.

Altitude: The altitude at which the pilot should fly over the steerpoint.

TOS (Time Over Steerpoint): The time at which the pilot should arrive at the steerpoint.

Distance: The distance to the steerpoint from the previous steerpoint.

Situations: The situations that should occur at the given steerpoint. For example at the first steerpoint (the departing airfield) the situation starting up, taxiing to runway and taking off will all occur.

Chapter 5

The Situation Recognizer

5.1 Introduction

In this chapter we will zoom in on the Situation Recognizer. In section 5.2 we will describe three possible reasoning models that can be used for the recognition process. Then in section 5.3, we will describe the architecture of the Situation Recognizer, show the different elements of the system and explain their functionality. Then in section 5.4 we will explain which of the techniques that were described in chapter 3 has been chosen to implement the probability inference and how that technique can be used.

5.2 Possible reasoning models

The Situation Recognizer is the part of the system that will do the actual reasoning. In chapter 3 we have discussed some of the techniques that can be used for this reasoning process. In this section we will describe three models that can be used to implement the Situation Recognizer in which some of those techniques will be used.

5.2.1 Model A: A finite state model

The finite state model is a very simplistic model. The basis for this model is the state diagram shown in figure 2.3. As explained before, the transitions between the states are very vague and can not be defined exactly. In the finite state model it is assumed that this vagueness does not exist and that we *can* define these transitions. The reasoning process of this model receives the information from the airplane sensors and checks whether the conditions for one of the transitions from the current situation are satisfied. If so the situation at the other end of the transition will become the current situation.

This model can be implemented using an expert system with a rule base of if-then rules that check if the conditions of a transition have been satisfied. As described in chapter 1 such a system has been developed for a normal airplane like a Cessna in the ICE project and works reasonably well if the pilot performs all actions he should perform during a situation and the whole flight is flown according to plan. If the pilot forgets to perform an action or performs an action he should not have performed the system will make a mistake. This is not the desired behaviour, because the system should be able to cope with pilot mistakes and therefore we will not use this model for the Situation Recognizer.

5.2.2 Model B: A probabilistic model

Since the finite state model does not satisfy the requirements we need a more flexible model. This flexibility can be achieved using a probabilistic model. Basically this model calculates the probability that a situation is occurring or that it is not occurring based on the state of the aircraft, information from the environment and the actions of the pilot. The reasoning process of this model is depicted in figure 5.1.

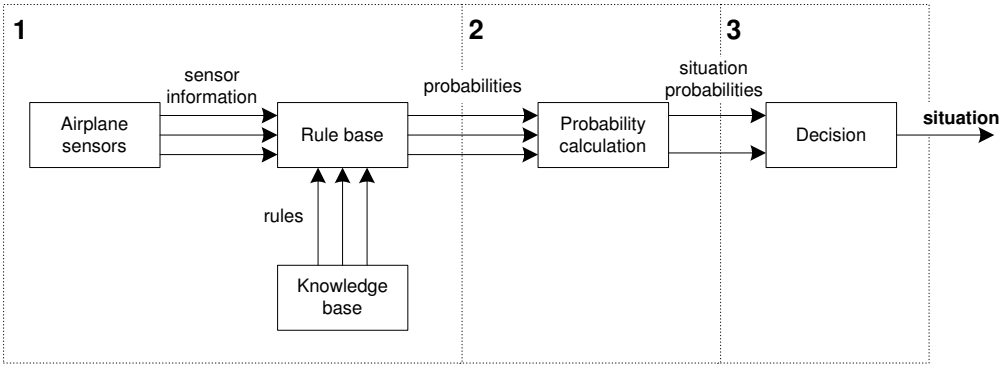

Figure 5.1: The reasoning process of the probabilistic model.

The reasoning process of the probabilistic model consists of three phases:

1. Compare the data from the airplane sensors with the data in the knowledge base and generate a set of probabilities that the situations are occurring or are not occurring. This can be done by an expert system with a rule base that is created from the data in the knowledge base.

2. Combine the probabilities for the situations and calculate for every situation the probability that it is occurring. Also calculate for the current situation the probability that it is not occurring. This should only be done for the current situation, because the probability that a situation that is not the current one is not occurring is useless. For these calculation we can use one of the techniques described in chapter 3.

3. Look at the probabilities that the situations are occurring or are not occurring and decide which situation is most likely the current one.

This process is repeated at some predetermined time interval. This interval should not be too long, because events can happen very fast in an F-16. For the program to be useful it will have to detect changes in the aircraft state as soon as possible. This means that the process should be repeated a couple of times per second.

An important aspect of the probabilistic model is that the probabilities of all situations are calculated at the same time. Because of this it is not a problem that the transitions between situations are vague and that situations might overlap. This just means that those overlapping situations will have probabilities that will be close together. In that case the system will choose that situation with the highest probability. Another advantage is that this feature enables the system to correct its own mistakes because if the system does make a mistake and detects the wrong situation it will detect the correct situation once it has received enough indications that that situation is occurring and not the other one.

Another important aspect of the probabilistic model is that we can make the effect of some actions or events bigger than the effect of other actions or events. This is done by assigning different probabilities to different actions or event. If this is done the probability that a situation is occurring or is not occurring will rise more if one action is performed (or event occurs) than if another is performed (or occurs). This means that a lot of insignificant actions that indicate that the situation is occurring can be outweighed by an important action that indicates that the situation is not occurring.

Furthermore the fact that a pilot may make a mistake is accounted for because that will only result in the probabilities of a situation becoming higher or lower, which usually does not immediately lead to the system making an error. Except when the pilot makes a lot of errors the system might make a mistake. But even if the system makes a mistake and detects the wrong situation or fails to detect a situation it will be able to recover itself, because it keeps calculating the probabilities of all situations.

In short the probabilistic model calculates the probability that a situation is occurring based on the conformance of the aircraft state and the actions of the pilot to the data about that situation which is stored in the knowledge base.

5.2.3 Model C: A causal model

The causal model is an extension of the probabilistic model. It has the same three phases in which the recognition process is performed. It has the following differences compared to the probabilistic model:

1. Relations between the actions of the pilot or between events are taken into account. This means that if (failure of) performing an action by the pilot has consequences for the probability of another action, that probability will change. One can think for example of two actions that should be performed in chronological order. If the first action is not performed when the second action is performed the effect of the second action on the probability that the situation is occurring should be less than if the first action had been performed.

2. It is possible to add some extra "intelligence" that has not been explained in detail before, because we can account for uncertainty in the input variables. What we mean by this is that the values of the airplane variables that are measured by the sensors can not always be expected to be exactly equal to the exact values that are specified in the knowledge base. For example, if the airplane is standing on the ground the altitude of the airplane should be zero, but the sensor that measures the altitude is not placed on ground level, so the altitude of the airplane will never be exactly zero. Another possibility is that a sensor has not been calibrated correctly and has a small deviation. In the causal model we can account for this deviation by assigning a probability to the input variables that state how probable it is that the value from the aircraft sensor corresponds to the value given in the knowledge base.

3. Another aspect of the causal model is that it is possible to adjust the time interval at which the recognition process is repeated. This can be useful in some very stressful situations in which a lot of things can happen at the same time. In those situations it is very important that the system detects new situations very fast and it can only accomplish this if the time between two recognition loops is small. Therefore in some situations the time interval should be smaller than in other situations.

4. The causal model will have the data in the flight plan of the flight as a reference to what situations it might expect during the flight.

5.2.4 Comparing the models

Of the three models that have been described above, model A has already proven to be too simplistic to deliver good results. Therefore we will concentrate in the remainder of this chapter on the probabilistic and the causal models. Although both models will be described in detail, only the probabilistic model will be implemented in the prototype. This is done because the probabilistic model is a little simpler than the causal model. If the performance of the prototype is satisfactory there is no reason to implement the causal model. However if the performance of the probabilistic model is not good enough implementing the causal model might improve the results of the program.

5.3 The Situation Recognizer architecture

The Situation Recognizer will have the architecture that is shown in figure 5.2. Of all the elements in the architecture only the input module and the overall controller will differ significantly in the probabilistic and the causal models.

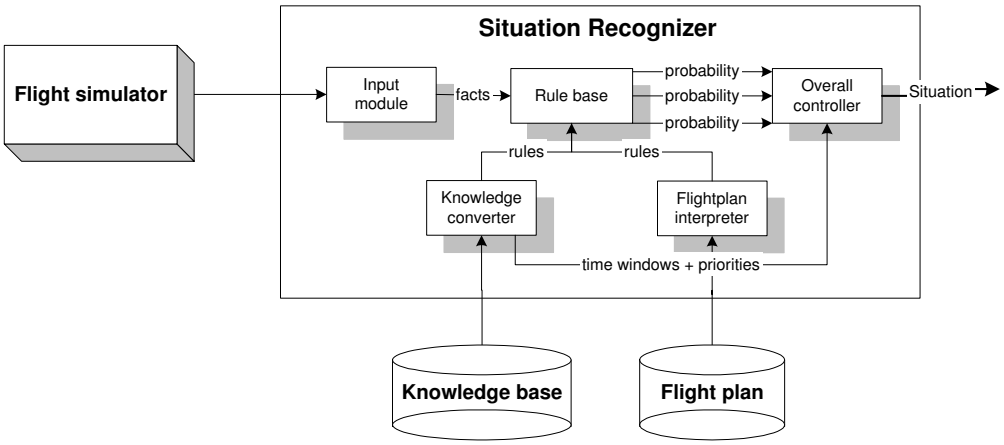

Figure 5.2: The architecture of the Situation Recognizer.

The components of the architecture will be described in the following sections.

5.3.1 The input module

The input module will convert the information from the flight simulator into a form that is compatible with the knowledge in the knowledge base and that the rule base can handle. The exact form will depend on the language that is used to implement the rule base, but in general the information will be stored in the rule base as a number of facts. For example when the pilot has raised the landing gear a fact will be added to the rule base that the landing gear has been raised.

In the causal model we have to account for uncertainty in the input data. How that uncertainty is handled in the causal model is explained in the following text.

The uncertainty of the input

The mapping from the information from the flight simulator to the form that is compatible with the information in the knowledge base might introduce some uncertainty that we have to account for. For example, because the sensor that measures the altitude is not placed on the ground the altitude from the ground will never be exactly zero even if the airplane is standing on the ground. This means that we will need a way to determine our measure of belief that the pilot has performed an action or that a condition has been satisfied given a value from the flight simulator and a value from the knowledge base. How this measure of belief is calculated is described in the following section.

After considering several possible techniques we have chosen to model the uncertainty of the input with Fuzzy Logic, because this enables us to easily map the numerical input values to the values in the knowledge base, from which some look a lot like fuzzy values. For example the values of the variable throttle, which can be IDLE or MIL. We will have to make a distinction between the following cases:

- The value of the variable is one of a limited set of possible values. An example is the value of the landing lights variable. This can be either "ON" or "OFF".
- The value of the variable is a crisp numerical value. For example the value of the variable altitude, this can be any positive number. In this case we will define a fuzzy function that provides a window around the value in the knowledge base. The fuzzy function will determine the measure of belief with which the value from the flight simulator corresponds to the value in the knowledge base. An example of such a fuzzy function for a value of 1000 for the variable altitude in the knowledge base is given in figure 5.3.

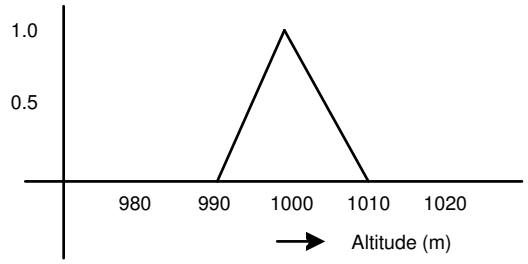

Figure 5.3: A fuzzy function that defines a window around a value in the knowledge base.

- The value of the variable is numerical but specified as a string in the knowledge base. An example of such a value is the value of the throttle variable. For this variable the values "IDLE", "MIL" and "MAX_AB" have been defined in the knowledge base. These values can be directly mapped to a fuzzy set. What those fuzzy sets look like will not be described here, because they too have to be determined by trial and error.

Because there are a lot of possibilities when it comes to determining the shape of the fuzzy functions and it is time consuming to determine what the best shape is we will use a symmetrical triangular shape for all fuzzy functions. The best shape should be determined by experimentation.

5.3.2 The knowledge converter

The knowledge converter will read the knowledge in the knowledge base and convert that knowledge to a set of if-then rules. These rules will be stored in the rule base. The knowledge converter will also pass some of the information in the knowledge base to the overall controller. This information concerns the time windows of the situations and the priority values of the actions.

5.3.3 The flight plan interpreter

The flight plan interpreter will look at the flight plan that has been defined and that will be flown during the flight and creates another set of rules from the flight plan. These rules try to predict what the next situation will be. They are also stored in the rule base.

5.3.4 The rule base

A big part of the reasoning process will be done in a rule base with if-then rules. These rules will be created from the information in the knowledge base and the flight plan. The rules will say something about the probability that a situation is or is not occurring depending on the information from the flight simulator. This information is added to the rule base by the input module. After the information has been added, some of the rules will fire. If a rule fires the result of the rule will be a probability that one of the situations is or is not occurring. This probability will be passed to the overall controller. The following rules will be stored in the rule base:

Start constraint rules: The start constraint rules will fire if and only if all of the start conditions for a situation have been satisfied. If a start constraint rule fires it will produce as output the start constraint probability that is given in the knowledge base and the fact that the start conditions have been met will be stored so that the start conditions will not have to be checked again in the next iteration.

End constraint rules: The end constraint rules are comparable to the start constraint rules, but they fire only if the end conditions have been met.

Action rules: The action rules are rules that fire when an action has been performed. When they fire they produce as output the probability of that action that is stored in the knowledge base. When looking at the actions we have to take into account that the pilot might perform an action, for example by pushing a button, and undo this action, by resetting the button, later on in the situation. This would mean that the rule would not fire if the button is reset and the program would think the action has not been performed yet which would be a false assumption. Therefore when an action rule fires a fact will be added to the rule base stating that the action has been performed.

The actions of the pilot should only be monitored after the start conditions have been met. This means that action rules can only fire after a fact has been added to the knowledge base that the start conditions have been satisfied.

Additional rules: These rules are rules that check whether a certain instrument or variable has a certain value. If that is the case the rule fires and has as output the probability that belongs to the additional rule that is stored in the knowledge base. These are the rules that watch for important events during a situation.

The rule base has a kind of memory. It remembers whether or not the start or end conditions have been satisfied and it remembers that actions have been performed. This memory should be cleared from time to time, because the system should not remember for ever that an action has been performed. For this reason the facts that are added to the rule base for a certain situation get a time stamp so that it is known at which time they have been added to the rule base. Once the time window of the situation the fact belongs to has passed the fact will be removed from the knowledge base. If an action is performed multiple times the time stamp will be refreshed every time the action is performed.

How the rule base is implemented is described in chapter 7.

5.3.5 The overall controller

The overall controller is the part of the Situation Recognizer where most of the reasoning takes place. It takes all probability values it receives from the rules that have fired concerning the situations and combines them. After the probabilities have been combined there will be two probabilities per situation. One that states the likeliness that the situation is occurring and one that states the likeliness that the situation is not occurring. This last probability will be 0 for all situations except the current situation, because that is the only situation for which it is interesting to know if it has ended.

The overall controller determines from those resulting probabilities which situation is most likely the current one. This process is visualized in figure 5.4.

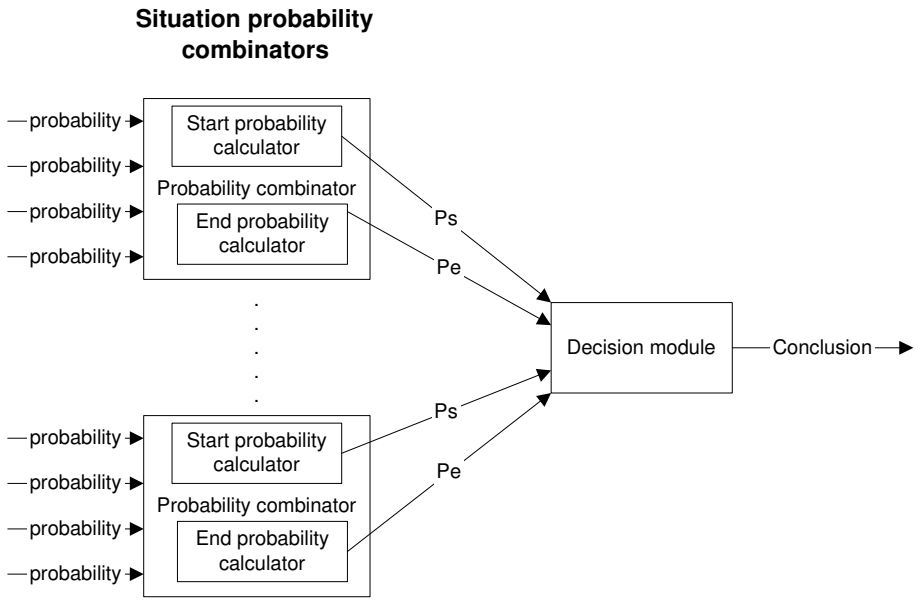

Figure 5.4: The reasoning process of the overall controller.

In the following sections we will describe the elements of the overall controller.

The situation probability combinators

As can be seen from figure 5.4 the situation probability combinators consist of two parts, a start probability calculator that calculates the probability that the situation is the current one and an end probability calculator that calculates the probability that the situation is not occurring. To get an idea of the probabilities that will be combined in the calculators a more detailed picture of a situation probability combinator can be found in figure 5.5.

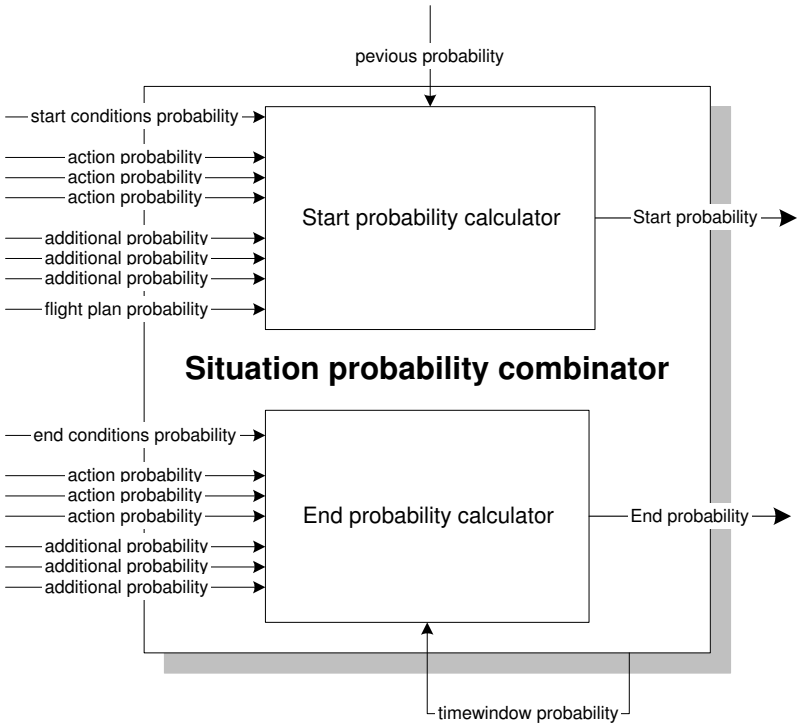

Figure 5.5: A situation probability combinator.

The probabilities that are shown in this figure have the following meaning:

Start conditions probability: If the start constraint rule for this situation fires this probability will be the probability for the start constraints that is given in the knowledge base, otherwise this probability is zero.

Action probabilities: These probabilities are the result of the action rules that fire.

Additional probabilities: These probabilities are the result of the additional rules that fire.

End conditions probability: If the end constraint rule for this situation fires this probability will be the probability for the end constraints that is given in the knowledge base, otherwise this probability is zero.

Flight plan probability: This probability is used in the causal model only. It is based on the distance of the airplane from a steerpoint where, according to the flight

plan, the situation should occur. The closer the airplane gets to the steerpoint, the higher this probability will be.

Previous probability: This is the highest end probability of the situations that could have occurred before this situation. This probability is important because the probability that a situation has started should rise when the probability that one of the previous situations has ended becomes higher. This probability is retrieved from the decision module and is the highest end probability of the previous situations in the previous iteration.

Time window probability: This is the probability that the situation has ended based on the part of the time window that has lapsed since the start of the situation. Because the time window is a maximum duration for a situation the probability that the situation has ended should rise when time progresses. The time window probability is calculated by the situation probability combinator itself based on the time window it gets from the knowledge base and the time at which the situation became the current one. The combinator is notified by the decision module when this happens. If the situation is not the current one the time window probability will be zero.

The algorithm that will be used for the combination of the probabilities is very important because it determines for a big part the accurateness of our program. The algorithm that we have chosen is described in more detail in section 5.4.

The decision module

After all the probabilities have been combined there will be one start probability and one end probability for each situation. These probabilities will be passed to a decision module that will then decide what situation is actually occurring. The decision process of this decision module has been visualized in a Flow Chart in figure 5.6.

In this figure Ps is the start probability of the situation, Pec is the end probability of the current situation, Ts is a threshold value for the start probability (this is explained later in this section), Psc is the start probability of the current situation, Pec is the end probability of the current situation and Te is a threshold value for the end probability of the current situation (this is also explained later in this section). First of all the overall controller will determine which of the situations has the highest start probability. If this situation is the current situation the situation with the next to highest start probability will be taken. This situation will become the current situation in the following cases:

- The end probability of the current situation is 1 and there is another situation with a start probability higher than 0. This means that the recognizer is sure that the current situation has ended. It will then make the situation with the highest start probability the current one, given that there is a situation with a start probability higher than 0. If there is no situation with a start probability higher than zero, the current situation will not change.

- The start probability of the new situation is high enough, this means that it must be higher than some threshold and higher that the start probability of the current situation. The exact value of the threshold will have to be discovered by experimentation.

- The start probability of the new situation is higher than the threshold value and the end probability of the current situation is very high. How high the end probability

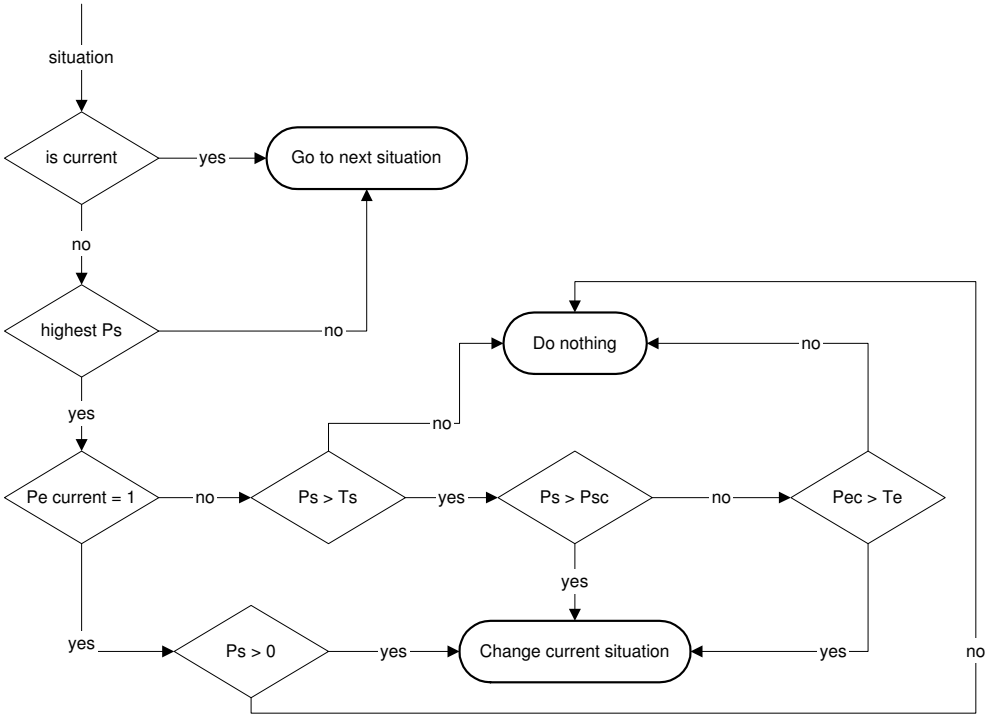

Figure 5.6: The decision process of the decision module.

must be will be determined with another threshold, the value of which will also be determined by experimentation.

If none of these conditions apply the current situation will not change.

5.4 Calculating the probabilities for the situations

From the techniques that were described in chapter 3 we will have to choose those techniques with which we can satisfy the required functionality. The requirements on the functionality of the system have already been outlined (section 4.3). In section 5.4.1 we will choose the technique we will use and explain why we made that choice. Then we will describe how those technique will be implemented in the probabilistic model in section 5.4.2. Finally we will explain how the chosen technique can be used to implement the causal model in section 5.4.3.

5.4.1 Choosing the inference method

From the techniques that have been described in chapter 3 we can choose between the certainty factor model, the Markov model, Bayesian belief networks or the Dempster-Shafer theory.

For combining the probabilities we discard the certainty factor model, because it has gotten so much criticism over the years that we do not have much confidence in the model.

If we would want to use Markov models, we would need information about states and state transitions. As explained in section 2.3 the transitions between one situation to another are very vague and there is no detailed information about the probabilities of these transitions. Although it might be possible to gather this kind of information and use Markov models for the reasoning process, this is not the kind of knowledge that has been stored in the knowledge base. Therefore we will not use Markov models for our reasoning process either.

That leaves the Bayesian belief networks and the Dempster-Shafer theory. As said before the Dempster-Shafer theory is a generalization of Bayesian belief networks. The Dempster-Shafer model is reduced to the Bayesian model when for every subset on a space Y the probability measure is known [29]. This means that the Dempster-Shafer model is more flexible because we do not need to know the probability assignments for all facts. But because we have already defined all probability values in the knowledge base and we will calculate the probabilities of the evidence by using Fuzzy Logic we can just as well use Bayesian networks. This is a proven model which is quite intuitive and does not involve very complex combination algorithms. Furthermore we expect the time needed for the inference in the belief networks will be acceptable because we can keep our networks relatively simple, with probably only two or three layers and no cycles.

Therefore the system will be implemented using Bayesian belief networks.

5.4.2 The probabilistic reasoning process

For every situation two probabilities must be calculated. The probability that the situation is occurring and the probability that the situation is not occurring. Therefore we will need two Bayesian belief networks for every situation. The belief networks for the probabilistic model are described in the following sections.

The start probability calculator

We will first show the complete Bayesian network for the start probability calculator and explain the elements of the network subsequently. The Bayesian belief network that is used to come to a conclusion about the start of a situation is visualized in figure 5.7.

We will now give a description of the elements in the belief network:

The start conditions: The start conditions for a situation are conditions that must be satisfied before a situation can possibly have started. Therefore it would be logical to start verifying these conditions, because if these conditions have not been met then we can be sure that the situation has not started yet, except when an additional rule fires that states the probability that the situation has started is equal to 1. When that happens the additional rule will have precedence and the situation will be considered to have started. When the start conditions are met the probability of the start constraints that is specified in the knowledge base will be the output of this node.

The actions: The situation probability combinator receives a lot of action probabilities from the rule base. These probabilities all contribute to the probability that the situation is occurring and will thus be passed to the situation started node.

The additional rules: The additional rules can contain important information. Every additional rule has its own node in the belief network. When an additional rule for the start of a situation fires the probability belonging to the rule will be passed

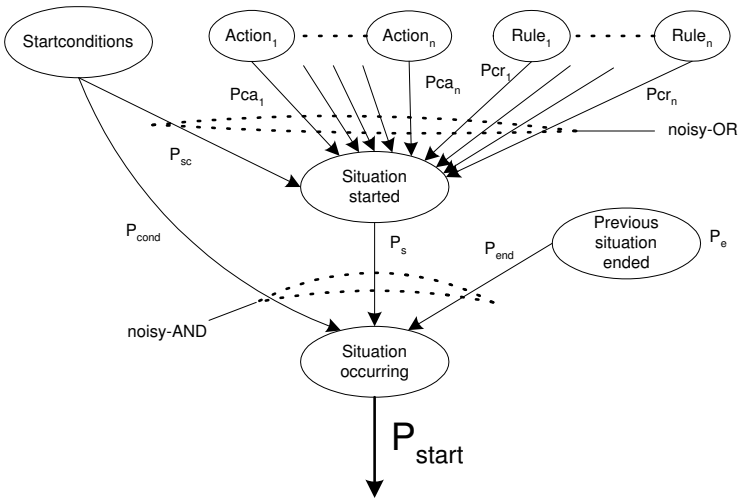

Figure 5.7: The Bayesian belief network used to detect the start of a situation.

to the node and will become the output of the node, otherwise the output will be zero. The additional rules have one special characteristic and that is that if one of the additional rules fires with a resulting probability of 1, this means that it is certain, no matter what, that the situation has started. Thus if this happens, no more calculations are made and the start probability will be set to 1.

Situation started: The probabilities of the nodes that have been described above are all indications that the situation has started. They are combined using the noisy-OR model. The result of this noisy-OR calculation is the total probability that the situation has started and is the output of this node. The noisy-OR model is explained in section 3.2.1.

Previous situation ended: The probability that one of the previous situations has finished is received from the decision module. This probability will be the input of this node. We could just let that end probability be the output of the node, however this would result in the start probability of a situation being zero whenever none of the previous situations is the current one. This is not the desired behaviour, because according to the system requirements (see section 4.3) we want the system to be able to correct a mistake if it has made one, for example by failing to detect a situation. We incorporate this behaviour by setting a minimum value to the output of this node. Thus a situation can have a start probability even if none of its previous situations is the current one. What the best minimum value is will have to be determined by experimentation.

Furthermore we do not want the start probability of a situation to be influenced at all if the probability that one of the previous situations has ended is very high. Therefore the output of the node will be 1 if the end probability is higher than some threshold. The best value for this threshold will also have to be determined by experimentation.

Situation occurring: The resulting probability is the probability that comes from the situation started node after it has been corrected for the end probability of the

previous situation. This correction is necessary because the probability that a situation is occurring does not only depend on the probability that it has started, but also on the probability that the previous situation has ended. This correction is implemented by using the noisy-AND model, described in section 3.2.1. Furthermore the start probability should be 0 as long as the start conditions have not been satisfied yet. This is indicated by the arrow from the start conditions node to the start probability node.

From this Bayesian belief network we can deduce the following formula to calculate the start probability in the probabilistic model:

$$P_{start} = P_{cond} * P_{end} * P_s$$

$$P_{cond} = \begin{cases} 0 & \text{if } P_{sc} = 0 \\ 1 & \text{if } P_{sc} > 0 \end{cases}$$

$$P_{end} = \begin{cases} 1 & \text{if } P_e > T_e \\ P_e & \text{if } P_e <= T_e \end{cases}$$

$$P_s = 1 - ((1 - P_{sc}) * \prod_i (1 - Pca_i) * \prod_j (1 - Pcr_j))$$

The end probability calculator

The belief network that is used to come to a conclusion about the end of a situation is visualized in figure 5.8.

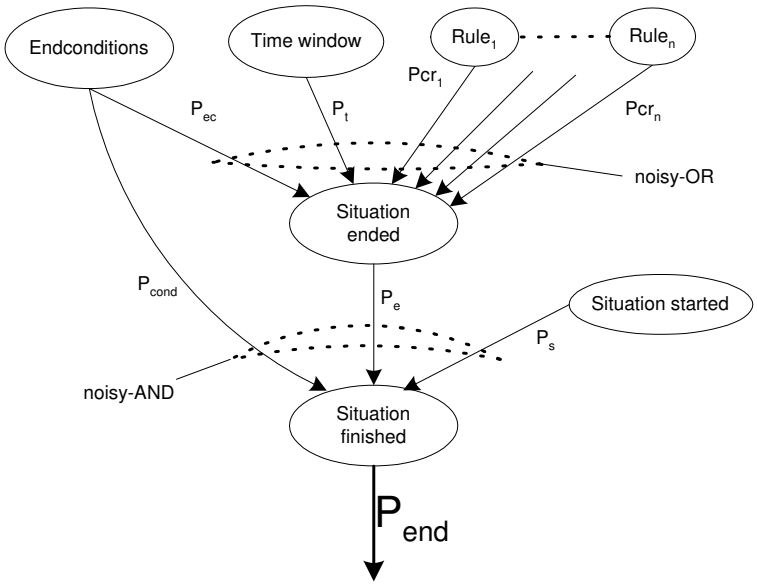

Figure 5.8: The belief network used to detect the end of a situation.

The elements of the network are explained in more detail in the following sections.

The end conditions: The end conditions are conditions that have to be satisfied at the end of the situation. So if the end conditions are not all satisfied yet the program

does not have to look any further to the actions or the time window. If the end conditions are satisfied the end probability defined in the knowledge base is passed to the situation started node and P_{cond} will be 1. If not P_{cond} will be 0 and the resulting end probability will also be 0.

The time window: How the time window probability is calculated has already been discussed in the previous section. It is the time that has lapsed since the start of the situation has been detected divided by the time window.

The additional rules: The additional rules nodes for the end of the situation function the same as the additional rules nodes for the start of the situation. When an additional rule for the end of a situation fires the probability belonging to the rule will be the output of the node, otherwise the output will be zero. The additional rules also have the special characteristic that if one of the additional rules fires with a resulting probability of 1 the end probability will be set to 1 immediately and no more calculations are performed.

Situation ended: The situation ended node combines the probabilities from the end conditions, the actions, the time window and the additional rules to calculate the probability that the situation has ended. The combination is accomplished using the noisy-OR model.

Situation started: Of course the situation can only be finished if it has started somewhere in the past, which means that the situation must be the current one. Therefore the output of this node will be 1 if the situation is the current one and 0 if it is not.

Situation finished: The result of the situation ended node is combined with the output of the situation started node and the end conditions node by using the noisy-AND model. This is done because a situation is only finished if the result from the situation ended node is high enough AND the situation is the current one AND the end conditions have been satisfied. As soon as one of these inputs is 0 the output of this node will become zero and no more calculations will be performed.

From this Bayesian belief network we can deduce the following formula to calculate the end probability in the probabilistic model:

$$P_{end} = P_{cond} * P_s * P_e$$

$$P_{cond} = \begin{cases} 0 & \text{if } P_{ec} = 0 \\ 1 & \text{if } P_{ec} > 0 \end{cases}$$

$$P_e = 1 - ((1 - P_{ec}) * (1 - P_t) * \prod_i (1 - Pcr_i))$$

5.4.3 The causal reasoning process

The probabilistic reasoning model that was described in the previous section uses only a limited amount of the information in the knowledge base. The causal model uses more knowledge from the knowledge base and is a little more "intelligent". In the causal reasoning process we will incorporate the following extra information:

- The uncertainty about the input will be taken into account.
- The chronological order of the actions will be taken into account.

- The priorities of the actions from the knowledge base will be taken into consideration.
- The information that is present in the flight plan will be incorporated in the reasoning process.

In the following sections the changes to the reasoning process will be explained in detail.

The start probability calculator

The complete Bayesian belief network for the start of a situation is shown in figure 5.9.

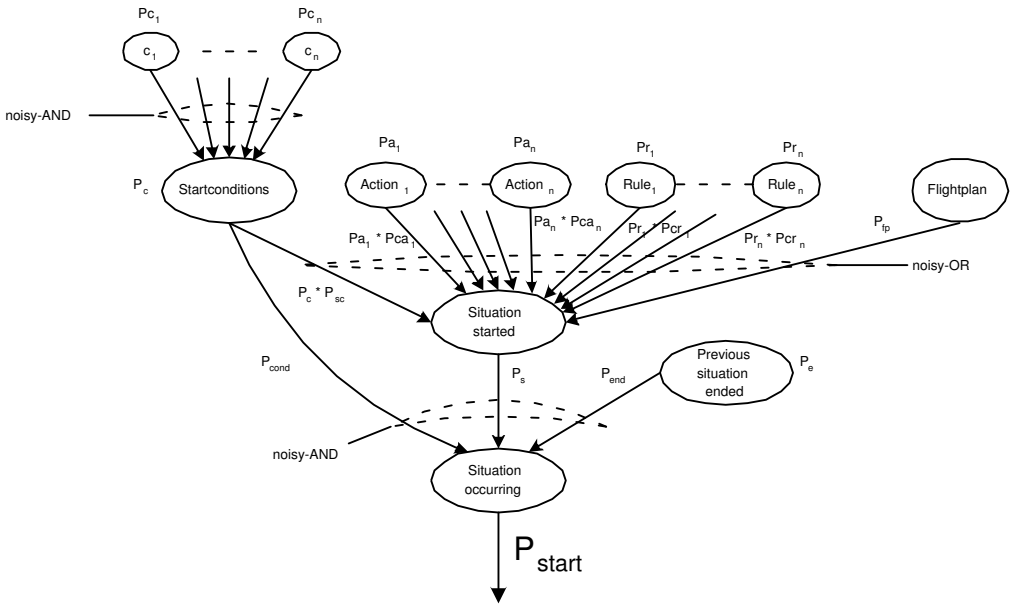

Figure 5.9: The total Bayesian network for calculating the probability that the situation has started.

We will discuss the elements of the network individually.

The start conditions: The knowledge base contains a probability value for the start of a situation when all the start conditions are satisfied. We have to adjust this probability for the uncertainty of the input data. After evaluation of the input data every start condition will have a probability value ranging from zero to one that the condition has actually been satisfied. These probabilities will be combined using the noisy-AND inference method described in section 3.2.1. The resulting probability will be combined with the probability of the knowledge base to calculate the probability that the situation has started based on the starting conditions. The part of the network that will take care of these calculations is shown in figure 5.10.

The actions: When looking at the actions we have to take several things into account that we did not reckon with in the reasoning process for our first prototype:

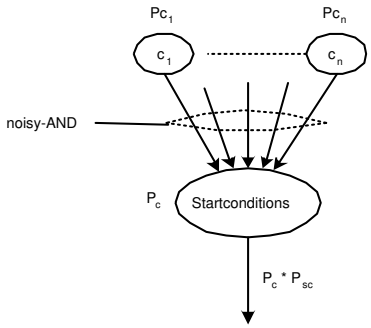

Figure 5.10: The part of the Bayesian network that calculates the probability that the start conditions have been satisfied.

- Some actions are not obligatory. When an action has a priority value of less than 0.5 the pilot is not obligated to perform the action.
- Some actions are time dependent. This means that the actions should be performed in the given order. Our program will use this time dependency by checking if the actions have been performed in the order which is given in the knowledge base.

The extended reasoning process works as follows. Every action has its own node in the belief network. The program iterates over these action nodes and when the action is not yet in the list of performed actions, it checks if the action has been performed by checking if the variable belonging to the action has the value that is specified in the knowledge base. If so the action is added to the list of performed actions. If after this the action is in the list of performed actions the output of the node is set to the probability belonging to that action. If the program encounters a time dependent action that is obligatory but which has not been performed yet the probability values of all subsequent time dependent actions will be multiplied by a value between 0 and 1.

This process has the effect that the time constraint is not very strict, because if the pilot performs two actions in reverse order they will still both be in the list of performed actions and the program will act like they have been performed in the correct order. This is actually a good point because a pilot can not be expected to do everything by the book. Furthermore the possibility that the pilot might forget to perform an action is also taken into account (one of the system requirements defined in section 4.3) because this will not cause the following actions to be completely ignored. They will still contribute to the probability that the situation is occurring, their contribution will just be a little less. The more actions the pilot forgets the less influence subsequent actions have. This method results in a very intuitive reasoning process.

Furthermore we will have to compensate for the uncertainty of the input data. To do this we will have to multiply each action probability by the probability that the input data from which the fact that the action has been performed was deduced is correct.

Finally the probabilities of all individual actions will be combined with the noisy-

OR method to get the total action probability. This results in the part of the network visualized in figure 5.11.

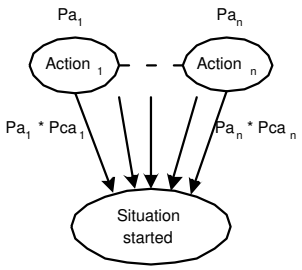

Figure 5.11: The part of the Bayesian network that calculates the actions probability.

The flight plan: As mentioned before the flight plan can be an important aid in detecting the current situation. The proposed method for using the flight plan is to map the distance of the airplane to the next steerpoint to a probability that the situations belonging to that steerpoint are being performed. This probability should rise when the distance to the steerpoint becomes smaller. This way the program will focus more and more on the situations that will arise near a steerpoint when the airplane approaches the steerpoint. When the pilot has arrived at the steerpoint he should either select a new steerpoint himself or the program should automatically select the next steerpoint when all the situations belonging to that steerpoint have occurred.

The additional rules: The only difference between the nodes for the additional rules in the probabilistic reasoning mechanism is that we have to account for the uncertainty in the input data. This means that the probability of a rule is multiplied by the probability that the input data is correct. The combination of the rule probabilities is done in the same way as described for the probabilistic reasoning process.

Situation started: This node combines all probabilities it gets from the nodes that were described above to calculate the probability that the situation has started. The probabilities are combined with the noisy-OR model.

Previous situation ended: This node is the same as the node in the probabilistic model.

Situation occurring: In the causal model the probability that a situation has started has to be corrected for the probability that the previous situation has ended in the same way as in the probabilistic model.

Given this Bayesian belief network we can use the following formula to calculate the

probability that a situation has started in the causal model:

$$P_{start} = P_{cond} * P_{end} * P_s$$

$$P_{cond} = \begin{cases} 0 & \text{if } P_c = 0 \\ 1 & \text{if } P_c > 0 \end{cases}$$

$$P_{end} = \begin{cases} 1 & \text{if } P_e > T_e \\ P_e & \text{if } P_e <= T_e \end{cases}$$

$$P_c = \prod_i Pc_i$$

$$P_s = 1 - ((1 - P_c * P_{sc}) * (1 - P_{fp}) * \prod_i (1 - Pa_i * Pca_i) * \prod_j (1 - Pr_j * Pcr_j))$$

The end probability calculator

This total Bayesian belief network for calculating the probability that a situation has ended is depicted in figure 5.12.

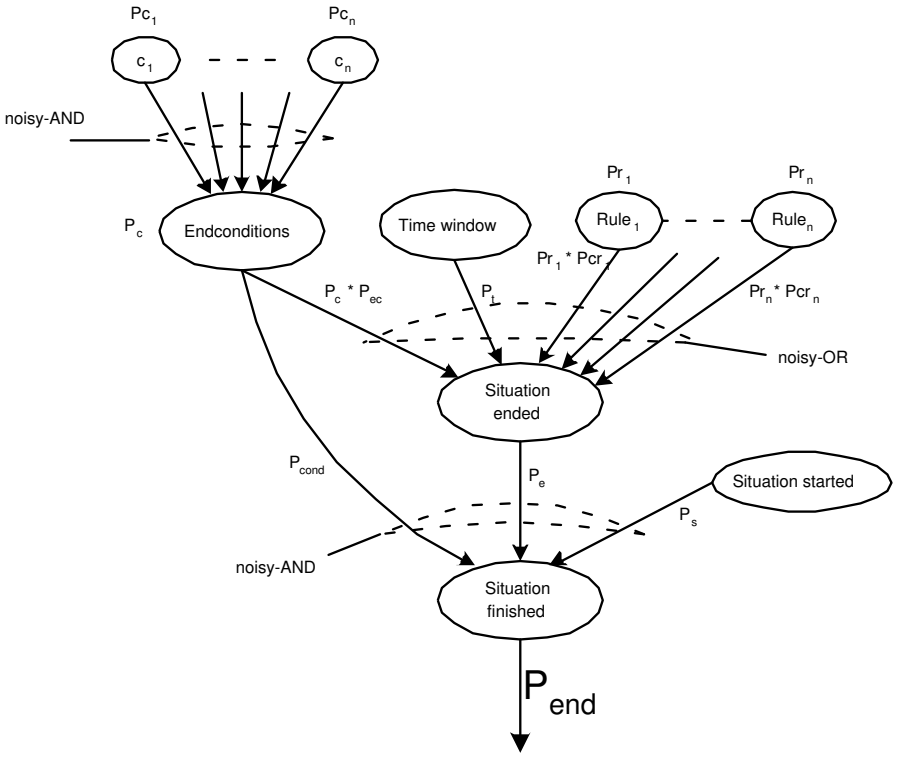

Figure 5.12: The total Bayesian network for calculating the probability that the situation has ended.

The generic structure of the Bayesian belief network for the end probability calculator looks a lot like the generic structure of the Bayesian belief network for the start probability calculator. The part that calculates the contribution of the end conditions is exactly the same as that of the start conditions. We have to combine the probability from this part of the network with the probabilities we get from evaluating the time window and the additional rules that have fired. These probabilities will be combined with the noisy-OR method, because they all contribute to the probability that the situation has ended. Furthermore we have to take into account that the resulting probability will always be zero when the end conditions are not all satisfied and when the situation has not started yet.

This results in the following formula that can be used to calculate the probability that a situation is finished in the causal model:

$$P_{end} = P_{cond} * P_s * P_e$$

$$P_{cond} = \begin{cases} 0 & \text{if } P_c = 0 \\ 1 & \text{if } P_c > 0 \end{cases}$$

$$P_c = \prod_i Pc_i$$

$$P_e = 1 - ((1 - P_c * P_{ec}) * (1 - P_t) * \prod_i (1 - Pr_i * Pcr_i))$$

Part II

The implementation

Chapter 6

The UML model of the situation recognizer

6.1 Introduction

Now that the reasoning process has been described we have to implement the system. Before we start implementing we will create a design of the software program using the Unified Modelling Language, also known as UML. We have chosen for this design methodology because it is widely used to design Object Oriented software systems and because it is possible to design software at different levels of abstraction. The only disadvantage of the methodology is that it has the danger that one goes too far into detail with the design of the software. This is dangerous because the more detail is added to the design the more will have to be changed when the implementation is started. This is because the implementation of a software product will never exactly follow the design, there will always be differences. The more detailed the design the more differences there will be in the implementation and the more time it will take to change the design to make it coherent with the implementation. Therefore the design of the software as it is described in this chapter is not as detailed as is possible with UML, but stays at a level of abstraction that is somewhat higher than the actual implementation.

Furthermore it is important to note that it is not our intention to give a complete specification of our program. The reason we use UML to create a software design is that we want to make sure that our software is reusable by other people. This means that the design must be easy to understand and it must give the reader a general understanding of the structure of our program and the functionality of the different parts of the program.

For those people that have never seen an UML model before, a short introduction to UML is given in the next section. After that the diagrams that have been created as a part of the design will be described in section 6.3.

6.2 Overview of UML

UML provides the means to draw several diagrams that all explain a certain aspect of the software. Because it is not our goal to give a complete specification of the software not all diagrams that can be created with UML will be created for our program. The following diagrams have been created and are described in section 6.3.

Use Case diagram: A use case diagram shows the functionality of the system from the user's point of view. A use case diagram contains actors and use cases. An actor is represented by a stick figure and a use case is represented by an ellipsoid. The actor represents the user, which can be a person or an external process. The use cases represent tasks that can be performed by the actors. An example of a use case diagram is shown in figure 6.1. There is more to use case diagrams than this, but because these extra elements are not used in our design we will not discuss them here.

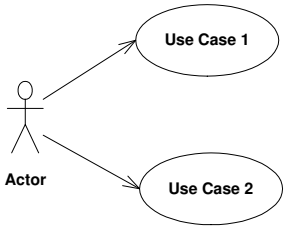

Figure 6.1: An example of a use case diagram.

Class diagram: A class diagram shows the classes that are part of the program and the relationships between those classes. It can show inheritance between classes for example. It can also show the methods and the attributes of the classes. We have split the class diagram up into two parts. In the first part we will only show the classes with their dependencies on other classes. In the second part we will show each class separately with the most important attributes and methods. We only show those methods and attributes that are important to get a clear understanding of the functionality of the class. To clarify the meaning of our classes even more we also create some CRC cards for some of the classes. CRC stands for Class, Responsibilities and Collaborations. In a CRC card the functionality (or responsibility) of a class can be described in natural language and the classes that this class collaborates with to perform its tasks can be specified. CRC cards are not part of UML but provide a nice way to explain the functionality of a class.

In a class diagram an arrow from one class to another means that the class at the end of the arrow is an attribute of the class at the beginning of the arrow and is used by that class. If there is an $*$ at the end of the arrow this means that the class at the beginning of the arrow can contain more than one instance of that class. Inheritance can also be shown in a class diagram. This is shown by an arrow with a closed head from the child class to the parent class. This is illustrated in the figure 6.2, where Child is extended from Parent and uses multiple instances of the Helper class.

Collaboration diagram: Collaboration diagrams are a kind of interaction diagrams. They show the interaction and communication between instances of classes. The instances are represented by a rectangle and the messages that are exchanged between those instances are represented by arrows between those rectangles. Collaboration diagrams provide a good way to get a clear understanding of how the different elements of a program cooperate to provide the required functionality.

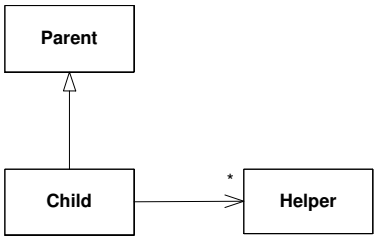

Figure 6.2: An example of a class diagram.

Sequence diagram: A sequence diagram is another kind of interaction diagram. It contains almost the same information as a collaboration diagram and it can be argued that drawing a sequence diagram is not necessary when a collaboration diagram has already been created. However because the information is presented differently in a sequence diagram than in a collaboration diagram it can still be useful to create a sequence diagram. Where a collaboration diagram is very good at showing the relations between the class instances, a sequence diagram concentrates on the time order of the communication messages between the instances.

In a sequence diagram the class instances are represented by rectangles, just like in a collaboration diagram. But where the instances can be placed anywhere in a collaboration diagram in a sequence diagram they are usually placed in a horizontal line. Every instance has a lifeline, which is represented by a vertical line that goes down from the instance rectangle. When one instance calls a method of another instance an arrow is drawn from the lifeline of the calling instance to the lifeline of the called instance. If due to this call the called instance calls a method of another instance (or one of its own methods), another arrow is drawn from lifeline to lifeline below the previous arrow to indicate that this call has occurred later than the previous call. An example of a sequence diagram is shown in figure 6.3.

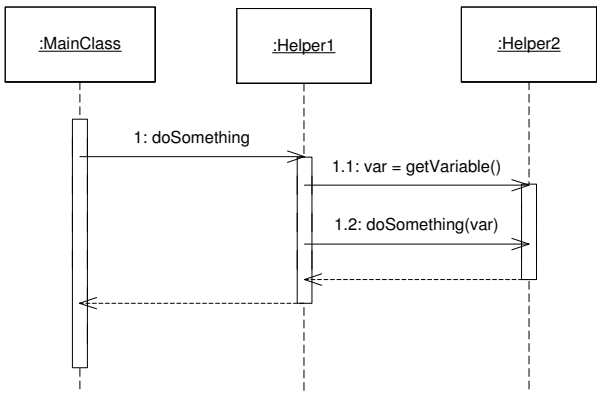

Figure 6.3: An example of a sequence diagram.

6.3 The UML design

In the previous section four useful diagrams for showing the functionality and structure of a software program were explained. In this section the diagrams that were created for the situation recognizer are shown and the way in which those diagrams should be interpreted is explained.

6.3.1 The Use Case diagram

The use case diagram for our program is quite simple because there is only one actor for our system, which is the user. The user can ask the program to perform the following actions:

- load an xml file, in which the knowledge for our reasoning process is stored.
- start the recognition process.
- pause the recognition process.
- stop the recognition process.
- Show/hide a window on which all flight simulator variables and their values are displayed.
- Show/hide a window on which all information in the JESS engine can be displayed.
- Show/hide the table that displays the situation states. If this table is hidden the program will be a lot faster.

This is displayed in figure 6.4.

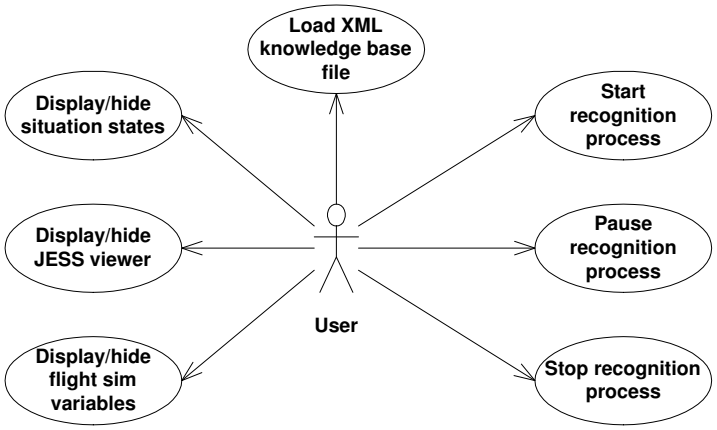

Figure 6.4: The Use Case diagram for the Situation Recognizer program.

6.3.2 The Class diagram

A full class diagram may be difficult to interpret because it gives a lot of information about the contents of the classes from which the functionality of the class is not directly evident. Therefore we created a class diagram with empty classes and show those classes with their content individually after that. Also we have created some CRC cards that describe the responsibilities of the classes in natural language. We will show the class diagram first, then the CRC cards and then the classes with the important methods and variables will be shown.

The class diagram

The full class diagram of the program is shown in figure 6.5. It must be noted that

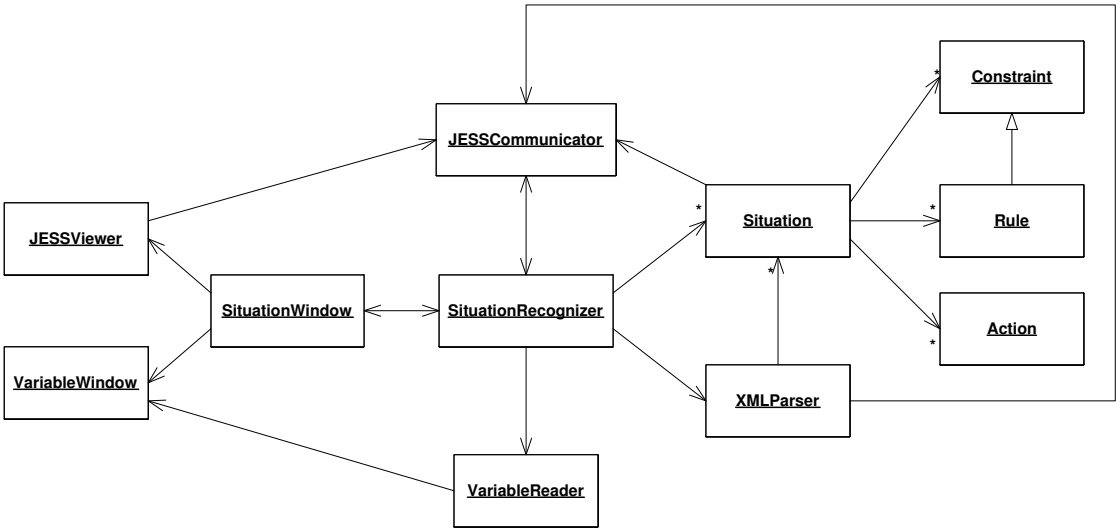

Figure 6.5: The class diagram of the Situation Recognizer program.

not all classes of the program have been shown here. There are some additional helper classes that either contain constant variables that are used in more than one class or that help handle exceptions. These classes are not shown here, because they are not vital to understanding the functionality of the program.

The CRC cards

There is not much to say about the CRC cards, because the nice thing about CRC cards is that they do not need a lot of explaining. The CRC cards are shown in figure 6.6.

The classes

The classes that are shown in the figure 6.5 have the content that is shown in figures 6.7 and 6.8.

SituationRecognizer	
Responsibility	*Collaborators*
This is the main class of the application. Its main responsibility is to run a loop in which it gets the start and end probabilities of the situations, determines the most likely current situation and gives that with the probabilities to the SituationWindow.	SituationWindow XMLParser JESSCommunicator VariableReader VariableWindow JessViewer Situation

JessViewer	
Responsibility	*Collaborators*
The JessViewer can display the information that is stored in the JESS engine. It can either display the facts, the rules or the global variables that are currently in the JESS engine.	JESSCommunicator

SituationWindow	
Responsibility	*Collaborators*
Provides a user interface to the user with which the user can choose the xml file that contains the knowledge base and can start, pause and stop the program. It also shows the situation probabilities and the current situation to the user.	SituationRecognizer VariableWindow JessViewer

Situation	
Responsibility	*Collaborators*
Stores the characteristic constraints and actions from which the situation can be recognized.	JESSCommunicator Constraint Action Rule

XMLParser	
Responsibility	*Collaborators*
The XMLParser parses an XML file that contains the knowledge. It creates Situation classes from this file, which can be retrieved by the SituationRecognizer.	JESSCommunicator Situation Constraint Action

Action	
Responsibility	*Collaborators*
Contains the information of an action tag in the XML file. It contains the name of the flight simulator variable and the value that the variable will have after the action has been performed. It also contains the probability and priority values of the action.	

JESSCommunicator	
Responsibility	*Collaborators*
The JESSCommunicator is a shell around the JESS engine. This class can be used to convert the knowledge to XML rules and load them into the JESS engine.	SituationRecognizer Situation Constraint Action

Constraint	
Responsibility	*Collaborators*
Contains the information of a constraint tag in the XML file. It contains the name of the flight simulator variable and the start and end condition on the value of that variable.	

VariableReader	
Responsibility	*Collaborators*
This class retrieves the variable values from the flight simulator and puts them as facts into the JESS engine. The variables and their values can be displayed on the VariableWindow.	JESSCommunicator VariableWindow

Rule	
Responsibility	*Collaborators*
Contains the information of an additional rule in the XML file. It is an extension of a Constraint, so it contains the same variables plus an additional probability value.	

VariableWindow	
Responsibility	*Collaborators*
The VariableWindow can be used to display the variables and their values. This can be convenient for debugging.	

Figure 6.6: The CRC cards for the classes of the Situation Recognizer program.

6.3.3 The Collaboration diagram

To get a clear understanding of how the program works we need to know how the classes communicate with each other. This can be shown in a collaboration diagram. The structure of such a diagram has been explained in section 6.2. The collaboration diagram of the Situation Recognizer is shown in figure 6.9.

6.3.4 The Sequence diagram

Because the Collaboration diagram is not very good in showing the order of the messages between the classes in time, we have also created some Sequence diagrams. These diagrams are better in showing time relations between the different method calls. The Sequence diagrams are shown in figures 6.10 and 6.11. There are a couple of elements in the second diagram that need explaining. First of all the arrow with only half of a head that goes from the SituationWindow to the SituationRecognizer with the label *start*. This is not a printer error. This arrow indicates an asynchronous procedure call. This means that a new thread is started in which the SituationRecognizer starts reasoning so that the control is given back to the SituationWindow immediately. This enables the SituationWindow to update the user interface while the SituationRecognizer is reasoning.

Furthermore the arrow that goes from the SituationRecognizer to itself with the label *determineCurrentSituation* is not a method call like the other arrows. It is actually a group of instructions that determine which situation is most likely to be occurring at that moment. This is not a standard UML notation, but because it is essential to know that this is done here to understand the functionality of the program I added this arrow. What this group of instructions does exactly is described in section 5.3.5.

Figure 6.7: Some of the classes of the Situation Recognizer program.

Situation
JESSCommunicator jessCommunicator String situationName Vector constraints Vector actions boolean startConditionsSatisfied boolean endConditionsSatisfied double startConstraintsProbability double endConstraintsProbability int nrPerformedActions long timeWindow boolean isCurrent double startProbability double endProbability Vector previousSituations
Situation(String situationName, JESSCommunicator jessComm) addPreviousSituation(String situationName) setState(String state) addPerformedAction() reset() calculateStartProbability(HashMap situations) : double getStartProbability() : double calculateEndProbability() : double getEndProbability() : double setCurrent(boolean current) getSituationName() getConstraints() : Vector setConstraints(Vector constraints) getActions() : Vector setActions(Vector actions) setStartConstraintsProbability(double startConstraintsProbability) setEndConstraintsProbability(double endConstraintsProbability) setTimeWindow(long timeWindow)

Action
String variableName double priority double probability String value
Action(String name, String val, String prob, double priority) getVariableName() : String getValue() : String

Constraint
String variableName String startConstraint String endConstraint
Constraint(String name, String start, String end) getVariableName() : String getStartCondition() : String getEndCondition() : String

Rule
double probability
Rule(String name, String start, String end, double probability) getProbability() : double

Figure 6.8: Some of the classes of the Situation Recognizer program.

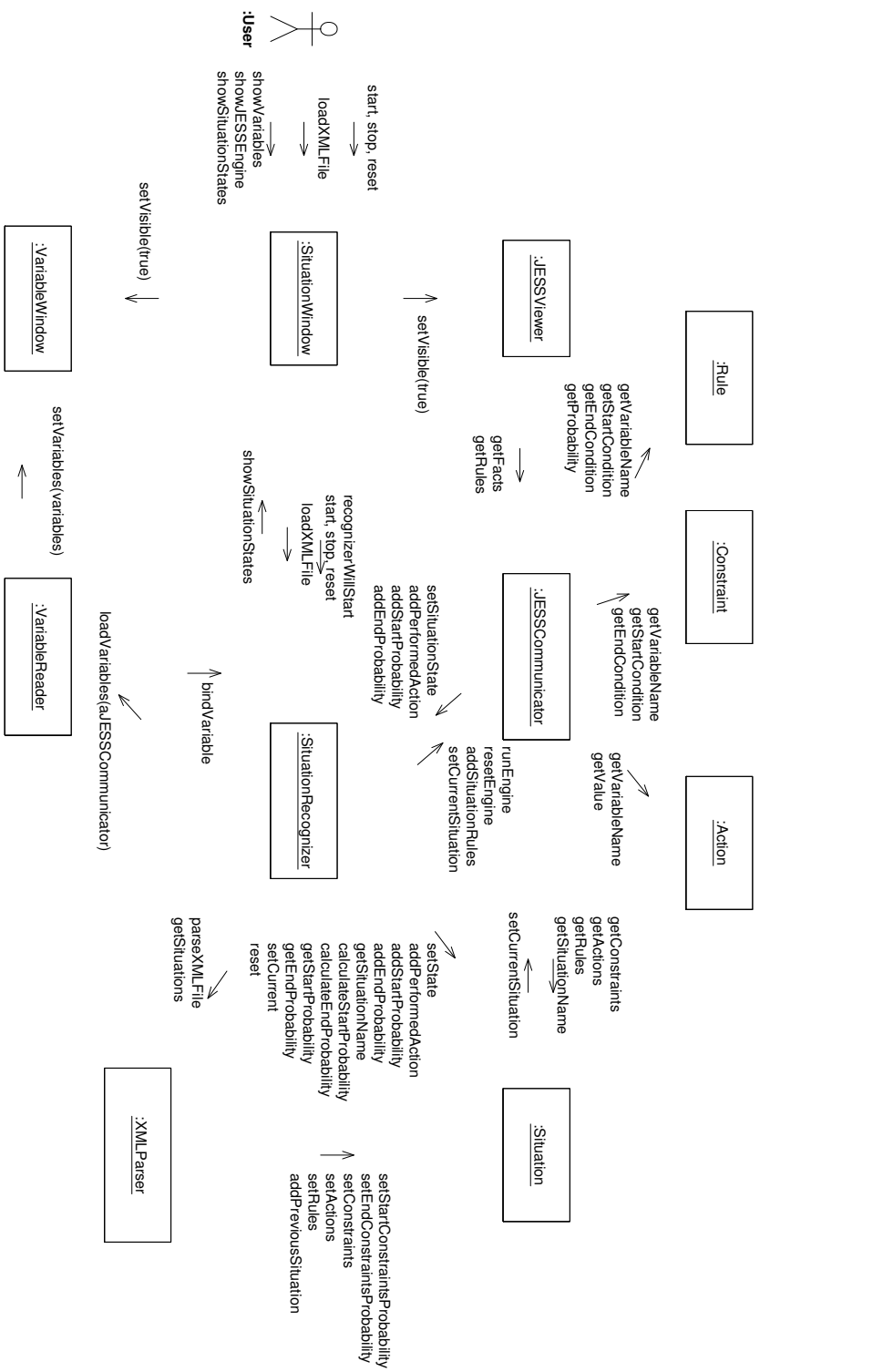

Figure 6.9: The Collaboration diagram of the Situation Recognizer.

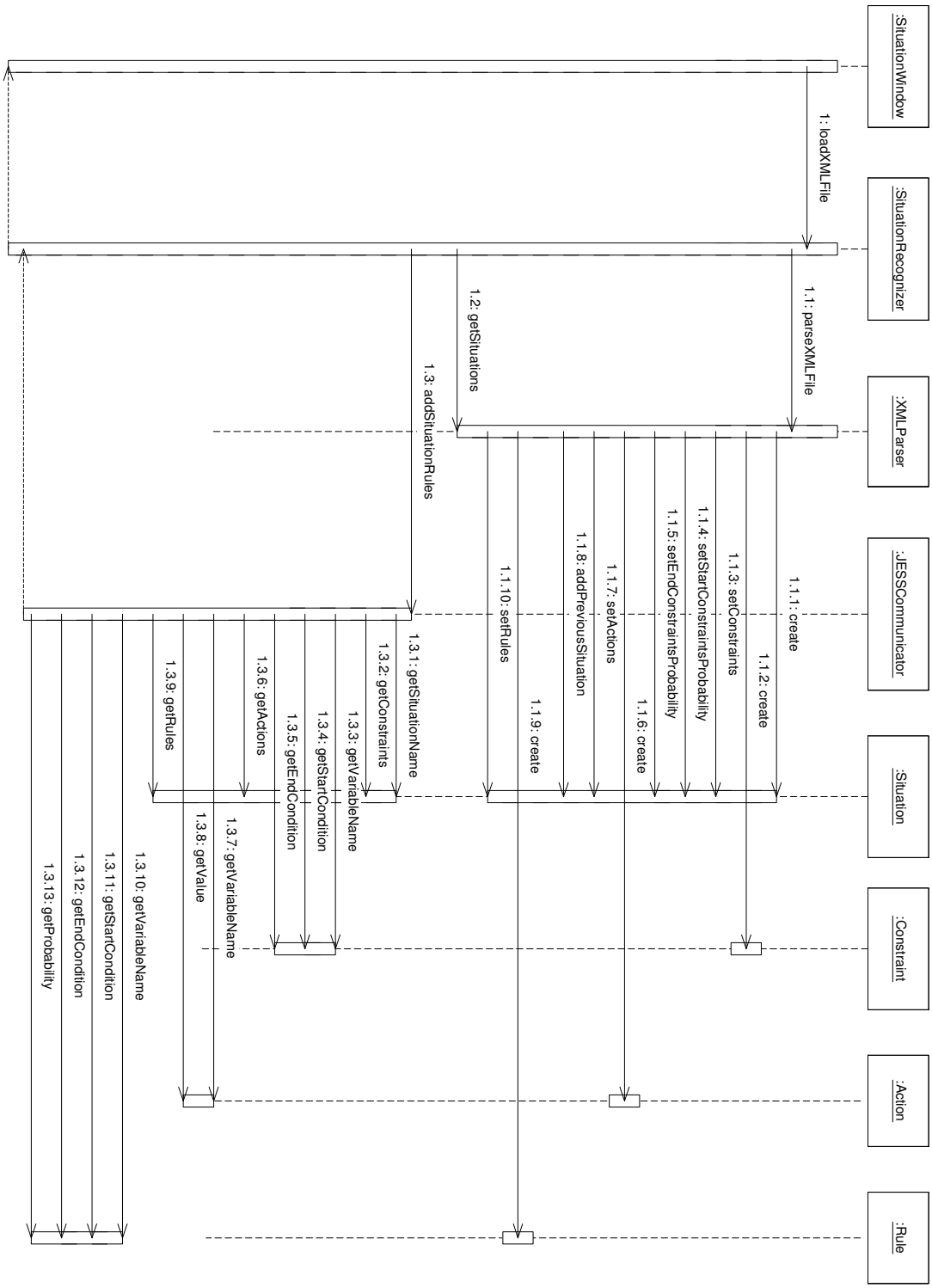

Figure 6.10: A Sequence diagram of the Situation Recognizer.

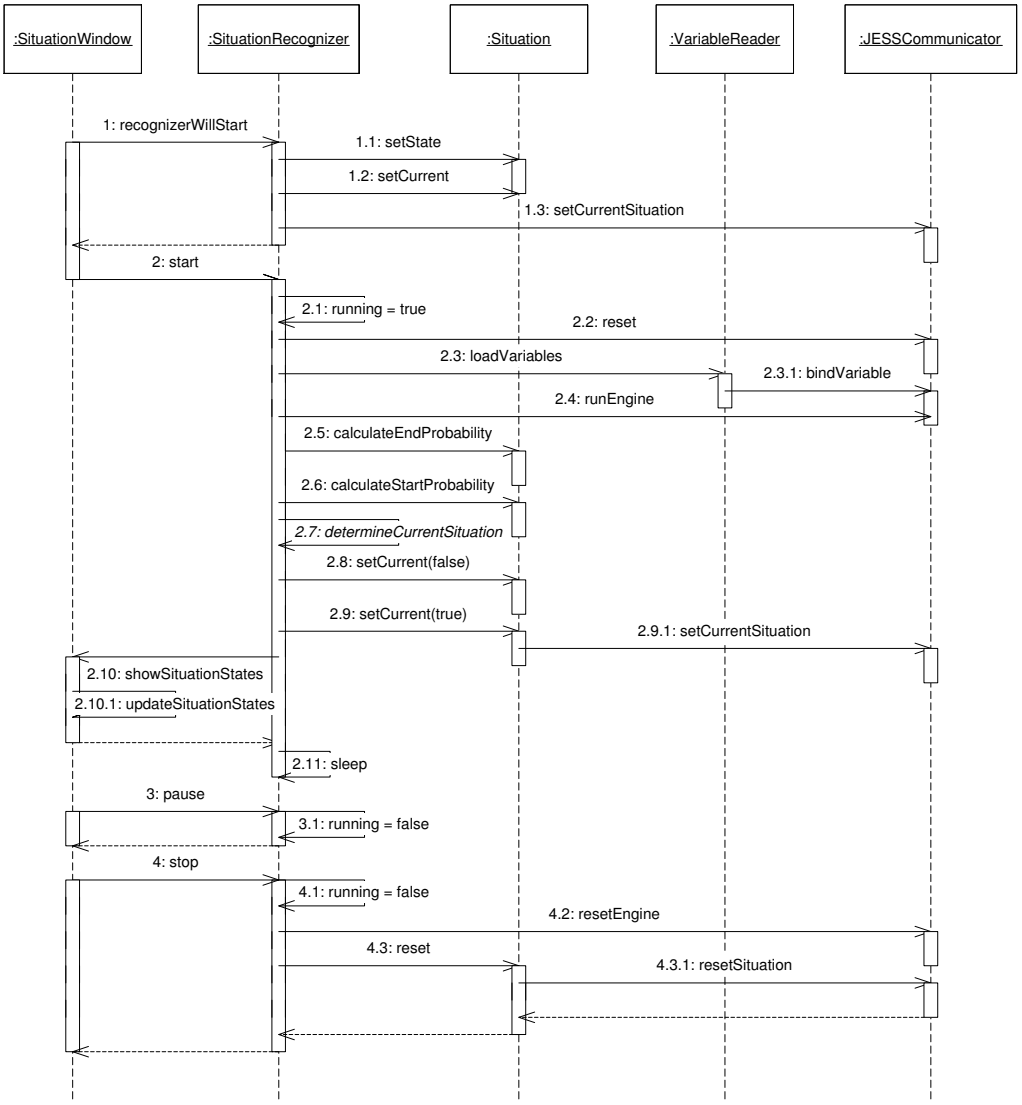

Figure 6.11: A Sequence diagram of the Situation Recognizer.

Chapter 7

Implementing the rule base and the Bayesian belief networks

7.1 Introduction

When we started implementing the prototype two important choices had to be made. First of all, we had to figure out how the rule base would be implemented and secondly we had to choose the tools that we would use to implement the Bayesian belief networks. Because we intended to implement the prototype in JAVA we chose to implement the rule base in JESS (Java Expert System Shell), which is discussed in section 7.1.1.

To implement the Bayesian belief networks we intended to use SMILE (Structural Modelling, Inference and Learning Engine), which is described in section 7.1.2. Unfortunately, due to the limited time that was available, we did not get to implementing the Bayesian belief networks with this tool. The calculations that would have been done in the Bayesian belief networks have instead been programmed directly in the JAVA code. This does not affect the results, but it probably does make the program slower in calculating the probabilities.

In the following sections we will describe JESS and SMILE. Then we will explain how the data from the flight simulator is converted to facts that JESS can handle by the input module. After that we will explain how the knowledge in the knowledge base is converted to JESS rules. Finally we will point out some of the problems that we encountered during the implementation of the prototype.

7.1.1 JESS: The Java Expert System Shell

JESS is a language specifically designed for defining and reasoning with rule bases. The knowledge from the knowledge base will be converted to a set of rules in the JESS language and exported to a JESS engine. Before we describe the format of the rules, we will explain some things about the JESS language.

JESS is written in the JAVA programming language. It is compatible with CLIPS and most JESS scripts are valid CLIPS scripts. JESS is more extensive than CLIPS though, because it is has some other extra functionality [39]. JESS makes it possible to communicate with JAVA objects during execution of a JESS program. Although it

might seem strange to choose JAVA for a real time application because of the known slowness of JAVA compared to languages like C or C++, this is not so strange when one considers that the parts that are written in JAVA are actually quite small. These parts are the part that parses the XML file and the part that provides the user interface. Parsing the XML file is done only once before the reasoning process and thus the time this takes is not so important. The user interface only has to show the conclusions of the reasoning process which is just the name of the current situation and will therefore not be very time consuming either. Because the biggest part of the reasoning process is done by the JESS engine which has been optimised for this kind of reasoning we will still be able to get the program running in real time.

JESS is continuously under development and new versions are released regularly. We used the versions 6.0 at first, which was replaced during this project with version 6.1 and which was used in the eventual prototype.

As can be seen in the picture of the architecture of the program (figure 4.2) there are three modules that add things to the rule base: the input module, the knowledge converter and the flight plan interpreter. Because we have not implemented the flight plan converter yet in the prototype, this input will not be described here. What the input module and the knowledge converter put into the rule base is described in the following sections.

7.1.2 SMILE: Structural Modelling, Inference and Learning Engine

SMILE is a C++ library containing functions that can be used to create and reason with Bayesian belief networks. Some tools have been created to make it easier to use SMILE. These are a graphical user interface that is called GeNIe and an easy to use ActiveX interface called SmileX. SmileX has been described in [30]. To create the Bayesian belief networks we have two options:

1. We can use SmileX and access the ActiveX interface from the JAVA program.
2. We can access the SMILE library directly using JNI (Java Native Interface) from our JAVA program.

Because we lacked the time to implement the Bayesian belief networks in the prototype, we have not made a choice between these two approaches. This should be looked into in following research projects.

7.2 The data from the input module

The input module adds the values of the aircraft variables to the rule base as a set of facts. The way in which the variable values are retrieved from the flight simulator is explained in appendix A. Once the values of the variables are known they have to be put in the rule base so that the rules can reason with them. There are two possible ways to store the variables with their values in the JESS rule base, as global variables or as facts. Because the JESS engine is optimised for reasoning with facts we have chosen to store the variables with their values in the JESS engine as facts. So for a variable var with a value val we will give the following command to the JESS engine:

```
(assert (var val))
```

This command will add the fact `(var val)` to the JESS engine.

7.3 The data from the knowledge converter

To convert the knowledge base to a format that JESS can understand we will have to create some rules with the information in the knowledge base. There are basically three kinds of rules that can be derived from the knowledge (see section 5.3): constraint rules, action rules and additional rules. These rules are described in the following sections.

7.3.1 The constraint rules

There are two kinds of constraint rules per situation: start and end constraint rules. The start constraints specify the values that the flight simulator variables must have at the start of a situation. Therefore the start constraint rules check these variable values and if they satisfy the conditions given in the XML file a message is sent to the Situation Recognizer program to let it know that it is possible that the situation has started. The same goes for the end constraint rules. To optimise the performance of the program the start and end constraint rules are only checked if the start or end conditions have not been satisfied yet. Furthermore the end constraint rules are only checked if the situation is the current one since a situation can only have ended if it has started somewhere in the past. As an example we have taken a start and an end rule of a takeoff. The start rule is shown in figure 7.3.1 and the end rule is shown in figure 7.3.1 In the example some of the commands are explained by comment lines (beginning with a ;).

```
; defines the global variable that is "true" when the start conditions
; have been satisfied, "false" otherwise
(defglobal ?*Taking_off_canStart* = "false")

; the start of the rule
(defrule Taking_off_start
   ; check that there is an instance of the class with which we can
   ; communicate with the Situation Recognizer program.
   (JESSCommunicator (OBJECT ?jessCommunicator))
   ; check that the start conditions have not been satisfied yet.
   (test (<> (str-compare ?*Taking_off_canStart* "true") 0))
      ; check the values of the start constraint variables.
      (not (landing_gear ~LOWERED))
      (not (altitude ?altitude&:(<> ?altitude 0)))
      (not (radio ?radio&:(<> ?radio 7)))
      (not (FCR ~NO_RAD))
   ; if all conditions have been met pass a message to the Situation
   ; Recognizer program that the situation Taking off has started.
    => (call ?jessCommunicator setSituationState "Taking off" "started")
   ; set the global variable to "true" so that the rule is not
   ; evaluated again.
   (bind ?*Taking_off_canStart* "true")
)
```

Figure 7.1: A start rule for the situation Taking off.

These examples need a little more explaining. First of all the variables currentSituation, Taking_off_canStart and Taking_off_canEnd are global variables. Global variables can be defined in JESS with the command `defglobal` and by putting the variable name

```
; defines the global variable that is "true" when the end conditions
; have been satisfied, "false" otherwise
(defglobal ?*Taking_off_canEnd* = "false")

; the start of the rule
(defrule Taking_off_end
   ; check that there is an instance of the class with which we can
   ; communicate with the Situation Recognizer program.
   (JESSCommunicator (OBJECT ?jessCommunicator))
   ; check that the situation is the current one.
   (test (= (str-compare ?*currentSituation* "Taking off") 0))
   ; check that the end  conditions have not been satisfied yet.
   (test (= (str-compare ?*Taking_off_canEnd* "false") 0))
      ; check the values of the end constraint variables.
      (not (ground_speed ?ground_speed&:(<= ?ground_speed 0)))
      (or (not (altitude ?altitude&:(<> ?altitude 0)))
         (not (altitude ?altitude&:(<= ?altitude 100))))
   ; if all conditions have been met pass a message to the Situation
   ; Recognizer program that the situation Taking off has finished.
    => (call ?jessCommunicator setSituationState "Taking off" "finished")
   ; set the global variable to "true" so that the rule is not
   ; evaluated again.
   (bind ?*Taking_off_canEnd* "true")
)
```

Figure 7.2: An end rule for the situation Taking off.

between `?*` and `*`. Global variables will not be removed from the JESS engine when the engine is reset. When global variables are defined they are given a default value (in this case the string value "false"). When the JESS engine is reset the global variables are normally reset to their default values. This is not what we want and this can be prevented by setting the JESS property `set-reset-globals` to `FALSE` or `nil`.

Furthermore the way in which JESS can call methods of object instances in a JAVA program needs to be explained. For this to be possible two things have to be done:

1. The class has to be introduced to JESS by the JAVA program. This can be done by calling the function `defclass` with two arguments: the name that will be used to reference the class in the JESS engine and the classname itself.

2. A pointer to the instance of the class has to be given to the JESS engine. This can be done by calling the JESS function `definstance` with three arguments: the name that was used to define the class, the pointer to the instance and an argument that specifies whether the instance is dynamic or static. The instance is put in the JESS engine as a fact with the following form `(ClassName (OBJECT instancePointer))`. Therefore the instance pointer can be bound to a variable in the way that is shown in the examples.

The JESS function `call` can be used as shown in the examples to call a method of the instance. The first argument of the `call` function is the pointer to the instance, the second one is the name of the method and the remaining arguments will be passed to the method that will be called.

One other interesting thing about the constraint rules is that we want the rule to succeed even if for some of the variables of that constraint there are no facts present in the fact base. Because then we will be able to reason with a limited amount of variables. To do this it is not sufficient to just check for a variable var if the fact `(var val)` exists, because this check would evaluate to false if no fact for the variable var exists. We have to check that there exists no fact in the fact base that states that the value of variable var is not equal to the given value. In other words we need a double negation. To achieve this we can use the statement `(not (var  val))`. This check will be true when the value or variable var equals val AND when there is no fact in the fact base for the variable var.

7.3.2 The action rules

Action rules are rules that check if an action that belongs to a situation has been performed. Because a situation must have started if an action is to be performed an action rule can only fire if the start conditions of a situation have been met. An action rule is only evaluated when the action has not been performed yet. The action rule checks if the action has been performed by checking if the variable that belongs to the action has the value it will have when the action has been performed. If this is the case a message is sent to the Situation Recognizer program that an action of the situation has been performed and the probability of that action is given to the program. We have taken the rule that checks if the landing gear has been raised during a takeoff as an example (see figure 7.3.2).

```
; define the global variable that indicates whether or not the action has
; already been performed.
(defglobal ?*Taking_off_action6* = "unperformed")

; the start of the rule.
(defrule Taking_off_action6_rule
   ; check that there is an instance of the class with which we can
   ; communicate with the Situation Recognizer program.
   (JESSCommunicator (OBJECT ?jessCommunicator))
   ; check that the start conditions of the situation have been satisfied.
   (test (= (str-compare ?*Taking_off_canStart* "true") 0))
   ; check that the action has not been performed yet.
   (test (= (str-compare ?*Taking_off_action6* "unperformed") 0))
   ; check the variable that belongs to the action.
   (landing_gear RAISED)
   ; if all conditions have been satisfied set the global variable so
   ; that the rule will not be evaluated again.
   => (bind ?*Taking_off_action6* "performed")
   ; notify the Situation Recognizer program that an action has been
   ; performed and give it the probability of the action.
   (call ?jessCommunicator addPerformedAction "Taking off" 0.9)
)
```

Figure 7.3: An action rule for the situation Taking off.

7.3.3 Additional rules

The additional rules can either say something about the start or about the end of a situation. When an additional rule fires the probability belonging to that rule is passed to the program and a variable is set to prevent the rule from firing again. As an example we have taken the rule that says that the situation taxiing to runway is certainly finished whenever the altitude of the airplane becomes bigger than 0. This rule is displayed in figure 7.3.3.

```
; define the global variable that can be checked to see if the rule has fired
; already.
(defglobal ?*Taxiing_to_runway_endrule0_fired* = "false")

; the start of the rule
(defrule Taxiing_to_runway_endrule0
   ; check that there is an instance of the class with which we can
   ; communicate with the Situation Recognizer program.
   (JESSCommunicator (OBJECT ?jessCommunicator))
   ; check that the situation is the current one.
   (test (= (str-compare ?*currentSituation* "Taxiing to runway") 0))
   ; check that the rule has not fired yet.
   (test (= (str-compare ?*Taxiing_to_runway_endrule0_fired* "false") 0))
   ; check the variable that belongs to the rule.
   (altitude ?altitude&:(> ?altitude 0))
   ; if all conditions have been satisfied set the global variable so that
   ; the rule will not fire again.
   => (bind ?*Taxiing_to_runway_endrule0_fired* "true")
   ; notify the Situation Recognizer program that the rule has fired and
   ; tell it the probability that belongs to the rule.
   (call ?jessCommunicator addEndProbability "Taxiing to runway" 1.0)
)
```

Figure 7.4: An additional rule for the situation Taxiing to runway.

7.4 Problems encountered during implementation

During implementation JESS turned out to be easy to work with. Thanks to the work of Bart van der Poel, who wrote a program in JESS that could be used as an example, it was not difficult to get to know JESS and to figure out how best to use it to implement our rule base. Furthermore the inference mechanism of JESS turned out to be very fast as expected.

However, we did encounter some problems with JESS during the project. On the JESS website [39] some of the problems that might be encountered are described. One of those problems in version 6.0 of the JESS package was that nested OR statements were evaluated very slowly and only a limited number of nested OR's could be handled by the inference mechanism of JESS. This problem became evident because at first we used a lot of OR statements in the constraint rules to check if the value of a variable matched the value in the knowledge base OR that there was no fact in the database. Because there are some situations with a lot of variables in the constraint rules (the Startup situation

for example) this caused some errors. When we wrote an e-mail describing the problem to the developer of JESS, we received an answer very quickly in which we were advised to rewrite our rules or wait for the next version of JESS (version 6.1) to be released in which the problems with nested OR's would be solved. We have chosen to rewrite the rules in the manner that is described in this chapter.

The test results and some of the problems that were encountered during testing are described in chapter 8.

Part III

Results, conclusions and recommendations

Chapter 8

The results

8.1 Introduction

In this chapter we will describe the results of the tests that were performed with the Situation Recognizer. In this introduction we will show pictures of the use interface of the Situation Recognizer and we will make some remarks about the way in which the tests were performed.

The system was tested by flying a number of scenarios that have been specifically designed to test if the program produces good results and satisfies the requirements that have been defined in section 4.3. The scenarios that were flown are described in section 8.2, after which the test results are given and explained in section 8.3.

8.1.1 The user interface

Figure 8.1 shows the program with all its windows during one of the test flights.

During a flight the program showed the situation that it thought was the current one. It also showed for each situation the probability that that situation had started and was thus the current one and the probability that the situation had ended.

The program offers the possibility to open a window that displays the values of the flight simulator variables (the window on the right). It shows a history of the values that goes four iterations back. Thus the first value after the name of the variable is the oldest value and the value in the fifth column is the most recent one.

The program also has the ability to show a window on which some information about the data in the JESS engine can be shown (the window at the bottom). The window can show the facts, the rules and the global variables that are present in the JESS engine at the moment the corresponding button is clicked.

8.1.2 Further remarks

The flight simulator that was used to fly the scenarios with is the Microsoft Flight Simulator 2002. The program was run under Windows 2000 on a 735 Mhz PC with 256 MB of memory. The program was set to perform five iterations per second. The program showed a message when it did not achieve this rate, which was seldomly.

It must be noted that the test scenarios were not flown by a real pilot. They were flown by somebody with only little experience with the Microsoft Flight Simulator and no experience at all with flying a real airplane. Since the knowledge in the knowledge

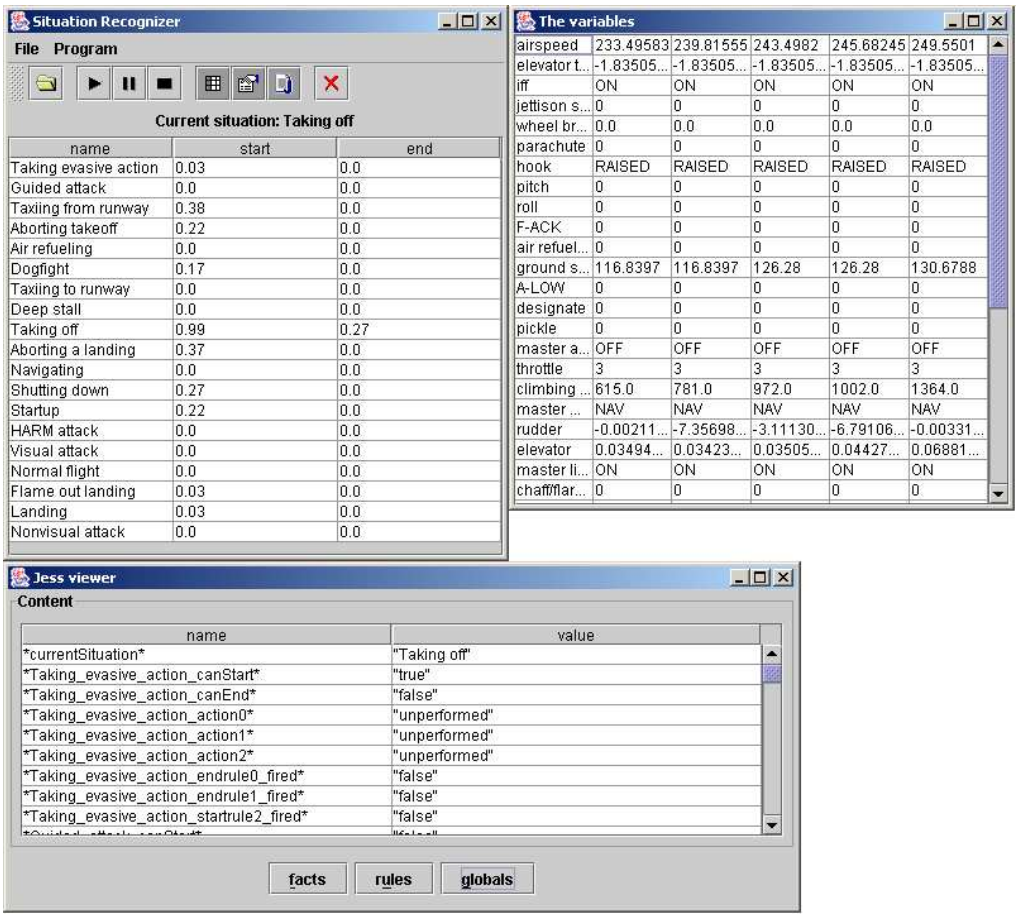

Figure 8.1: The Situation Recognizer program during one of the test flights.

base is knowledge about how a real pilot flies this could cause problems because there are usually differences between the flying behaviour of a real pilot and someone that has never flown before. This became evident during the interview with the former F-16 pilot Gideon Reisel who flew some scenarios with the F-16 flight simulator Falcon 4 during the interview and had significantly less problems with flying and especially landing the airplane than the interviewer himself. However the problems should not be too big since the program has been designed to recognize situations even if the pilot makes mistakes. The fact that the person that flies the airplanes during the test scenarios is not a real pilot should therefore have no significant influence on the results.

Microsoft Flight Simulator 2002 that was used to fly the test scenarios offers the possibility to record a flight and replay it at a later time. This was done for all the test scenarios to be able to test modifications of the program without having to fly the scenarios again. A drawback of this method is that a flight is recorded by saving the airplane state at a predefined interval. The minimum time interval that can be set is 250 ms. This means that a flight is recorded by saving only four frames per second. However after the first tests this turned out not to be a problem since there was no noticeable difference between the results of the program when it was tested on a recorded flight or

on a live flight.

Because of the limited amount of time that was available we have not been able to implement all the features of the probabilistic model that have been described in this report. The prototype has the following limitations:

- The rule base has almost permanent memory. As described in section 5.3.4 the rule base should have a memory that should be cleared regularly. This means that facts that concern a certain situation should be cleared from the rule base after some time. However, in the prototype those facts are only removed from the rule base if the situation they apply to has ended after it has become the current situation. This means that those facts are remembered too long by the rule base.
- To be able to test our program with an F-16 we had to create a cockpit of an F-16 for the Microsoft Flight Simulator 2002. How this cockpit has been created is described in more detail in appendix A. Because we had only a limited amount of time available we have not been able to create a complete F-16 cockpit. Therefore only a limited amount of instruments could be used during the test flights and only a limited amount of the knowledge in the knowledge base could be used to reason about the situations.

8.2 The test scenarios

8.2.1 Test scenario 1: A standard circuit

The first scenario with which the program was tested is a standard circuit. This means the pilot will takeoff, circle around the runway and land again (see figure 8.2). This is

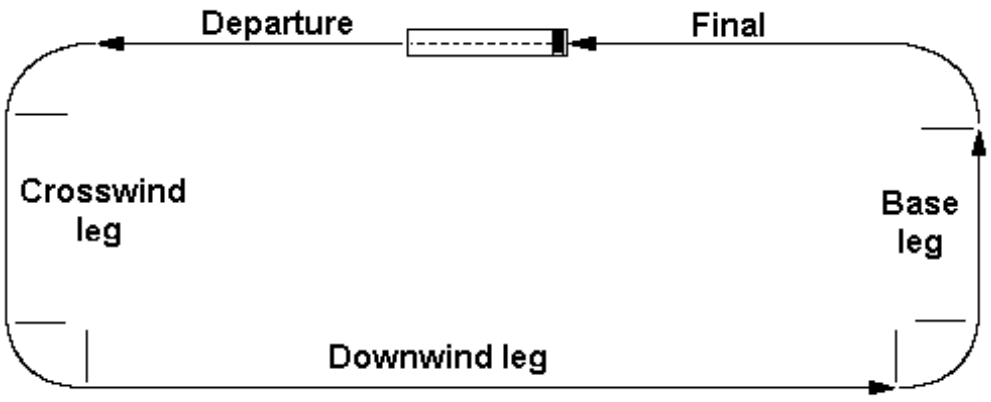

Figure 8.2: A standard circuit.

a simple flight that could also be flown with a civilian airplane like a Cessna. Therefore we will test the compatibility of the system with a knowledge base of another airplane by substituting the F-16 knowledge base by a knowledge base for a Cessna and flying a circuit with a Cessna. These results will be compared to the results we got when we flew the circuit with the F-16.

8.2.2 Test scenario 2: A problematic circuit

The second scenario will also be a standard circuit, but now we will simulate that there are problems during the takeoff because of which the pilot has to abort the takeoff. He will then try again and this time he will succeed. When the pilot starts to land we will do the same and simulate a problem during landing. The pilot will then abort the landing. After aborting the landing he will turn around and land the airplane again, this time without any problems.

8.2.3 Test scenario 3: An attack on a ground target

The third scenario will be one in which the pilot has to attack an imaginary ground target. It will be imaginary because it is not possible to perform an attack in the Microsoft Flight Simulator 2002. The pilot will have to imagine there is a ground target and perform all actions as he would have if there really had been a ground target. The results will be evaluated and compared to the results we got with the other scenarios.

8.3 The test results

What we want to deduce from the test results is not only if the program is able to detect the situations, but also how long it takes the program to detect the situations. To be able to get a good idea of the performance of the program in this respect we will compare the times at which the situations began during a flight with the times at which the situations were detected. As mentioned before in this report the times at which the situations begin are often not well defined. Therefore the starting times for the situations that have been given in the test results are only indications.

The test results will be presented in tables with three columns. The first column in the tables is for the names of the situations that have occurred or have been detected. In the second column we will put the times at which the situations have begun. These times are given in seconds from the moment the program was started. The third column is for the times at which the program has detected the situation. These are also given in seconds from the time that the program was started. From the tables we will be able to draw conclusions about the performance of the situation recognizer.

It might be that the program makes a mistake and detects a situation that did not occur. These situations will be in the table but will have no time value in the second row, but only a time value in the third row. Furthermore a situation can occur two times during a flight. In that case the situation will also occur two times in the table.

At every table we will give an explanation for the results.

8.3.1 Results of scenario 1: A standard circuit

A circuit flown with an F-16

The following table represents the results of the Situation Recognizer after testing it on a standard circuit flown with an F-16.

Table 8.1: The test results for the standard circuit flown with an F-16.

Situation	Time started (s)	Time detected (s)
Startup	0	0
Taxiing to runway	22	27
Taking off	52	56
Normal flight	84	75
Landing	230	242
Taxiing from runway	270	271
Shutdown	283	284

Startup When the program starts it assumes, correctly in this case, that the current situation is the Startup situation. So at time 0 it detects the Startup situation.

Taxiing to runway In the test flight the pilot finished starting up the systems in 21 seconds, after which he started taxiing by releasing the parking brakes and increasing the throttle. As soon as the program detected that the airplane started moving it knew that the Startup had finished and it detected that the pilot had started taxiing to the runway. The reason that it took the program a few seconds to detect the situation Taxiing to runway is that the start of the situation is considered to be the moment at which the pilot increases the throttle, but the program only knew for certain that taxiing had started when the airplane started moving, which happened a few seconds after the pilot had increased the throttle.

Taking off It took the pilot about 30 seconds after he started taxiing to get to the runway and get cleared for takeoff, after which he applied full throttle and when he had enough speed disengaged the NWS (Nose Wheel Steering). The moment at which the pilot applied full throttle was considered to be the start of the takeoff. The program did not detect the takeoff immediately, but as soon as the airplane reached a speed that was higher than the maximum speed that is allowed for taxiing (this is 20 knots) the program knew that the pilot had started taking off.

Normal flight The next column might seem a bit strange, because the situation Normal flight was detected earlier than it actually started. This is because it is unclear at what time the situation Normal flight starts after taking off. The start of the situation Normal flight was considered to be the moment at which the pilot leveled the airplane, which was after 84 seconds. But for the program it was already clear that the takeoff had finished after the pilot had raised the landing gears and had ascended above the minimum altitude for the flight, which he did after 75 seconds. Thus the program considered the situation Normal flight to be started after 75 seconds. The result could be improved in two ways. The first is to adapt our definition of the start of normal flight to the definition that is in the knowledge base, the other is to change the contents of the knowledge base so that it matches our definition. We have not done this here to be able illustrate this vagueness in the starts and ends of the situations.

Landing After the pilot had circled around the airfield and had aligned with the runway he started landing. This is another situation for which the start is unclear. We have taken the moment at which the pilot was aligned with the runway and started the descend as the moment at which he started the landing, which was after 230 seconds. The program only detected the landing at the moment at which the pilot lowered the landing gear,

which he did after 242 seconds, because prior to that moment the pilot could just as well have started a normal descent.

This approach means that the program will detect the landing very late if the pilot decides to lower the landing gear at the last moment. However in this prototype there is no other satisfactory way to detect that the pilot has started landing. In section 9.2 we will explain what changes are necessary to improve the performance of the program in this respect.

Taxiing from runway The pilot landed the airplane and hit the brakes to slow the airplane down. When the airplane had slowed down enough he engaged the NWS and started taxiing the airplane from the runway after the speed had become lower than the maximum taxi speed. The program detected the situation almost immediately.

Shutdown After 283 seconds the pilot stopped the airplane and engaged the parking brakes to shut down the systems. The program saw that the pilot had engaged the parking brakes and concluded that the pilot had finished the flight and had started shutting down the systems.

A circuit flown with a Cessna

After we modified the XML file with the knowledge base we flew the standard circuit with a Cessna 182 with retractable gear. The retractable landing gear is important because some types of the Cessna do not have retractable gears and then it becomes very difficult for our program to detect a landing or a takeoff. It would probably be possible if the knowledge base was adjusted for flying with such an airplane, but the adjustments we made to the knowledge base consisted mainly of removing the knowledge that was specific for an F-16, we did not add any knowledge that was specific for a Cessna.

Table 8.2: The test results for the standard circuit flown with a Cessna.

Situation	**Time started (s)**	**Time detected (s)**
Startup	0	0
Taxiing to runway	7	10
Taking off	22	27
Normal flight	61	59
Landing	119	121
Taxiing from runway	220	221
Shutdown	251	251

When we look at the results we see that they are similar to the results we got when we flew a circuit with an F-16.

Taxiing to runway After taxiing was started it took the airplane some time to gain speed. When it did the program detected that the pilot was taxiing to the runway.

Taking off When the pilot started the takeoff by moving the throttle to its maximum it took the airplane 5 seconds to get to a speed that was above the maximum taxi speed, which convinced the program that the pilot had started the takeoff.

Normal flight When we look at the time at which the situation Normal flight was detected we see that this happened before we considered the situation to have started, just as it did for the F-16. This is caused by the same difference in definition of the start of the situation Normal flight, as was described in the previous scenario. The time difference is smaller for the Cessna than for the F-16, because we leveled the airplane at a lower altitude than we did with the F-16.

Landing In this scenario the pilot lowered the landing gear soon after he was aligned with the runway and began its descend. This is why the landing was detected soon after it was considered to have started. For the Cessna the situations Taxiing from runway and Shutdown were detected very fast by the program just as for the F-16.

8.3.2 Results of scenario 2: A problematic circuit

This scenario delivered the following results.

Table 8.3: The test results for a problematic circuit.

Situation	Time started (s)	Time detected (s)
Startup	0	0
Taxiing to runway	20	23
Taking off	36	38
Aborting takeoff	45	50
Taxiing to runway	61	62
Taking off	138	140
Normal flight	172	161
Landing	330	338
Aborting landing	359	360
Normal flight	362	362
Landing	551	558
Taxiing from runway	647	649
Shutdown	660	661

Aborting takeoff Up until the Taking off situation the scenario and its results are the same as the previous scenarios. But this time the pilot moved the throttle to IDLE during the takeoff and hit the brakes. When it looked like the airplane would go too fast and there would not be enough runway the pilot lowered the hook. At this time the program detected that the pilot was aborting the takeoff. The fact that the program only detected that the pilot was aborting the takeoff when he lowered the hook means that if the pilot had not lowered the hook the program would not have detected that the pilot had aborted the takeoff. Unfortunately there is no way to improve the performance of the program in this respect using the probabilistic model. At first thought one might think the behaviour of the program could be improved by raising the importance of the actions of setting the throttle to IDLE and hitting the brakes. However these actions might also be performed during normal taxiing, for example if the pilot has to wait before entering the runway. Because the prototype uses the probabilistic model and not the causal model and because the program remembers the actions of the pilot for some time, the program might come to the conclusion that a takeoff is being aborted as soon as is has started if the pilot had to stop the airplane before entering the runway. In the causal model one could prevent this mistake by adding the knowledge that the action of

setting the throttle to IDLE should only be taken into account if the throttle had been set to the maximum recently. However, despite this limitation the program can still give good results as long as there is enough information about the situations, like the fact that the pilot lowers the hook.

Taxiing to the runway and taking off After the takeoff was aborted and the pilot increased the throttle to turn the airplane around the program detected that the pilot had started taxiing to the runway. This was correct in this case, but in some test runs the program thought that the pilot was taxiing from the runway. This is an understandable confusion, because it is not sure if the pilot will takeoff again after he has aborted a takeoff. This depends entirely on whether there is a malfunction in the airplane. When the pilot starts taxiing again after aborting a takeoff the program can not know if the pilot intends to takeoff again. The fact that the program says one time that the pilot is taxiing to the runway while it may say the next time that he is taxiing from the runway represents this uncertainty. This behaviour is not a problem since the program detected correctly that the pilot had started the takeoff when the speed of the airplane became higher than the maximum taxi speed even when the program thought that the pilot was taxiing from the runway after aborting the takeoff. This shows that the program is able to correct its own mistakes.

Landing and aborting a landing After taking off the pilot flew a circuit again and started landing. But during this landing the pilot aborted the landing by pushing the throttle to the maximum and raising the landing gear. This was enough for the program to know that the pilot was aborting the landing.

Normal flight As soon as the landing gear was raised and the airplane was flying above the minimum altitude again the situation Aborting landing was considered to be finished and the program detected that the situation Normal flight had started again.

Finishing the flight After turning the airplane around the pilot began to land the airplane again and this time he succeeded in landing the airplane. The program detected all further situations correctly as it did in the previous test scenarios.

8.3.3 Results of scenario 3: An attack on a ground target

The attack that was simulated by the pilot in this scenario was a DTOS (Dive Toss) attack. How such an attack is performed is described in detail in [15]. During the flight to the target the pilot simulated that he was navigating by calling up the steerpoint page on the DED (Data Entry Display). In a real F-16 this page displays information about the steerpoints in the flight plan. This was done twice before the attack was performed.

The test flight in which the pilot performed an attack on a ground target resulted in the following table.

Table 8.4: The test results for a ground attack.

Situation	Time started (s)	Time detected (s)
Startup	0	0
Taxiing to runway	10	12
		continued on next page

continued from previous page		
Situation	**Time started (s)**	**Time detected (s)**
Taking off	14	15
Normal flight	43	34
Navigating	83	83
Normal flight	91	91
Navigating	123	123
Normal flight	128	128
Visual attack	186	193
Normal flight	221	221
Landing	361	366
Aborting landing	-	410
Taxiing from runway	409	410
Shutdown	427	427

Navigating Until the situation Navigating there were no differences with the previous test scenarios. But after 83 seconds the pilot pushed the STPT button on the ICP (Instrument Control Panel) to call up the steerpoint page on the DED. At this moment the program detected that the pilot was navigating. As soon as the pilot called up another page on the DED, which was after 91 seconds, the program switched back to the Normal flight situation. This process was repeated in the 123th second.

The visual attack The DTOS attack was considered to have started when the pilot had lined up with the target and was flying straight towards it. The pilot had already set the master mode to A-G, the radar to A-G and had selected the DTOS page on the SMS. The master arm switch had already been set to Master arm when the pilot entered enemy territory. So when the pilot had aligned with the target the airplane was configured for the attack. Soon after the pilot was flying towards the target he began a dive and pushed the designate button to lock the target. He then slewed (moved) the radar cursors over the target. This might be necessary if the target lock is not exactly on the target. After slewing the radar cursors the pilot pulled up, pressed the pickle button and held it depressed until the moment at which he should release the bombs. After the bombs were released he switched the master mode to A-A to watch for enemy fighters.

The moment at which the program detected the attack was the moment at which the pilot slewed the radar cursors. The fact that the pilot designated the target was already a big indication that the attack was being performed, but not enough for the program to decide that it must be the current situation. When the pilot slewed the radar cursors the program was convinced that the pilot was locking onto a target and was performing a visual attack. When the pilot switched the master mode to A-A after the bombs had been released the program assumed that the pilot had finished the attack and concluded that the situation Normal flight had started again.

Aborting landing The landing was a normal landing and was detected correctly as in the previous scenarios. But as is shown in the table the program thought for a moment that the landing was being aborted. This happened at the moment that the airplane was already on the ground and had slowed down to the maximum taxi speed. At that moment the program knew that the landing had been finished and looked for the situation with the highest start probability. This turned out to be the situation Aborting landing. This is probably because at some point since the start conditions of the situation Aborting

landing had been satisfied the pilot had moved the throttle to the maximum. This action was still in the memory of the program when the landing ended. And if the start conditions had been satisfied before the landing gear was lowered even the fact that the landing gear had been raised would be in the memory of the program. Because of this the probability that the landing was aborted was high and the situation Aborting landing became the current one once the landing had finished. However as soon as that happened the program realized that the airplane was actually taxiing and it corrected the mistake in the next iteration.

This mistake could be prevented in the causal reasoning model that has been described in this report. In such a model where causal relations between actions and events can be reckoned with, the program can for example remove the action that the throttle had been set to the maximum when the throttle was reset to a lower value. Or it could only look at the action of raising the landing gear if the landing gear has been lowered recently and not all the time as it does now.

Shutdown When the pilot stopped the airplane and applied the parking brakes the program detected the situation Shutdown correctly.

8.4 Evaluation of the test results

We will now give a short evaluation of the test results that have been described in this chapter. In chapter 9 we go into more detail and give recommendations for future work.

8.4.1 Performance of the program

The test results

The test results are quite satisfactory. The time between the start of a situation and the time at which the program detects the situation is in most cases quite small. Those cases in which it takes the program a while longer to detect a situation are usually the cases for which the start of a situation is not very clear and the definition of the start of the situation that we used was different from the definition of the start of the situation that was used in the knowledge base. We have even seen that this difference in definitions caused the program to detect the situation Normal flight before it was considered to have started. The performance of the program can be improved for these situations by adjusting our definition of the start of the situations or by adjusting the knowledge base so that the definition of the start of the situation in the knowledge base matches our definition.

The program has shown it can be used with other airplanes than the F-16, because it detected all the situations correctly when a standard circuit was flown with a Cessna. This only required the knowledge base to be adjusted for the Cessna.

Furthermore the program has shown it can recognize the correct situation even if the pilot makes a mistake, because the pilot that flew the test flights was not a real pilot and did make some mistakes during the test flights. The system has also shown and that it can correct its own mistakes, because when it thought that the landing was aborted when the pilot started taxiing after a ground attack was performed, it corrected itself immediately and detected the correct situation.

The real time performance

The program was able to draw conclusions in real time. The program was set to perform a recognition loop five times per second. The program succeeded in doing this most of the time, only when another process required a lot of processor time the program needed more time to perform the recognition loop.

8.4.2 Problems encountered during testing

When we tested the program we encountered some problems that could not be solved in the time that was available. The first problem we encountered was that sometimes the variables were not read from the flight simulator when the program was started. It seemed to happen only when a new flight was started after a flight video had been recorded. When the program was run it did not read the variables from the airplane with which the new flight was started correctly. After restarting the flight simulator and the program the problem was solved and the variables were read correctly again. When this happens it is advised to restart the flight simulator and the program.

Furthermore, we discovered that sometimes when the flight simulator is being run together with the Situation Recognizer and a lot of processor time is taken by the flight simulator and the recognition loop of the Situation Recognizer, the graphical interface of the Situation Recognizer is not updated correctly. This only happens when the table with the probability values is visible on the screen. Sometimes the names of the situations that occupy the first column of the table appear not only in the first column, but also in the second or even in the third. Once the processor is not so occupied anymore the table appears normally again. This is probably a bug in JAVA that could not be fixed. It is not a big problem but can be annoying sometimes.

Otherwise no big problems were encountered and the program worked correctly.

Chapter 9

Conclusions and recommendations

In this chapter we will discuss the conclusions we can draw about the performance of the Situation Recognizer. We will do this in section 9.1. Based on those conclusions we will give some recommendations for future work in section 9.2.

9.1 Conclusions

9.1.1 Project overview

In this research project we have investigated the effectiveness of a system that tries to detect the current situation during a flight with an F-16. First of all knowledge has been gathered about how to fly an F-16. This knowledge has been stored in a knowledge base in the form of an XML file. This was done to keep the knowledge definition as generic as possible so that the system would be able to work with other airplanes just by changing the knowledge base.

When the knowledge had been gathered a pilot model was created to understand the cognitive processes of a pilot during a flight. After that we discussed a number of artificial intelligence techniques that can be used in the system. When that had been done the architecture of the system was created and the system requirements were defined. Then we described three possible reasoning models for the recognition process. These were a finite state model, a probabilistic model and a causal model. Of these models the finite state model turned out to be too simplistic to satisfy the requirements that had been defined. The main reason that the model was too simplistic was because it assumed that the transitions from one situation to another could be clearly defined. Because some of the situations may overlap these transitions are usually very vague and the performance of the finite state model was unsatisfactory.

After the finite state model had been abandoned the probabilistic model and the causal model were described in detail. A prototype was implemented based on the probabilistic model. The probabilistic model uses an expert system with a rule base to reason about the current situation. The rules in the rule base are created from the knowledge in the knowledge base. When the program is run the rules produce a number of probabilities about the likeliness that a situation is occurring. These probabilities are put in a number of Bayesian belief networks that have been created. These Bayesian

belief networks calculate for every situation the probability that the situation is occurring and the probability that the situation is not occurring. From these resulting probabilities a conclusion is drawn about the situation that is most likely to be the current one.

9.1.2 The test results

The prototype was implemented in Java and tested on a PC running Microsoft Windows 2000. The flight simulator that was used to fly the test flights is the Microsoft Flight Simulator 2002. The results show that the performance of the Situation Recognizer program is good. It made very few errors and when it did it corrected them immediately. The time difference between the actual start of a situation and the moment at which the situation was detected varies for most situations between zero and five seconds. Although we have not tested the program on all situations that have been defined in the knowledge base, the performance of the program was so good that we do not have any reason to suspect that the program would perform worse on the situations that have not been tested.

The results show that the probabilistic model can be used effectively to detect the current situation during a flight with an F-16. The system has shown that it is able to work with other airplanes than the F-16, just by changing the knowledge base. This feature is a very important one because that sets it apart from the system that were described in chapter 1. Another important aspect of our program is that it can work in real time, something that some of the other systems had problems with.

9.1.3 Satisfying the requirements

When we look back at the requirements that were set out for the program in the introduction and that were refined in section 4.3 we can make the following remarks:

1. The program was able to work with only limited information about the actions of the pilot, about the environment and about the state of the airplane. There was time to implement only the most important gauges in the F-16 cockpit that was created, so we have not been able to use the entire knowledge base. But even with this restriction the program functioned very well.

2. The tests were all performed in real time. The interval in which the program drew a conclusion was 200 ms. The program needed more time only now and then when another program needed a lot of processor time. This means that most of the time the program made five recognition loops per second. Furthermore the program running in the background did not significantly affect the performance of the flight simulator.

3. The program has shown to be reliable, because:

   (a) The program makes a mistake rarely and when it does it has shown that it is able to correct them very fast.

   (b) The program has shown to be able to cope with pilot mistakes, provided that the mistakes are not too big. For example, if the pilot forgets to lower the landing gear during a landing the program might not detect that the pilot is trying to land the airplane until it hits the ground. Since it is not likely that a pilot will forget such a vital action this restriction is not considered to be a big problem.

4. It is possible to use the Situation Recognizer with another program like an AI-bot, but because there was no good idea about the interface that would be needed between the Situation Recognizer and such a program, some adjustments to the Situation Recognizer would be necessary. However the program has been designed to keep those adjustments as easy and as few as possible.

5. The test flight with the Cessna showed that the program can work with other airplanes as long as the knowledge base is adjusted for the other airplanes.

9.1.4 Further remarks

When we tested the prototype it became increasingly clear that it is vital that the knowledge in the knowledge base is structured correctly. This means for example that the start constraints that are given in the knowledge base are constraints that always have to be satisfied for a situation to have started. An example of a mistake that was made initially is that we had added a start constraint to the landing situation that the altitude of the airplane had to be bigger than the minimum altitude. This worked fine for normal flights in which no mistakes were made. But suppose that the program makes a mistake and thinks that a pilot is aborting a landing which turns out not to be true. If this happens after the airplane has already descended to an altitude that is below the minimum altitude, this constraint would prevent the program from correcting its mistake. This is just one example of a mistake that can be made easily if one does not think very carefully about the information that is put into the knowledge base.

Another important aspect of the knowledge base are the probabilities that are specified for the constraints, actions and events. As was mentioned before these probabilities should ideally be based on expert knowledge or on statistical data. Since this was not available for this project the probabilities have been specified by using common sense. As expected a lot of the probabilities had to be adjusted during the tests that were performed. But as the test results show we have eventually found probabilities that were good enough.

Finally it is very important to have enough information about the situations that are stored in the knowledge base. If the knowledge base contains two situations with little knowledge that are a lot alike it will be very difficult for the program to decide which one of those two is actually happening.

9.2 Recommendations for future work

If the Situation Recognizer is to be used by other programs we recommend the following steps to be taken:

- To make the Situation Recognizer more reliable, it should be extended to incorporate causal relationships between events and actions. A possible causal model has been described in this report. For this model to work it might be desired to adjust the structure of the knowledge base. If this is necessary we recommend to stay as close as possible to the design guidelines that have been set out in [15].

- The program should be extended to incorporate a flight plan. It is expected that the ability to use the information in a flight plan will improve the performance of the program.

- Fuzzy reasoning should be added to the program to calculate the uncertainty in the input and incorporate that uncertainty in the calculations of the probabilities.
- The memory of the rule base could be refined. The prototype that has been implemented keeps all the facts concerning a situation in the rule base until the end of a situation is detected. This means that if a situation never ends because it is never detected the facts that concern that situation are never removed from the rule base. This behaviour could lead to mistakes. This could be improved by putting facts in the rule base with a time stamp and removing it after a specified time. This time should probably depend on the time window of the situation that the fact applies to.
- The system should be adapted so that it is able to adjust the time interval it needs to perform the recognition. The system should make that interval smaller in situation in which a lot is happening and the workload of the pilot is high, because in those situations it is important for the program to detect changes in the situation very fast.

Bibliography

Literature

[1] *Pilot operational procedures – F-16*, 21 April 1995.

[2] *F-16 combat aircraft fundamentals*, 10 May 1996.

[3] Bengio, Y., *Markovian models for sequential data*, Neural Computing Surveys 2, 129-162, 1999.

[4] Berliner, M., *Hierarchical Bayesian time series models*, National Center for Atmospheric Research & Ohio State University.

[5] Cooper, G., *Probabilistic inference using belief networks is NP-hard*, 1990.

[6] Ehlert, P.A.M., Rothkrantz, L.J.M., *The ICE project: The intelligent cockpit environment.*, Delft University of Technology, 2003.

[7] Falcon Unified Team, *Falcon 4, Superpak 3, User Manual*, Infogrames Inc.

[8] Fries, T.P., *Consensus Development in Fuzzy Intelligent Agents for Decision Making*, Department of Computer Science, Coastal Carolina University, Conway.

[9] Guo, H., Hsu, W., *A Survey of Algorithms for Real-Time Bayesian Network Inference*, Laboratory of Knowledge Discovery in Databases Department of Computing and Information Sciences, Kansas State University.

[10] Lalmas, M., Ruthven, I., Theophylactou, M., *Structured document retrieval using Dempster-Shafer's Theory of Evidence: Implementation and evaluation*, Department of Computing Science, University of Glasgow.

[11] Lazareva-Ulitsky, B., Haussler, D., *A probabilistic approach to consensus multiple alignment*, Department of Computer Science, University of California.

[12] Lucas, P.J.F., *Certainty-factor-like structures in Bayesian belief networks*, in *Knowledge Based Systems 14*, pages 327-335, 2001.

[13] Machado, R., *Rod Machado's Ground School*, Microsoft, 2002.

[14] Microprose, *Falcon 4.0, the new benchmark in flight sim technology*, Microprose.

[15] Mouthaan, Q.M., *Flying an F-16: a knowledge base describing the situations an F-16 pilot may encounter*, Knowledge Based Systems Faculty of Information Technology and Systems, Delft University of Technology, Delft, 2003.

[16] Mulgund, S.S., Zacharias, G.L., *A Situation-Driven Adaptive Pilot/Vehicle Interface*, presented at *Human Interaction with Complex Systems Symposium*, August 1996.

[17] Ng, A.Y., Jordan, M.I., *Approximate inference algorithms for two-layer Bayesian networks*, Computer Science Division and Department of Statistics, UC Berkeley, 1999.

[18] Nigro, J-M., Loriette-Rougregez, S., Rombaut, M., *Driving situation recognition with uncertainty management and rule-based systems*, in *Engineering Applications of Artificial Intelligence 15*, pages 217-228, 2002.

[19] Pan, H., McMichael, D., *Fuzzy Causal Probabilistic Networks - A new ideal and practical inference engine*, Cooperative Research Centre for Sensor Signal and Information Processing, Technology Park Adelaide, 22 May 1998.

[20] Parsons, S., Bigham, J., *Possibility theory and the generalised Noisy OR model*, Department of Electronic Engineering, Queen Mary and Westfield College, London.

[21] Pearl, J., *Graphical models, causality, and intervention*, Cognitive Systems Laboratory Computer Science Department, University of California, Los Angeles.

[22] Pennock, D.M., Wellman, M.P., *Graphical Representations of Consensus Belief*, in *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 531-540, 1999.

[23] Portinale L., Bobbio, A., *Bayesian Networks for Dependability Analysis: An Application to Digital Control Reliability*, Dipartimento di Scienze e Tecnologie Avanzate, Universitá del Piemonte Orientale, Alessandria.

[24] Rao, A.S., Georgeff, M.P., *BDI Agents: From Theory to Practice*, in *Proceedings of the First International Conference on Multi Agent Systems*, June 1995.

[25] Russel, S., Norvig, P., *Artificial intelligence, a modern approach*, Prentice Hall International Editions, 1995.

[26] Santos, E.Jr., Solomon, E.S., *Belief Updating by Enumerating High-Probability Independence-Based assignments*, in *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 506-513, 1994.

[27] Shafer, G., *A mathematical theory of evidence*, Princeton University Press, 1976.

[28] Shafer, G., *Dempster-Shafer Theory*, http://www.glennshafer.com/assets/downloads/article48.pdf.

[29] Smets, P., *What is Dempster-Shafer's model?*, in *Advances in the Dempster-Shafer Theory of Evidence*, pages 5-34, 1994.

[30] Thijssen, C.P.R., *SmileX: An ActiveX Decision-Analytic Reasoning Engine and Its Application to Evaluation of Credit Applicants*, MSc. thesis, Knowledge Based Systems Faculty of Information Technology and Systems, Delft University of Technology, Delft, 1999.

[31] Voorbraak, F., *Reasoning with Uncertainty in AI*, Department of Mathematics, Computer Science, Physics and Astronomy, University of Amsterdam.

[32] Wellman, M.P., Breese, J.S., Goldman, R.P., *From knowledge bases to decision models*, in *Knowledge Engineering Review*, 1992.

[33] Zhang, N.L., Poole, D., *Exploiting Causal Independence in Bayesian Network Inference*, in *Journal of Artificial Intelligence Research 5*, pages 301-328, 1996.

[34] Zhang, N.L., *Inference in Bayesian Networks: The Role of Context-Specific Independence*, Technical Report HKUST-CS98-09, Department of Computer Science, The Hong Kong University of Science and Technology, Hongkong, 1998.

[35] Zhang, N.L., *Inference with Causal Independence in the CPSC Network*, Department of Computer Science & Technology, The Hong Kong University of Science and Technology.

Internet

[36] http://www.w3c.org/XML

[37] http://www.w3c.org/XML/Schema

[38] http://www.uml.org

[39] http://herzberg.ca.sandia.gov/jess

[40] http://www.microsoft.com/games/fs2002

[41] http://zone.msn.com/flightsim/FS02DevDeskSDK00.asp

Appendix A

The test environment: MS Flight Simulator 2002

A.1 Introduction

Because we do not have access to a real F-16 simulator, the Situation Recognizer program will be tested with a flight simulator on a PC. After considering several possible options we have chosen to use Microsoft Flight Simulator 2002 (see [40]) for this purpose. The reasons for this choice are the following:

- The Microsoft Flight Simulator is very realistic, A lot of effort has gone into creating flight models that are as realistic as possible.
- It is possible to change some flight dynamics, so although there is not yet a standard flight model for the F-16 built into the Microsoft Flight Simulator, such a model can be approximated by changing the flight dynamics.
- It is possible to create your own cockpit. This is a very important feature because as mentioned above, there is not a standard F-16 airplane in the Microsoft Flight Simulator. Therefore we will have to create one ourselves. Microsoft provides some SDK's to develop your own airplane (see [41]) for this. This is further explained in section A.2.
- It is possible to get flight data from the flight simulator. This is also a very important feature, because we need the flight data to reason with.
- Several extensions for Microsoft Flight Simulator with F-16 airplanes have been developed and are available on Internet. None of these fit our purposes exactly but by using parts of those extensions we can shorten the time needed for developing the test environment.

There are also some disadvantages of using the Microsoft Flight Simulator:

- The program is not open source. This limits our possibilities significantly. Thankfully, as mentioned before, Microsoft has developed some SDK's with which we can access flight data and create custom cockpits.

- The Microsoft Flight Simulator 2002 has been created solely for simulation of flying an airplane. This means we will not be able to simulate combat situations. This is a big disadvantage, but because our program will only look at the pilot's actions and the flight data and not at the environment, we will still be able to test the Situation Recognizer on most situations by flying as if we are in a combat situation.

As mentioned before we have to create the cockpit of the F-16 ourselves. This has been done by several other people before who have published their work on the Internet. None of these cockpits match our purposes exactly but we will use some of the elements of these cockpits to create our own and then add the necessary functionality. We will describe globally how to create a cockpit in Microsoft Flight Simulator 2002 in section A.2. Then in section A.3 we will describe the F-16 cockpit that we have created. Finally in section A.4 we explain how the data from the flight simulator is retrieved by the Situation Recognizer program.

A.2 Creating your own cockpit

Understanding how to create your own cockpit in Microsoft Flight Simulator 2002 is not trivial. Once you get the knack of it though, you can get very nice results with not too much effort. Microsoft provides the Panels SDK for developing your own cockpit. It goes too far to explain exactly how we have implemented the F-16 cockpit and how we have integrated it into Microsft Flight Simulator 2002. We will only give the global outlines of what we have done in this document, for more information have a look at [41].

Creating your own cockpit is done in the following steps:

1. Draw an image of the cockpit without the instruments (or gauges as they are called in the Panels SDK).
2. Draw images of the gauges.
3. Implement the behaviour of the gauges in C using the Panels SDK.
4. Compile the implemented gauge files and the images into one or more dll's with the extension gau.
5. Add the gau file(s) to the subdirectory called "gauges" in the Microsoft Flight Simulator directory.
6. Copy the image of the cockpit to the panel directory of the directory of the airplane.
7. Edit the panel.cfg file in the panel directory to place the gauges in the gau file(s) in the cockpit.

In the Microsoft Flight Simulator directory there is a subdirectory called "aircrafts" with subdirectories for all the airplanes that can be flown with in the flight simulator. Each of those airplane directories has a subdirectory called "panel". This directory is the directory we call the panel directory in the previous enumeration. In the following sections we will explain points 3 and 7 in more detail.

A.2.1 Implementing the behaviour of the gauges

Implementing the behaviour of a gauge usually means that the picture of the gauge has to be adjusted according to the value(s) of one or more variables. It can be that the entire picture of the gauge has to change, for example a light that is switched on or off, or that a part of the gauge picture moves in relation to the rest of the picture, for example the needle of the altitude gauge that turns when the altitude of the airplane changes. To get the values of the variable from the flight simulator and to implement the behaviour of the gauges a number of macros and methods have been defined in the file called gauges.h which belongs to the Panels SDK. In the following piece of code we give an example of how the altitude of the airplane is retrieved and given to the variable `alt`.

```
float alt;
MODULE_VAR altitude = PLANE_ALTITUDE;
initialize_var(&altitude);
lookup_var(&altitude);
alt = altitude.var_value.n;
```

A list of the variables that can be retrieved from the flight simulator can be found in the file TokenVars.doc which is part of the Panels SDK. It goes too far to explain the complete functionality of the Panels SDK here. For a further explanation we refer to the documentation that comes with the Panels SDK.

A.2.2 Editing the panel.cfg file

Once the images of the gauges and the files containing the implemented behaviour have been compiled to one or more gau files we need a way to place the cockpit on the screen of the flight simulator and to place the gauges in the cockpit. This can be done in the file panel.cfg in the panels directory. We will not explain the complete structure of the panel.cfg file, we will only explain how to indicate where the picture of the cockpit can be found and how to place the gauges in the cockpit.

The panel.cfg file is divided in a number of sections where each section defines a window that can be shown on the screen during the flight. In this way it is possible to define the main window containing the biggest part of the cockpit and to define for example an overhead window that contains a picture of an overhead panel with its instruments. It is also possible to define what the pilot should see when he looks in another direction. To give an impression of what a panel.cfg looks like we will show a part of the panel.cfg file that was used for the F16.

```
// define the windows that will be present in the cockpit.
[Window Titles]
Window00=Main panel

// this is the main window that contains the cockpit
[Window00]
// Tell the flight simulator where to find the image for
// this window.
file=f16_Falcon.bmp
// The size of the image.
size_mm=1024,786
// Tell the flight simulator at which position it should place
// the image.  7 is the position on the bottom in the centre.
position=7
// The image should be visible from the start.
visible=1
// The string that identifies this window.
ident=MAIN_PANEL

// In the next section the gauges are place in the cockpit.
// The format is:  gaugeXX=gaufile!gaugename, x,y,width,height

// Place the gauge with the name ICP (Instrument Control Panel)
// from the file f16c.gau at pos 416,421 with dimensions 150,129
// in the cockpit.
gauge00=f16c!ICP, 416,421,150,129
// Place the gauge with the name MasterModePanel from the
// file f16c.gau at pos 413,385 with dimensions 197,22 in
// the cockpit.
gauge01=f16c!MasterModePanel, 413,385,197,22
```

A.3 The F-16 cockpit

When we started designing the cockpit of an F-16 we had to make a choice about which instruments we would put in the cockpit. For the standard flight variables like altitude and airspeed we already had working gauges that resembled the instruments in the cockpit of an F-16 and which were freely available on the Internet. The gauges that we needed to implement ourselves were gauges for instruments and switches that were not standard flight instruments, like the Master Arm switch for example. After we had determined which gauges we would have to implement we had to decide if we would make a single gau file for all the gauges or a separate gau file for each gauge. To avoid the extra overhead that would be caused by trying to keep track of all the separate gauge files and copying them all to the gauges directory of the Microsoft Flight Simulator we decided to compile all the gauges to a single gauge file which we have called f16c.gau. The implementation of the gauges has been done in separate files though for the sake of scalability and modularity.

To give an idea of what the cockpit looks like and what gauges have been implemented figure A.1 shows the cockpit during a flight.

Figure A.1: The cockpit that was created during a flight.

A.4 Getting the data from the flight simulator to the Situation Recognizer

Now that the cockpit has been created we have to get the data from the flight simulator to the Situation Recognizer program. For this the Situation Recognizer program has to load the f16c.gau file as if it was a dll. However, it is not possible for the program to read the variables directly from this file. This is because the memory that is reserved for the dll when the Situation Recognizer program loads it is not the same piece of memory that was reserved for the dll when the Microsoft Flight Simulator loaded it. The following solution was found for this problem. When the flight simulator updates the gauges the dll is notified. At that moment a number of flight simulator variables are written to a shared piece of memory. When the Situation Recognizer program makes a request for the flight simulator data the dll reads the variables from the shared piece of memory and gives them to the Situation Recognizer program. Because the Situation Recognizer is a Java program and can not call methods in a dll directly we have created another dll from which the methods in the F16C.gau file are called. The Situation Recognizer program communicates with this second dll via JNI (Java Native Interface). The Situation Recognizer could not communicate directly via JNI with the F16C.gau file, because a dll with which can be communicated via JNI has to satisfy certain demands. An overview of this process is shown in figure A.2. The exact implementation will not

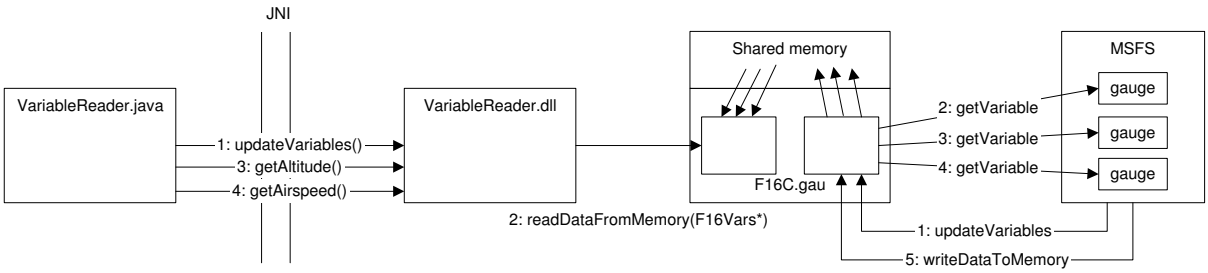

Figure A.2: The way in which the flight simulator variables are read.

be given here, for now it is enough to know the strategy that was used to get the data from the flight simulator to the Situation Recognizer.

Appendix B

Interview report: Interview with an F-16 pilot

B.1 Agenda

To get as much information out of the interview as possible I wanted to let mr. Reisel fly a couple of missions with the Falcon 4 flight simulator while observing his actions. After that I asked the questions that had yet remained unanswered. The agenda looked like this:
9.00 − 10.00: Introduction to the project.
10.00 − 12.30: Flying and monitoring missions with Falcon 4.
12.30 − 13.00: Lunch.
13.00 − 14.00: Answering remaining questions.

B.2 Questions and answers

To describe what information was exchanged during the interview the most interesting questions with the answers to them are given here.

Taxiing:

Q: There are three different kinds of brakes: speed brakes, wheel brakes and parking brakes. What exactly is the difference between these and when are they used?
A: Speed brakes are brakes which decrease speed while in the air as well as on the ground. They are located between the horizontal stabilizer and the rudder. The wheel brakes can be controlled with pedals at the pilot's feet. They have anti skid, so that the wheels can not block. Therefore there is no maximum speed above which the wheel brakes may not be used anymore. The parking brakes may only be used when the airplane is standing completely still, because these really block the wheels.

Q: If the pilot is taxiing and he is cleared for takeoff when he arrives at the runway is he allowed to continue without holding short?
A: Yes, the pilot can immediately enter the runway and takeoff without delays.

Q: When is NWS switched off?

A: As soon as the IAS indicator shows movement [this is at approximately 60 kts] the rudder is effective and the NWS becomes too sensitive and then the rudder will be the primary means to steer. As the nosewheel and the rudder are both controlled by the rudder pedals at the pilot's feet, the changeover is easy.

Q: Are the fuel tanks always completely full at takeoff?
A: The inner fuel tanks usually are, the extra outer fuel tanks may not be full. An exception might be an airplane that is giving a demonstration at an air show. Then the inner fuel tanks might not be full either.

Taking off:

Q: Is the afterburner often used at taking off?
A: Usually the afterburner is not used when taking off. Only when the stopping distance becomes a factor the take-off roll may be shortened by using the afterburner.

Q: At what moment must the pilot decide at the latest to abort a takeoff?
A: When the pilot reaches the calculated Refusal Speed. That speed is determined during the briefing and depends on the weight of the aircraft, the length of the runway etc.

Q: Is the hook usually lowered if a takeoff is aborted?
A: Yes, but the hook has proven to be unreliable, so it is still used, but mostly the parachute is deployed first.

Q: Could it happen that the pilot tries to takeoff again after aborting a takeoff?
A: Yes, if the takeoff was aborted due to something that does not affect the functionality of the airplane, the pilot might taxi to the runway and takeoff again. The only limiting factor may be the temperature of the brakes. They need to cool down before raising them into the wheel wells.

Q: What is the maximum gear speed?
A: Around 300 knots.

Navigating:

Q: What is the most important instrument that is used to stay on course?
A: The route between two steerpoints is called a track. When you follow the INS steering to the next steerpoint you will fly the correct track. The heading indicator can not be used to stay on this track, because the heading depends on the wind. The nose has to be turned into the wind to fly a straight path. Therefore visual checks of the environment are very important for navigating to the steerpoint and for evaluating the performance of the Navigation Systems.

Dogfight:

Q: Might some missiles be fired without a radar?
A: Most missiles can be fired without a radar lock. It is more difficult though because the distance to the target is not known. It might be too far away or too close. Furthermore the radar will calculate all the data compiled in a graphic display on the HUD called a DLZ or dynamic launch zone.

A-G attack:

Q: At what moment is the Master Arm switch set to ARM?
A: The moment enemy territory is entered.

Q: Which master mode is switched to after an attack?
A: It is usually switched to A-A mode. This gives almost the same info as NAV with the added advantage that the pilot is ready for other fighters.

Q: When a LGB attack is performed by two fighters of which one has to lock the target with his laser, what is the task of the supporting fighter?
A: The supporting fighter will circle at a safe distance around the target while he keeps his laser on the target. The laser arm switch is set to ARM at the latest moment possible because otherwise it emits a signal that might be picked up be enemy units. The fighter with the LGB's will toss his bombs in a cone above the target that assures the weapon(s) can "glide" towards the laser signal.

Q: What does FTT (Fixed Target Track) mean exactly?
A: When a target is designated twice it is in FTT, which results in some extra info about the target being displayed.

Taking evasive action:

Q: Should the master lights be switched off when a missile launch is detected?
A: In wartime the lights will always be switched off.

Deep stall recovery:

Q: What does the speed meter display in a deep stall?
A: Airflow is severely disrupted around an airplane in deep stall, therefore are indications on any instrument are highly unreliable. The airspeed should be bigger than zero. It is difficult to say exactly, but it could be plus minus 80 knots.

Air refueling:

Q: How is the TACAN set for air refueling?
A: The TACAN can be used to find the tanker, when the TACAN channel of the tanker is known. The F-16 pilot can then set his TACAN channel to a value that is a set distance from the channel of the tanker. This way the pilot can get a range on the tanker.

Landing:

Q: What is the airplane state before landing? Should the radar be switched off?
A: All weapon systems have been shut down on exiting the enemy territory, so they do not have to be shut off anymore. Radar does NOT have to be shut off before landing. It should be shut off before taxiing because the radar waves might not be healthy for the ground crew.

Q: How often does the pilot use ILS when landing?
A: Landing on instruments is only done when the visibility is very poor. Given a choice a pilot will always land the plane by visual means.

Q: Is the afterburner used when a landing is aborted? A: Usually not.

Shut down:

Q: What is the shut down sequence? And is it always done by the pilot or are some things done by the ground crew?
A: The shut down sequence is about the same as the startup sequence. It is done by the pilot, only some things on the outside of the airplane are taken care of by the ground crew.

Instruments:

Q: What is the difference between the positions START1 and START2 on the JFS switch?
A: The JFS is a starter for the engine. It has two bottles. When the JFS is in START1 only one bottle is used to start the engine. In START2 both bottles are used.

Q: What values does the Master Arm switch have? In the RC's there is a value AUTO?
A: The value AUTO does not exist. The Master Arm has the values: ARM, SIM and OFF.

Q: What do the VRP and VIP modes mean?
A: The VRP mode is a kind of offset mode. The target is known relative to another point. The radar cursors are slewed towards the target. In the VIP mode, the pilot notifies the system when he flies over the IP. Then the computer knows the position of the target relative to that IP.

Q: How does the IFF work?
A: The IFF broadcasts information about the airplane. It should be switched off in enemy territory and before air refueling.

Q: Can the trim be adjusted on the trim panel, or is that only a display of the trim settings?
A: Trim is set automatically in the F-16. It can be manually adjusted though on the trim panel but also with one of the buttons on the stick.

Q: What buttons are used to slew the radar cursors?
A: There is a small joystick on the throttle.

Q: What is the functionality of the ATT/FPM switch and must it be on for performing an A-G attack?
A: If the ATT/FPM switch is set to on, the FPM adjusts itself for the crosswind. This can be annoying when the crosswind is very strong, because the FPM might go out of the HUD. It is important that the switch is on for an attack.

Visual checks:

Q: Is it necessary to check the climbing rate during air refueling or is it enough to look at the tanker?
A: The most important variables during the approach to the tanker are the distance and

overtake values displayed in the HUD. When the pilot gets closer the visual cues will become more important and the attention of the pilot will be directed mainly at the tanker.

Q: Is the RPM gauge often checked?
A: No not really, the FTIT is checked more often, because that gives an indication of the temperature of the engine. The RPM is variable and depends on the outside atmospheric conditions.

Q: Does an F-16 pilot use his peripheral vision to check the instruments in the cockpit?
A: No, if an F-16 pilot wants to check something he looks directly at it.

B.3 Extra information

Next to the answers of the questions a lot of information was given by mr. Reisel outside the scope of the questions which was still very interesting. Some of that information is given in this paragraph.

- The most crucial information for which the presentation could be improved in the F-16 is the airspeed. It would be very convenient for a pilot to get a regular airspeed update or warning. The airspeed is very important for the energy of the airplane and it would help a lot if "bitching betty" would shout the airspeed in a dogfight or when the airspeed becomes low in a non landing situation.
- The EPU has three positions: OFF, AUTO and ON. When it is set in the AUTO position it will turn on automatically when the engine quits, when hydraulic pressure depletes or when the main engine driven generator fails.
- An evasive tactic that is often used for bullet shooting anti air guns is to change dimension every few seconds. This makes it impossible for the bullet shooter to predict the next position of the fighter to shoot at.
- In commercial airplanes the crucial altitudes are announced by a voice during the last part of landing which allows the pilot to concentrate on the landing better and to get a good feel for the "rate of descent" and trends.
- When the airplane gets in the danger zone of getting into a deep stall, a loud horn will sound to give the pilot warning. It is said that the pilot is then in the Dog House.
- The standard scanning pattern for airplanes with a HUD includes the attitude from the HUD instead of the attitude of the AID. This is because the attitude from the HUD compensates for airspeed. At low airspeed the nose has to be raised to keep flying level, In that case the AID indicates a higher attitude, but the attitude in the HUD will compensate for the lower airspeed and stay the same. For the airspeed and the altitude the indicators in the cockpit are more often used than the indicators in the HUD, because pilots find it easier to discover a trend when looking at a clock. They get a better feeling of the variations in the altitude and airspeed from the clocks than from the digital numbers in the HUD. What could be useful is an arrow that indicates in which direction the airspeed or altitude is expected to go, this is already used in some commercial airplanes.

Furthermore mr. Reisel gave some information about the switches and buttons that are located on the throttle and the stick.
Throttle:

- A joystick for slewing radar cursors.
- A switch for the speed brakes.
- A switch for the dogfight or missile override modes.
- A button to control the radar tilt angle.
- And a knob to switch the radio between UHF and VHF.

Stick:

- A trim button.
- The NWS button.
- The gun trigger.
- The pickle button.
- The designate / RTS (Return To Search) button.

B.4 The time windows

For all the situations mr. Reisel gave an indication of how long they would take on average:

Startup: $4 - 10$ minutes. Pilots that are on standby and must be in the air ASAP, a plane can be made ready in advance and then the pilot can be ready in 4 minutes. Normally the startup sequence will take about 10 minutes.

Taxiing: 3-5 minutes. The time needed for taxiing depends on the distance to the runway or hangar of course.

Taking off: 1 minute.

Dogfight: $2 - 30$ minutes. In wartime a dogfight might last as short as 2 minutes, provided that the F-16 does not fight another F-16, this is because of the superior qualities of the F-16 for dogfighting. In peacetime during training a dogfight between two F-16's might last as long as 30 minutes, but normally in wartime, a dogfight will not take that long, because there is no time for that.

A-G attack: $2 - 3$ minutes. This is the time it takes to get from the IP to the target.

refueling: 5 minutes.

Landing: 2 minutes.

B.5 Conclusion

After the interview it was clear that the knowledge that was collected was quite accurate. A number of corrections of and additions to the knowledge have been made due to the interview. It can also be concluded that Falcon 4 is a very realistic flight simulator. The only disadvantage is that the situational awareness of the pilot is lower because of the two dimensional representation of the environment. So in the end the interview turned out to be very important for my research project, not only because some assumptions that had been made turned out to be incorrect and which can now be corrected, but also because the collected knowledge has now been validated.

Appendix C

The XML definition of the knowledge base

C.1 The XML schema for the knowledge base

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="./KB"
    targetNamespace="./KB"
    xml:lang="en">

<xsd:annotation>
  <xsd:documentation>
  A schema for an XML file that describes a knowledge base for flying
  with an F16. The XML file will have to conform to the following
  hierarchy:
  - flight
    - situation
        - nextsituations
            - nextsituation
                  .
                  .
        - actions
            - phase
                - action
                    .
                    .
                .
                .
        - visual checks
            - instrumentgroup
                    - instrument
                        .
                        .
            - instrument
                .
```

```
                .
        - constraints
            - constraint
                .
                .
        - additionalrules
            - rule
                .
                .
        - situation
            .
            .
  </xsd:documentation>
</xsd:annotation>

<xsd:annotation>
  <xsd:documentation>
  The flight tag that contains an attribute with the name of the
  airplane that is described by the knowledge base and has situation
  tags as children.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="flight">
    <xsd:sequence>
        <xsd:element ref="situation" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="airplane" type="xsd:string" use="required" />
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The situation tag with an attribute that contains the name of the
  situation and child elements containing the situations that might
  occur directly after this situation, the list of time dependent
  and time independent actions, the list of visual checks and the
  list of constraints.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="situation">
    <xsd:sequence>
        <xsd:element ref="nextsituations" minOccurs="0" />
        <xsd:element ref="actions" minOccurs="0" />
        <xsd:element ref="visual checks" minOccurs="0" />
        <xsd:element ref="constraints" minOccurs="0" />
        <xsd:element ref="additionalrules" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="timewindow" type="xsd:integer" use="required" />
</xsd:element>
```

```
<xsd:annotation>
  <xsd:documentation>
  The element that contains all situations that might occur directly
  after the current situation. Situations are referenced by their names.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="nextsituations">
    <xsd:sequence>
        <xsd:element name="nextsituation" type="xsd:string" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The actions tag is the parent of a number of phase tags.
  The actions in the phases can be time dependent actions
  (which means that they have to be performed in the order
  in which they occur in the table) or time independent actions
  (they may be performed in any order). Time independent actions
  are grouped in a phase called "time independent".
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="actions">
    <xsd:sequence>
        <xsd:element ref="phase" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The phase tag is the parent of a number of action tags.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="phase">
    <xsd:sequence>
        <xsd:element ref="action" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string"/>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The action tag does not have any child tags, but does have three
  attributes. One containing the name of the control or instrument
  that action has an effect on. Another containing the priority
  value of the action. This value is a value between 0 and 1.
  And finally an attribute containing the fuzzy probability value
  of the action.
```

```
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="action" type="xsd:string">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="priority" type="priorityValue" use="required" />
    <xsd:attribute name="probability" type="probabilityValue" use="required" />
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The PriorityValue type is a float between 0 and 1.
  </xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="priorityValue">
    <xsd:restriction base="xsd:float">
        <xsd:minInclusive value="0"/>
        <xsd:maxInclusive value="1"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:annotation>
  <xsd:documentation>
  The probabilityValue must be one of the following values:
  VBP, BP, MP, SP, VSP. The value 1 is only used in rule elements.
  </xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="probabilityValue">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="1"/>
        <xsd:enumeration value="VBP"/>
        <xsd:enumeration value="BP"/>
        <xsd:enumeration value="MP"/>
        <xsd:enumeration value="SP"/>
        <xsd:enumeration value="VSP"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:annotation>
  <xsd:documentation>
  The visualChecks tag is the root tag for all instrument and instrumentgroup tags.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="visualChecks">
    <xsd:sequence>
        <xsd:element ref="instrumentgroup" minOccurs="0" maxOccurs="unbounded" />
        <xsd:element ref="instrument" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:element>
```

```
<xsd:annotation>
  <xsd:documentation>
  The instrumentgroup tag is a tag containing a list of instruments that the
  pilot should check consecutively during a situation. The instruments in a
  group should be placed close together in the cockpit.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="instrumentgroup">
    <xsd:sequence>
        <xsd:element ref="instrument" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The instrument tag is a tag describing an instrument that the pilot should
  check during a situation. It contains attributes containing the name of the
  instrument, a value describing whether or not the check is repetitive and
  the priority value of the check.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="instrument">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="repetitive" type="repetitiveValue" use="required" />
    <xsd:attribute name="priority" type="priorityValue" use="required" />
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The repetitive value of an instrument check can either be "yes" or "no".
  This is a kind of boolean value, but because we want the XML file to be
  easy to read, we define a new type here with the values "yes" and "no".
  </xsd:documentation>
<</xsd:annotation>
<xsd:simpleType name="repetitiveValue">
    <xsd:restriction base="xsd:string">
        <xsd:enumeration value="yes"/>
        <xsd:enumeration value="no"/>
    </xsd:restriction>
</xsd:simpleType>

<xsd:annotation>
  <xsd:documentation>
  The constraints tag is the parent tag for a set of constraint tags.
  It contains two attributes, the first contains the start
  probability, the second contains the end probability value.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="constraints">
```

```
    <xsd:sequence>
        <xsd:element ref="constraint" maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="endProbability" type="probabilityValue" use="required"/>
    <xsd:attribute name="startProbability" type="probabilityValue" use="required"/>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The constraint tag sets a condition on the value of a control or
  instrument for the start and/or the end of the situation. It
  contains one attribute with the name of the control or instrument
  and two optional attributes for the start and end value of the
  control or instrument.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="constraint" type="xsd:string">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="start" type="xsd:string"/>
    <xsd:attribute name="end" type="xsd:string"/>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The additional rules tag is the parent tag for a set of rule tags.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="additionalrules">
    <xsd:sequence>
        <xsd:element ref="rule" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The rule tag sets a condition on the value of a control or
  instrument for the start and/or the end of the situation. It
  contains one attribute with the name of the control or instrument,
  two optional attributes for the start and end value of the
  control or instrument and a probability for the start or end of a
  situation is the control or instrument has the given value.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="rule">
    <xsd:attribute name="name" type="xsd:string" use="required" />
    <xsd:attribute name="start" type="xsd:string"/>
    <xsd:attribute name="end" type="xsd:string"/>
    <xsd:attribute name="probability" type="probabilityValue"/>
</xsd:element>
```

```
</xsd:schema>
```

C.2 The flight plan in XML

C.2.1 The schema for a flight plan

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="flightplan"
  targetNamespace="flightplan"
  xml:lang="en">

<xsd:annotation>
  <xsd:documentation>
  This is a schema for a flightplan.
  </xsd:documentation>
</xsd:annotation>

<xsd:annotation>
  <xsd:documentation>
  The root tag containing all steerpoint tags.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="flightplan">
  <xsd:sequence>
    <xsd:element ref="steerpoint" maxOccurs="unbounded" />
  </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The steerpoint tag contains information about the steerpoint,
  like the coordinates of the steerpoint, the airspeed at which the
  pilot should fly to reach the steerpoint in time, the heading
  to fly to the steerpoint, the altitude at which the pilot should
  fly towards the steerpoint, the TOS (Time Over Steerpoint) which
  is the time at which the pilot should be over the steerpoint and
  the distance from the previous steerpoint. It also contains two
  attributes, one that says what type of steerpoint it is and one
  that contains the number of the steerpoint.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="steerpoint">
  <xsd:sequence>
    <xsd:element ref="coordinates" />
    <xsd:element name="airspeed" type="xsd:integer" minOccurs="0"/>
    <xsd:element name="heading" type="xsd:integer" minOccurs="0"/>
    <xsd:element name="altitude" type="xsd:integer" minOccurs="0"/>
    <xsd:element name="TOS">
```

```
      <xsd:restriction base="time">
        <xsd:pattern value="hh:mm:ss"/>
      </xsd:restriction>
    </xsd:element>
    <xsd:element name="distance" type="xsd:integer" minOccurs="0"/>
    <xsd:element name="action" minOccurs="0" maxOccurs="unbounded" type="xsd:string"/>
  </xsd:sequence>
  <xsd:attribute name="type" type="steerpointType"/>
  <xsd:attribute name="number" type="xsd:integer"/>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  The location of the steerpoint if given in longtitude and latitude coordinates.
  </xsd:documentation>
</xsd:annotation>
<xsd:element name="coordinates">
  <xsd:sequence>
    <xsd:element name="longtitude" type="xsd:float" />
    <xsd:element name="latitude" type="xsd:float" />
  </xsd:sequence>
</xsd:element>

<xsd:annotation>
  <xsd:documentation>
  A steerpoint can be one of the following types:
    - STPT: This is a normal steerpoint.
    - TGT: This is a steerpoint where a mission target
      is located.
    - IP: This steerpoint is the initial point for a
      pop-up attack.
  </xsd:documentation>
</xsd:annotation>
<xsd:simpleType name="steerpointType">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="STPT"/>
    <xsd:enumeration value="TGT"/>
    <xsd:enumeration value="IP"/>
  </xsd:restriction>
</xsd:simpleType>

</xsd:schema>
```

C.2.2 An example flight plan in XML

```
<?xml version="1.0" encoding="Unicode"?>

<!-- This is an example of a flight plan for an F16 fighter aircraft.
  This flight plan is NOT based on a real flight plan! -->
```

```
<flightplan xmlns="flightplan">

  <!-- Departure airfield -->
  <steerpoint type="STPT" number="1">
    <coordinates>
      <longtitude>-15.3435</longtitude>
      <latitude>28.366</latitude>
    </coordinates>
    <TOS>13:10:00</TOS>
    <action>Startup</action>
    <action>Taxiing to runway</action>
    <action>Taking off</action>
  </steerpoint>
  <!-- Navigational steerpoint -->
  <steerpoint type="STPT" number="2">
    <coordinates>
      <longtitude>-15.546</longtitude>
      <latitude>28.98</latitude>
    </coordinates>
    <airspeed>250</airspeed>
    <heading>160</heading>
    <altitude>2000</altitude>
    <TOS>13:28:30</TOS>
    <distance>30</distance>
  </steerpoint>
  <!-- Refueling steerpoint -->
  <steerpoint type="STPT" number="3">
    <coordinates>
      <longtitude>-14</longtitude>
      <latitude>27.194</latitude>
    </coordinates>
    <airspeed>300</airspeed>
    <heading>180</heading>
    <altitude>10000</altitude>
    <TOS>13:36:00</TOS>
    <distance>22</distance>
    <action>Air refueling</action>
  </steerpoint>
  <!-- Navigational steerpoint -->
  <steerpoint type="STPT" number="4">
    <coordinates>
      <longtitude>-15.59</longtitude>
      <latitude>27.685</latitude>
    </coordinates>
    <airspeed>280</airspeed>
    <heading>260</heading>
    <altitude>5000</altitude>
    <TOS>13:48:00</TOS>
```

```
    <distance>15</distance>
  </steerpoint>
  <!-- Initial point for a Visual attack -->
  <steerpoint type="IP" number="5">
    <coordinates>
      <longtitude>-16.587</longtitude>
      <latitude>28.343</latitude>
    </coordinates>
    <airspeed>320</airspeed>
    <heading>270</heading>
    <altitude>1500</altitude>
    <TOS>13:53:00</TOS>
    <distance>18</distance>
  </steerpoint>
  <!-- Target steerpoint for a visual attack -->
  <steerpoint type="TGT" number="6">
    <coordinates>
      <longtitude>-17.435</longtitude>
      <latitude>29.12</latitude>
    </coordinates>
    <airspeed>350</airspeed>
    <heading>310</heading>
    <altitude>2500</altitude>
    <TOS>13:55:25</TOS>
    <distance>5</distance>
    <action>Visual attack</action>
  </steerpoint>
  <!-- Navigational steerpoint -->
  <steerpoint type="STPT" number="7">
    <coordinates>
      <longtitude>-16</longtitude>
      <latitude>28.98</latitude>
    </coordinates>
    <airspeed>250</airspeed>
    <heading>270</heading>
    <altitude>2000</altitude>
    <TOS>14:05:10</TOS>
    <distance>35</distance>
  </steerpoint>
  <!-- Destination airfield -->
  <steerpoint type="STPT" number="8">
    <coordinates>
      <longtitude>-15.3435</longtitude>
      <latitude>28.366</latitude>
    </coordinates>
    <airspeed>200</airspeed>
    <heading>135</heading>
    <altitude>2000</altitude>
    <TOS>14:09:00</TOS>
```

```
    <distance>4</distance>
    <action>Landing</action>
    <action>Taxiing from runway</action>
    <action>Shutting down</action>
  </steerpoint>
</flightplan>
```

Appendix D

The controls and instruments

In this chapter all controls and instruments that are referred to in the knowledge base are described. Also for controls or instruments that can only have a limited amount of values those values are given. For every control or instrument first the position in the cockpit is given according to the layout in figure D.1. Then the name, the possible values and the function of that control or instrument is described. The values are usually given as a description of the state of the control or instrument in capital letters, with the exception of some controls that can have the value 0 or 1. In this case the 1 stands for a button that has been pressed and the 0 means that the button is released.

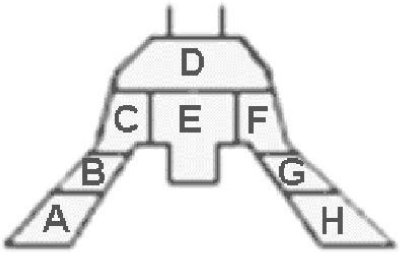

Figure D.1: The cockpit layout.

The locations of the controls or instruments are either the locations of the switches for those controls or instruments or the locations of the meters that display the value for that control or instrument. For example the power switch for the UFC panel is located at panel G, while the UFC panel itself is located on panel D. In these cases the location that is specified in front of the name of the control or instrument is the location of the switch while the location of the display of the control or instrument might be given in the description of the control or instrument. In the other cases the location is the location of the meter that displays the value or a "-" if the location of the switch or control is unknown or if there is no display or switch for the control or instrument on the layout in figure D.1. The following controls and instruments are referred to in the knowledge base:

(D) A-LOW: *1.* This is a button on the ICP with which the pilot can display the page with the minimum altitude on the DED.

(B) A/C lights (Anti Collision lights): *ON, OFF.*

(B) Air refuel door: *OPEN, CLOSED.* This door should be opened when the pilot is refueling in the air.

(G) Air source: *OFF, NORM, DUMP, RAM.* This controls the pressurization of the cockpit and the fuel tanks.

(E) Airspeed (in KIAS). This is the indicated airspeed in knots. This airspeed is also displayed in the HUD.

(E) Altitude (in AGL). This is the altitude from the ground. This can only be measured if the radar altimeter is switched on.

(-) AR (Air Refuel) disconnect button: *1.* This button can be pushed by the pilot to disconnect the refuel boom when there are problems during refueling.

(-) AR/NWS (Nose Wheel Steering/ Air refueling) light. This light is on when NWS is engaged or when the air refuel door is open. This light is located on a small vertical panel to the right of the HUD.

(F) ATT/FPM switch *ON, WIND CORR OFF, OFF.* This switch can be used to switch on the FPM.

(E) Attitude. The attitude of the airplane is displayed on the AI, the Attitude Indicator. The pilot can also determine the attitude using the HUD.

(F) Caution panel. This panel displays all the defects or problems of the airplane.

(C) Chaff/Flares *1.* This is an indication that the pilot has activated a release program for chaff or flares. The pilot can preprogram a release program to release a series of chaff and flares simultaneously.

(E) Climbing rate.

(D) DED (Data Entry Display). This display is located next to the ICP and displays information about steerpoints and minimum altitude among other things.

(-) Designate: *1.* The pilot can push this button to lock the radar to a target.

(-) DISC light. This light goes on when the tanker has disconnected the boom during air refueling. This is located on the same panel as the AR/NWS light to the right of the HUD.

(G) DL (Data Link): *OFF, ON.* The data link can be switched on if some data should be downloaded to the F-16. This can be done for example by an AWACS.

(B) ECM (Electronic Counter Measures): *OFF, STBY, ON.* The ECM pod can be used to jam enemy radars and radar guided missiles.

(C) Ejection seat: *ARMED, DISARMED.* If the ejection seat is disarmed ejecting is impossible.

(-) Elevator.

(A) Elevator trim.

(B) EPU (Emergency Power Unit): *ON, OFF.* The EPU should be switched on when the engines quit because some instruments will not receive any power when the engines fail.

(C) EWS (Electronic Warfare System). This is a group of controls and instruments that can be used to divert an attack.

(C) EWS JMR: *OFF, ON.* This switch should be set to ON to use the ECM jammer automatically.

(C) EWS CHAFF: *OFF, ON.* This switch should be set to on for the EWS to release chaff automatically.

(C) EWS FLARE: *OFF, ON.* Flares can be released automatically by the EWS only when this switch is set to ON.

(C) EWS PWR: *OFF, ON.*

(C) EWS mode: *OFF, MAN, STBY, AUTO, SEMI.* These are different levels of atomisation of the EWS. In SEMI or AUTO mode the EWS will release chaff and/or flares automatically.

(C) F-ACK: *1.* This is a button with which the pilot can turn the PFD (Pilot Fault Display) on and off and can cycle through the detected faults. The faults can also be displayed on the DED when the DED data switch is set to PFL (Pilot Fault List) data.

(G) FCC (Fire Control Computer): *ON, OFF.* The fire control computer calculates missile paths and help the pilot to execute attacks more accurately.

(F) FCR (Fire Control Radar):. The FCR is further described in the next section about the MFD's.

(-) Fire gun: *0, 1.* With this button the pilot can fire the A-G or A-A gun.

(D) FLIR (Forward Looking InfraRed): *ON, OFF.* The FLIR system enables the pilot to get an infrared picture of the surroundings.

(D) Fuel flow. This is the fuel flow that goes to the engine.

(B) Fuel pumps: *OFF, NORM, AFT, FWD.* With this switch the fuel tanks can be selected for which the fuel information will be displayed.

(G) GPS: *ON, OFF.*

(-) Ground speed.

(-) Hook: *RAISED, LOWERED.* The hook can be lowered when the pilot is landing on a carrier ship or when making an emergency landing.

(E) HSI (Horizontal Situation Indicator). The HSI displays the current and desired heading of the airplane and TACAN information if the TACAN has been switched on.

(D) HUD (Head Up Display): *ON, OFF.*

(D) ICP (Instrument Control Panel). This panel is located below the HUD and contains buttons with which the pilot can for example select steerpoints or display the F-ACK page on the DED.

(D) IFF (Identify Friend or Foe): *ON, OFF.* If this system is switched on it broadcasts a signal from which other airplanes can identify whether they are dealing with a friend or a foe. This system will be switched off when the aircraft enters enemy territory.

(G) INS (Inertial Navigation System): *ALIGN NORM, NAV, OFF.* This system computes the aircraft's position in latitude and longitude using the acceleration and deceleration of the airplane. The status of the INS can be checked in the DED.

(E) INSTR (Instrument) mode: *TCN, TCN-ILS, NAV, NAV-ILS.* This switch can be used to enable the TACAN and the ILS.

(C) Jettison stores: *1.* The jettison stores button can be pushed to jettison all stores when an emergency occurs.

(B) JFS (Jet Fuel Starter): *OFF, START1, START2.* This switch should be switched to START2 to start the engine.

(C) Landing gear: *RAISED, LOWERED.*

(C) Landing lights: *ON, OFF.*

(C) Laser arm: *LASER ARM, OFF.* The laser arm should be set to LASER ARM when executing an attack with laser guided bombs.

(-) Low speed warning: *ON, OFF.* The low speed warning horn will sound when a stall is imminent.

(C) Master arm: *OFF, MASTER ARM, SIM.* The weapons can only be fired when this switch is set to MASTER ARM.

(B) Master fuel: *ON, OFF.*

(B) Master lights: *NORM, OFF.*

(-) Master mode: *NAV, A-A, A-G, S-J, E-J, MSL OVRD, DGFT.* These are the possible modes for the FCC. NAV is the default mode and is selected automatically when no other mode is selected. A-A and A-G are combat modes that can be selected with buttons on the ICP. S-J (Stores Jettison) can be selected from the SMS page of the MFD and E-J (Emergency Jettison) is selected when the emergency jettison button is used. For DGFT (Dogfight) and MSL OVRD (Missile Override) there is a special switch on the stick with three states: OFF, DGFT and MSL OVRD. MSL OVRD is used for Medium Range Missiles like the AIM-120 and AIM-7. DGFT mode is used for Short Range missiles like the AIM-9 and it brings up the AIM-9 view as well as the EEGS view on the HUD.

(-) ML (Missile Launch) light: *ON, OFF.* This light will go on when the RWR detects a radar guided missile has been launched.

(B) MPO (Manual Pitch Override): *NORMAL, OVERRIDE.* This switched can be used to override the flight computer when recovering from a deep stall. Normally the flight computer locks all controls when the airplane is in a deep stall and shuts the pilot out of the control loop.

(D) NWS (Nose Wheel Steering): *ON, OFF.* Nose wheel steering can be switched on for taxiing.

(C) Parachute: *1.* The parachute can be deployed in an emergency if the airplane goes to fast.

(C) Parking brakes: *ON, OFF.*

(-) Pickle: *0, 1.* The pickle button can be depressed to give the FCR permission to release the bombs.

(-) Pitch.

(F) RALT (Radar Altimeter): *OFF, STBY, ON.* The radar altimeter calculates the altitude from the ground.

(B) Radio: *0-7.* The radio can be set to different channels. These radio modes 0 to 7 have the following meanings: Off, to the entire flight, to a package, to and from a package, in the proximity (40 nm), guard, broadcast, to the tower set in the TACAN channel.

(-) RDY light: *ON, OFF.* The ready light goes on during air refueling when a tanker gives a clearance for contact. It is located on the vertical panel to the right of the HUD.

(-) Roll.

(-) Rudder.

(A) Rudder trim.

(D) RWR (Radar Warning Receiver): *ON, OFF.* If the RWR is turned ON it will sound a warning when a launched radar guided missile is detected.

(-) Slew: *1.* This is an indication that the pilot is slewing the radar cursors to a target.

(C) Speed brakes. The speed brakes can have a value between 0 and 1.

(D) Steerpoint type: *STPT, IP, TGT.* This is the type of the steerpoint that has been selected. This is displayed on the steerpoint page of the DED.

(-) Sighting option: *VIP, VRP.* These options can be chosen when performing a CCRP or LADD attack.

(B) TACAN function: *TR, AA-TR.* This switch is used for specifying the kind of TACAN channel that has been set. This is either a channel on the ground (TR) or one in the air (AA-TR, e.g. a tanker aircraft).

(D) Throttle: *IDLE, MIL, MAX AB.* The throttle stick is located besides the left knee of the pilot. The RPM display is located on panel D.

(C) Throttle idle detent: *1.* This idle detent should be switched to enable the throttle stick to be moved below the idle position which will cause the fuel flow to the engine to stop.

(D) TWS (Threat Warning System). This is a panel on which the ML light is located as well as a scope that can display radar threats in the area.

(C) U/C. This is a panel located to the left of the ICP in the cockpit with a light for every part of the landing gear that indicates the status of the landing gear.

(G) UFC (Up Front Controls): *ON, OFF.* These controls are all the controls on the UFC panel. This panel is located right below the HUD on panel D and contains the ICP and the DED among other things.

(-) Wheel brakes: *ON, OFF.*

D.1 The Multi Function Displays

In the cockpit of the F-16 are two displays called Multi Function Displays (MFD's) that show information about the stores on the F-16 and the radar image among other things. It is with these two displays that the pilot can choose which kind of radar and which weapon to use. There are a lot of different pages that can be displayed on the MFD's but the most important ones are given below.

OFF.

Main. The main page is the starting point for navigating to other pages.

HSD (Horizontal Situation Display). The HSD displays a radar image of the surrounding area.

FCR (Fire Control Radar):

- OFF.
- STBY.
- RWS (Range While Searching). This radar mode is used for targeting with the AIM-120 and AIM-7 armament.
- VS (Velocity Search). This radar modes shows the velocity with which other aircraft are closing in on your position.
- TWS (Track While Scan). With TWS multiple targets can be tracked simultaneously.
- ACM (Air Combat Manoeuvring). The ACM radar modes are used to point weapons.
  - NO RAD (Not Radiating): This is the default mode in which the radar is not radiating any radar beams.
  - HUD scan. With the HUD scan targets that are visible in the HUD can be locked.
  - Vertical scan. The radar beam is sweeped up and down in a vertical line.
  - SLW (Slewable scan). In this mode the radar pattern can be "slewed" (moved). This is the only ACM mode that can be used to lock up targets that can not be seen.
  - BSGT (Boresight scan). In this mode the radar beam is pointed straight out of the nose of the jet.
- GM (Ground Map). In GM mode the radar will see man made objects like buildings and bridges. The submodes of this mode are for adjusting the detail and magnification.
  - NORM (Normal)
  - EXP (Expand)

  - DBS1
  - DBS2
- GMT (Ground Moving Target). In GMT mode the radar will only detect moving objects like tanks and trucks.
  - NORM
  - EXP
- SEA (Sea mode). The SEA mode is used for detecting ships at sea.
  - NORM
  - EXP

SMS (Stores Management Systems) The SMS can be used to display information about the weapons and stores on board of the aircraft and to make a selection of those weapons.

- AAM (Air-to-Air Missile) page. On this page one of the AAM's can be chosen. The following missiles are examples of possible AAM's:
  - AIM-120 (AMRAAM). This is a radar guided, medium range missile.
  - AIM-9x: The AIM-9 is a heat seeking missile. The AIM-9P missile can only see the heat signature of an airplane if it is behind that airplane. The AIM-9M can see the heat signature from any direction.
  - AIM-7 (Sparrow). This missile is like the AIM-120, but unlike the AIM-120 the fighter has to stay locked on the target until the moment of impact.
  - A-A Gun (Vulcan cannon). There are several modes for the A-A gun:
    * EEGS (Enhanced Envelope Gun Sight). This mode will predict the target's position one bullet's time in the future.
    * LCOS (Lead Computing Optical Sight). This mode is like the EEGS but does not provide any prediction.
    * SNAP (Snapshoot Sight). In SNAP mode the HUD will display a snapshoot line that indicates where the gun has been pointing.
- AGM (Air-to-Ground Missile) page. What this page displays depends on the selected A-G missile. The following weapons are examples of AGM's:
  - Maverick. When the Maverick is selected the MFD displays an electro-optical image recorded by the missile's seeker head.
    * SLAVE. In the SLAVE submode the missile's seeker head is linked to the radar. This means the seeker head is always aimed at the object that has been locked by the radar.
    * BSGT (Boresight). In this submode the missile's seeker head is independent of the radar.
  - LGB (Laser Guided Bomb). When LGB is selected the MFD displays an electro optical image recorded by the missile's seeker head, just like when the Maverick is selected. For the LGB there are also the two submodes SLAVE and BSGT.
    * SLAVE
    * BSGT (Boresight)

  - HTS-HARM (HARM Targeting System, High-speed Anti Radiation Missile). The HARM missile has its own radar display, the HTS. The HARM missile can be fired at radar emitting targets at any direction from the aircraft. A HARM can for example be fired at a SAM site located behind the F-16.
  - TARS (Tactical Aerial Reconnaissance System). The TARS can be used for gathering intelligence.
  - A-G bombs page. This page can be used to select one of the available A-G bombs. This page can also be used to select whether the bombs should be dropped singly or in pairs.
    * CCRP (Continuously Computed Release Point).
      · sighting option: VRP, VIP.
    * CCIP (Continuously Computed Impact Point).
    * DTOS (Dive Toss)
    * RCKT (A-G Rocket)
    * LADD (Low Altitude Drogue Delivery)
      · sighting option: VRP, VIP.
  - A-G Gun page.
- INV (Inventory) page. On this page an overview of everything that is loaded on the F-16 is displayed.
- S-J (Selective Jettison) page. This page look like the INV page, but from this page stores can be selected to be jettisoned.

Appendix E

Glossary

A-A: Air-to-Air.

A-G: Air-to-Ground.

A-LOW: Automatic Altitude Low Warning.

A/C lights: Anti Collision Lights.

AAM: Air-to-Air Missile.

AB: Afterburner.

ACE: Adaptive Cockpit Environment.

ACM: Air Combat Manoeuvring / Air Combat Mode.

AGL: Air Ground Level.

AGM: Air-to-Ground Missile.

AIM: Air Intercept Missile.

AMRAAM: Advanced Medium Range Air-to-Air Missile.

AOA: Angle Of Attack.

AR: Air refueling.

AWACS: Airborne Warning And Control System.

BCN: Beacon mode.

BSGT: Boresight.

CCIP: Continuously Computed Impact Point.

CCRP: Continuously Computed Release Point.

DED: Data Entry Display.

DGFT: Dogfight.

DISC light: Disconnect light.

DL: Data Link.

DTOS: Dive Toss.

E-J: Emergency Jettison.

ECM: Electronic Counter Measures.

EEGS: Enhanced Envelope Gun Sight.

EPU: Emergency Power Unit.

EWS: Electronic Warfare System.

F-ACK: Fault Acknowledgment.

FCC: Fire Control Radar.

FCR: Fire Control Radar.

FLCS: Flight Control System.

FLIR: Forward Looking Infra Red.

FPM: Flight Path Marker.

FTT: Fixed Target Track.

GBU: Guided Bomb Unit.

GM: Ground Map.

GMT: Ground Moving Target.

GPS: Global Positioning System.

HARM: High-speed Anti Radiation Missile.

HSD: Horizontal Situation Display.

HSI: Horizontal Situation Indicator.

HTS: HARM Targeting System.

HUD: Head Up Display.

ICP: Instrument Control Panel.

IFF: Identify Friend or Foe.

ILS: Instrument Landing System.

INS: Inertial Navigation System.

INSTR mode: Instrument mode.

IP: Initial Point.

JFS: Jet Fuel Starter.

JMR: Jammer.

KCAS: Knots Calibrated Airspeed.

KIAS: Knots Indicated Airspeed.

KTAS: Knots True Airspeed.

LADD: Low Altitude Drogue Delivery.

LCOS: Lead Computed Optical Sight.

LGB: Laser Guided Bomb.

MFD: Multi Function Display.

MIL: Military Power.

ML: Missile Launch.

MLU: Mid Life Update.

MPO: Manual Pitch Override.

MSL: Mean Sea Level / Missile.

MSL OVRD: Missile Override.

NWS: Nose Wheel Steering.

PFD: Pilot Fault Display.

PFL: Pilot Fault List.

RP: Release point.

RALT: Radar Altimeter.

RCKT: Air-to-Ground Rocket.

RDY light: Ready light.

RWR: Radar Warning Receiver.

RWS: Range While Search.

S-J: Stores Jettison.

SLW: Slewable mode.

SMS: Stores Management System.

SNAP: Snapshoot Sight.

STBY: Standby

STPT: Steerpoint.

TACAN: Tactical Air Navigation.

TARS: Tactical Aerial Reconnaissance System.

TCN: see TACAN.

TD: Target Designator.

TGT: Target.

TWS: Track While Scan / Threat Warning System.

UFC: Up Front Controls.

VIP: Visual Initial Point.

VRP: Visual Release Point.

VS: Velocity Search.

Appendix F

Paper

Situation recognition as a step to an intelligent situation-aware crew assistant system

Quint Mouthaan Patrick Ehlert Leon Rothkrantz

Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands

Abstract

In this paper we will present a system that can recognize situations during a flight in real-time based on the data from an aircraft's systems. The system uses Bayesian belief networks to calculate the probabilities of both the start and end of possible situations and from this distillates the most probable situation. The situation recognizer system is part of our test environment to create better human-machine interfaces in the cockpit.

1. Introduction

1.1. Problem statement

Anyone who has seen the cockpit of an F-16 aircraft knows that it is stuffed with control buttons, meters, and displays. The meters and displays provide the pilot with a wealth of information during a flight. Since the F-16 is capable of speeds of over 2000 Km/h, pilots have very little time to process the large amount of available information and make decisions. To help a pilot deal with information processing and decision-making, avoid information overload, and optimise flight performance, a Crew-Assistant System (CAS) or intelligent pilot-vehicle interface (PVI) has been proposed [1,2]. A typical CAS is shown in Figure 1. The idea is that the system presents relevant information to the pilot at the right moment and in the appropriate format, depending on the situation, the status of the aircraft, and the workload of the pilot. It is even possible that

the CAS takes over (simple) tasks. Not only military pilots can benefit from such a system, it is useful for commercial pilots as well.

1.2. The ICE project

The Intelligent Cockpit Environment (ICE) project is a project of the Knowledge Based Systems group of Delft University of Technology. The goal of the ICE project is to design, test, and evaluate computational

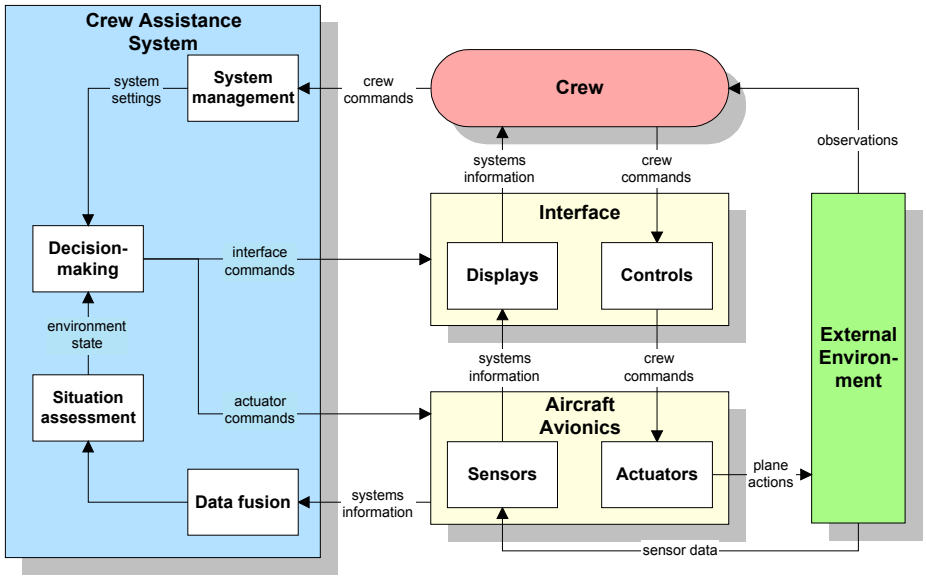

Figure 1: A generic crew assistance system architecture

techniques that can be used in the development of intelligent situation-aware CASs. Using methods from artificial intelligence, ICE focuses primarily on the data fusion, data processing and reasoning part of these systems. Special issues addressed in the ICE project are situation recognition, mission or flight plan monitoring, pilot workload monitoring, and attack management [4].

In this paper we will discuss a system for real-time situation recognition that we developed, which is the most important subsystem in the situation assessment module shown in Figure 1. First, we will describe the design of the recognition system. Then we will discuss an example scenario that was used to test our system. Finally, we will draw some conclusions about the performance of the system.

2. The design

2.1. System architecture

The goal of our system is to derive the current situation in real-time from available aircraft data. The system receives information from the airplane's sensors about the state of the airplane (e.g. the airspeed), the actions of the pilot (e.g. lowering the landing gear), and the environment (e.g. a missile that has been launched). Our system will use all this information to determine which of all possible situations is occurring. This list of possible situations is defined in an XML file. For every situation, the actions the pilot might perform and typical situation-related events are defined. These events can either be changes in the state of the airplane or changes in the environment. During a flight the system will compare the information it receives from the sensors with the stored situations data and it will try to determine which situation is occurring. The architecture of the system is depicted in Figure 2. We will now explain in short the elements of the system.

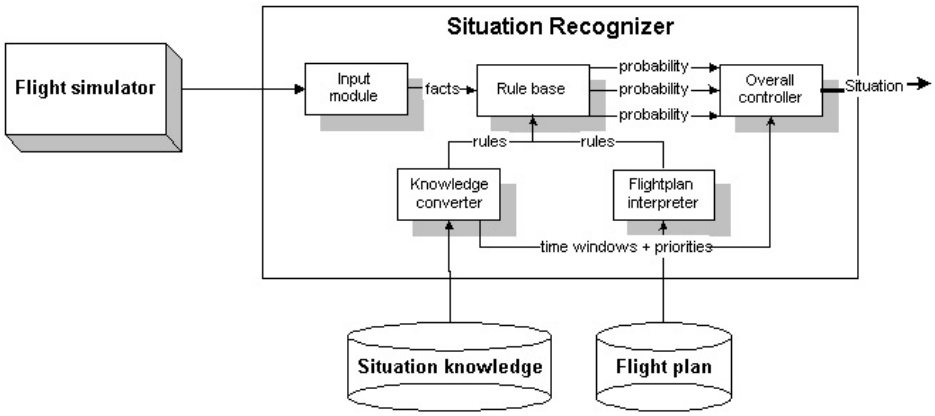

Figure 2: The architecture of the situation recognition system.

The input module receives aircraft data such as airspeed, altitude, throttle, etc. from the flight simulator and converts this data to facts that are forwarded to the rule base.

The knowledge converter converts all the situations knowledge to IF-THEN rules and puts them in the rule base.

The flight plan interpreter converts the information in the flight plan to a number of rules that are put in the rule base. These are rules that predict which situations will occur in the near future.

The rule base contains all the rules that have been generated from the earlier described modules. When data from the flight simulator is added to the rule base in the form of a number of facts, some of the rules will fire and will generate probabilities concerning the start or end of a situation. These probabilities are passed to the overall controller.

The overall controller gets all probabilities from the rules in the rule base and some extra information about the situations. The overall controller combines the probabilities and calculates for every situation the probability that it has started and the probability that it has ended. Based on the resulting probabilities, the overall controller will draw a conclusion about the situation that is most likely to be the current one. Calculating the probabilities will be done using Bayesian belief networks, which will be discussed in the next section. More information about the recognition system can be found in [5].

2.2. Probability inference using Bayesian belief networks

We want to calculate two probabilities for every situation: the probability that the situation has started and that it has ended. Therefore we need two Bayesian belief networks. The two networks are explained in the following sections.

2.2.1. The start probability calculator

Figure 3 shows the Bayesian belief network that is used to calculate the probability that a situation has started.

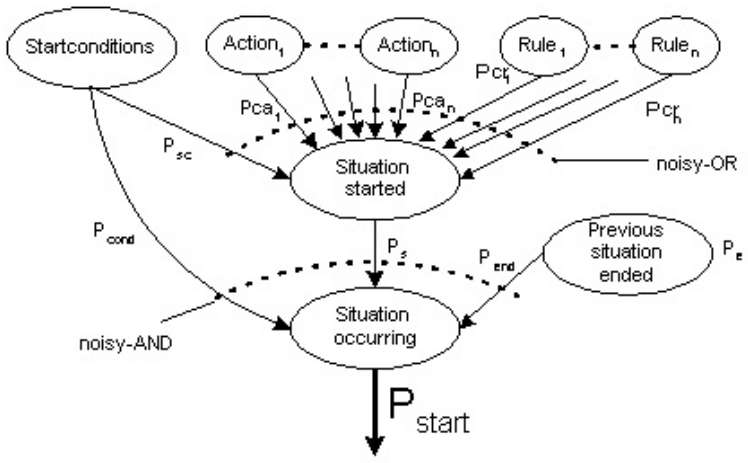

Figure 3: The Bayesian belief network that calculates the probability that a situation has started.

The start conditions for a situation are conditions that must be satisfied before a situation can possibly have started. When the start conditions are satisfied the probability of the start constraints that is specified in the situations knowledge will be the output of this node.

The action probabilities are passed to the Bayesian belief network by the rules in the rule base. These probabilities all contribute to the probability that the situation is occurring.

The additional rules are rules that fire when the state of the aircraft changes or when a specific event happens. When they fire they can generate a probability that a situation has started or ended.

The probability calculator combines the probabilities of the nodes that have been described above using the noisy-OR model.

The previous situation influences the start probability of a situation because the probability that a situation is occurring should rise when the probability that one of the previous situations has ended becomes higher.

Based on this Bayesian belief network the probability that a situation has started and is occurring can be calculated with the following formula:

$$P_{start} = P_{cond} * P_{end} * P_s$$

$$P_{cond} = \begin{cases} 0 & if\ P_{sc} = 0 \\ 1 & if\ P_{sc} > 0 \end{cases}$$

$$P_s = 1 - ((1 - P_{sc}) * \prod_{i=1}^{n} (1 - Pca_i) * \prod_{j=1}^{n} (1 - Pcr_j))$$

In this formula is P_{sc} the probability of the start conditions, P_{end} is the probability that one of the previous situations has ended, Pca_i is the probability of the i-th action that should be performed during the situation and Pcr_j is the probability of the j-th event or change in state that can occur during the situation.

2.2.2. The end probability calculator

In Figure 4 the Bayesian belief network to calculate the probability that the situation has ended is shown. In this Bayesian belief network we see a lot of the same nodes as in the belief network for the start of the situation. The nodes that are different are discussed below.

The time window for a situation is the maximum duration of that situation. If the start of a situation has been detected the probability that it has ended should become larger when a bigger part of the time window has elapsed. This is implemented with the time window node.

The situation started node produces a 1 if the situation has started and a 0 if the situation has not started yet. This node is necessary because we only want to calculate the probability that the situation has ended if the situation has started.

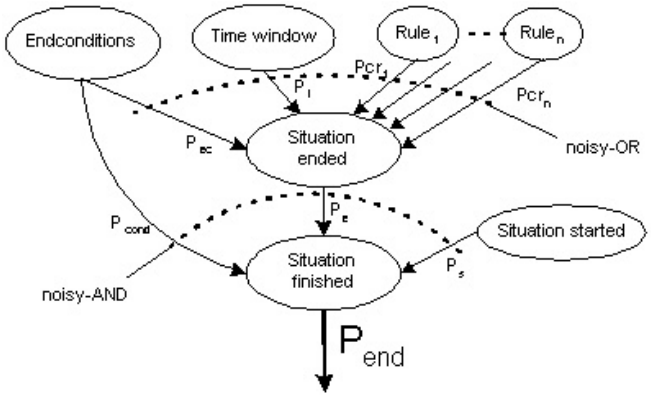

Figure 4: The Bayesian belief network that calculates the probability that a situation has ended.

The probability that the situation is not occurring can be calculated with the following formula:

$$P_{end} = 1 - ((1 - P_{sc}) * (1 - P_t) * \prod_{i=1}^{n} (1 - Pcr_i))$$

3. Experiments and results

The system was tested using the Microsoft Flight Simulator. Experiments were performed with an F-16 and with a Cessna. The results of one of our experiments are presented in Table 1. The first column contains the names of the situations that occurred and/or were detected. The second column contains the times at which we considered the situations to be started. The third column contains the times at which the situations were detected by the system. The times are given in seconds from the moment the program was started. The particular mission of Table 1 consisted of an attack on a ground target in an F-16. During the flight to the target the pilot had to check his course twice. After the attack had been performed the pilot returned to the airbase and landed the airplane.

Table 1: The test results

Situation	Time started (s)	Time detected (s)
Startup	0	0
Taxiing to runway	10	12
Taking off	14	15
Normal flight	43	34
Navigating	83	83

Normal flight	91	91
Navigating	123	123
Normal flight	128	128
Visual attack	186	193
Normal flight	221	221
Landing	361	366
Aborting a landing	-	410
Taxiing from runway	409	410
Shutdown	427	427

From the table it is clear that the program is able to recognize most situations in a matter of seconds. The reason that it sometimes takes the system some time (5 seconds or more) to detect some of the situations is mainly because the start and end points of those situations are very vague. For example, we have considered the situation *Normal flight* to start at the moment at which the pilot levelled the airplane, while the program considered the situation to be started when the pilot reached a particular minimum altitude. This could be solved by adjusting the situations knowledge to change the definition of the start of a landing of the system.

The landing was a normal landing and was detected, but as is shown in the table the program thought for a moment that the landing was being aborted. This happened when the program knew that the landing had been finished and looked for the situation with the highest start probability. This turned out to be the situation *Aborting landing*. This is because at some point the pilot had moved the throttle to the maximum. This action was still in the memory of the program when the landing ended. The fact that the landing gear was raised at the start of the landing was also still in the memory of the program. Because of this the probability that the landing was being aborted was high and the situation *Aborting landing* became the current one once the landing had finished. However as soon as that happened the program realized that the airplane was actually taxiing and it corrected the mistake immediately.

4. Conclusions and future work

We have created a system that can recognize the current situation during a flight with an F16 and with a Cessna. The system is based on a probabilistic model. A rule base was created that compared data from a flight simulator with the situations knowledge defined in an XML file. The rules in the rule base generate a number of probabilities that are combined using Bayesian belief networks to calculate the probabilities

that the situations are occurring. Based on these probabilities a conclusion is drawn about the situation that is most likely to be occurring.

The program was tested on a number of test scenarios. After analysing the results we can conclude that the system performs really well. It makes few mistakes and is able to correct them. Furthermore it is able to come to a conclusion about the current situation in real-time.

We can conclude that the program would perform even better if it would be extended to a causal model. This causal model could take into account the relations between actions and events.

Future work will consist of adding causality to make the system more reliable. Once that has been done the system can be used in conjunction with a workload assessment module to create a more complete situation awareness module. This situation awareness module can be used in an intelligent cockpit system that monitors and supports the pilot during a flight.

References

[1] Banks, S.B and Lizza, C.S. (1991) "Pilot's Associate: a cooperative, knowledge-based system application", in *IEEE Intelligent Systems, Vol. 6*, No. 3, pp. 18-29, June 1991.

[2] Onken, R. (1997) "*The cockpit assistant system CASSY as an on-board player in the ATM environment*", paper presented at 1st USA/Europe Air Traffic Management R&D Seminar, Saclay, France, June 1997.

[3] Endsley, M.R. (1995) "Towards a theory of situation awareness in dynamic systems", in *Human Factors, Vol. 37*, No. 1, pp 32-64.

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

[5] Mouthaan, Q.M. (2003) "*Towards an intelligent cockpit environment: a probabilistic approach to situation recognition in an F-16*", MSc. thesis, Knowledge Based Systems, Delft University of Technology, The Netherlands.