Automatic Suspicious Behaviour Detection

A distributed approach to multi camera car surveillance

ing. G.D. Eigenraam

TU Delft

Automatic Suspicious Behaviour Detection

A distributed approach to multi camera car surveillance

by

ing. G.D. Eigenraam

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	1524798	
Completed in:	June, 2016	
Thesis committee:	Prof. drs. dr. L.J.M. Rothkrantz,	TU Delft, supervisor
	Dr. J. Broekens,	TU Delft
	Dr. M. M. de Weerdt,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

Anyone who is familiar with rowing in the Netherlands is probably aware of the famous "Ringvaart". This epic rowing tour is different from any other rowing event, simply by its sheer size. During the tour, the rowers must travel a distance of 100 kilometres through the Randstad. With an average overall time of 11 hours, the tour makes both a physical as well as a mental impact. Because of the impact, each competitor will face him- or herself at some point in the tour. Every participant who signs up for this tour is most likely out of his/her mind. Needless to say, I rowed it twice.
The truth is that this tour is a brilliant experience. To me, the beauty is that it is a team effort. Every team member goes through the same struggle, pain and exhaustion. The thought of quitting will continuously enter your mind, but you just have to keep rowing. For yourself and for everyone in the team. For most people, it isn't a competition, you either finish a victor or stop a loser.
Each year after the second tour I considered rowing my third tour. It is the craziest thing that during the tour you always think to yourself "why did I join up for this?" and yet a week after the tour, the "never-again" starts to change into a "maybe next year". Some time during the thesis it hit me: this thesis is my third Ringvaart. The scala of emotions one experiences during this tour, were all present in this project. The same holds for the role of friends and family. Without their support, distractions and advise I would have never been able to finish this journey. Before we more the focus to the research itself, I want to take the time to acknowledge the people that have helped me through this memorial time. I wanted to hold onto the analogy of the Ringvaart to show why the each and everyone of the people on this list played a significant role. Come with me on a journey through the Ringvaart.

0 - 10 kilometre

The first 10 kilometres of the Ringvaart are the easiest of them all. I remember the first time I rowed the tour and after 10 kilometres I found myself thinking "Ow, this is going to be easy". I ended that same tour with more blisters on my hands than I could count.
These first kilometres set the tone for the tour. It is important to start off with a good spirit in the boat and having a laugh here and there. As it was in the Ringvaart, such was the same for studying in Delft. It is important to form a good group of friends around you. For the most part, I was set once I met the great fellows of "De Welvaart BV" (DWBV). They are a group of well-spirit, up for anything friends and at the beginning of our studies in Delft, we had all the time in the world. Christmas dinners, random trips and of course lots of "meetings" about the future of DWBV. I've always been amazed by how each and everyone of the members were such great guys. With no exception, you could gather any combination and have a great time. Although I would love to talk about each member individually, I want to take to time to thank three "Welvarende" in particular. First off, I would like to thank Siem Kok for all the great times, but also his patience. Siem is an incredibly smart, disciplined and hard working guy that was unlucky enough to end up in a team with me. The famous facebook post reading "Sorry Siem" has become the leading image to represent this friction. The picture showing two aspirin, was posted on the morning Siem and I scheduled to row early in the morning. And as Siem had changed into his sporting outfit and was ready to go, this image was posted from my bed after an evening of way too much fun on the rowing club. But despite all that, we ended up being the best of friends.
The last member of DWBV I would like to personally thank is Wietse Bosch. No matter what weird idea I came up with, Wietse would always be the first in line to try it out. Running, cycling, to name a few. He was always in for drinks or the next adventure. The latest of which is the boardgame madness. I want to thank him for the great times, which hopefully will continue in the same pace as before the graduation.
Overall I would like to thank the members of DWBV for the great times I had with them and wih them a prosparus future. DWBV may officially have ended, but technically it will never end. After all, that DWBV is, is defined by its members.

10 - 18 kilometer

After 10 kilometres in the race, you will to face you first challenges. You will notice small bumps in the railing of your seats or ache in your back. Quickly, your training kicks in and you start reminding yourself of the proper techniques. Oddly enough, the core technique is taught in the first couple of weeks of training. You find yourself remembering how it all started.

My foundations in the field of computer science was laid at the Hogeschool Windesheim in Zwolle. I had the honor of meeting some great people that, with whom I'm still in contact with today. When I met Vincent de Vries, he already had nice long history of studying and was 29 when he graduated. I remember joking about that at the time. Now, a few year later, I'm standing here 30 years old, not joking anymore. I guess it is clear who has the last laugh. The group of us that stayed in touch after our time in Zwolle had to bear with me the last few years whenever I told them why I was still working on the master thesis. I would like to explicitly thank Vincent de Vries and Stephane Rouault for their support and friendship during this thesis. Particularly for the patients the last year, while I was slowing down some great initiatives become I was either too busy or without any money. I promiss to make it worth the wait.

18 - 26 kilometer

Going back to our epic tour, we past the 18 kilometre mark and starting to become tiered. You start telling yourself to stop wining and push through. Thankfully, I had friends around me that turned the focus to the positive and reminded me to have fun. Although I have met many great people during my time in Delft, a few of them in particular come to mind.

Hans-Peter van den Heuvel definitely has a talent for barging in and force you to get away from the computer once in a while. The Technical University of Delft produces has the brilliant skill of producing some unique characters. Hans-Peter is maybe the greatest of them all. This brilliant mind maybe hidden away behind his boorish exterior, but never ceases to amaze. My thanks go out to Hans-Peter for enriching my life with his well-needed distractions and brilliant stories.

There is one person I particularly want to thank in this period during the thesis. Halfway through the thesis, I suddenly found myself without a job and struggling to manage my finances. Rolf van Rijn was the one who answered to call and introduced me to Allseas Engineering BV, which started the most exiting job I have ever had. Before I started working here, I never considered myself working in the offshore business. But it turned out to be an unforgettable working experience and all thanks to one man that listened to my request on a barbecue.

26 - 34 kilometer

Moving on, you suddenly feel a pain erupting from one of your fingers and it turns out the first blister on your finger has popped. The rain a while back, made the handle of the oar wet and the friction did its work. Blisters start appearing all over your hands and from here on, you will just have to suck it up and plan your moments to regenerate.

Thankfully during my thesis, I have a great group of friends in Amersfoort that helped me gather my strength. They kept supporting me, gave me advise or simply listened when I needed to fend some frustrations. Special thanks go out to Menno Amoraal, Jan Huijser and Akke van Veen for being such great hosts every time I was in Amersfoort and great times we had during these evenings. If it wasn't for these moments, I most likely would never have made it this far.

34 - 42 kilometer

After a third of the way, you are committed. There is no way back now, everything else is going to have to wait until this tour has reached its conclusion. Everything around you now is going to be the status quo for the next 60+ kilometres.

Throughout my whole master, I lived in the same house. Room mates play an important role in the day to day life. And we had a lot of great times together the past years. The nightly games of Risk with Thomas Döbkens et al., the sailing trips with Cees Biesheuvel and the gym sessions with Jesse van Meulen. The all made my time in Delft fantastic and unforgettable. I would like to thank all of my room mates from the Fuutlaan for making this possible.

42 - 63 kilometer

We fast forward to 42 kilometres in the tour. You and everyone around you understand the state you are in and what situation you are in. You just have to keep on rowing, even though you may be frustrated and tiered. At this point you just want people to be there for you, too make you laugh when you are sad, listen when you are stressed and come around with pleasant distractions once in a while.
The people that sympathized with me through the whole thesis were my brothers and sister. They stuck with me through it all and were there for me through think and thin. I am truly blessed with great family. I want to thank all of them, Peter, Anniek, Tim & Erwin, for there support during the thesis.

63 - 75 kilometer

Around 60+ kilometres is where it really becomes tough. You start to feel the emotional and physical weight of the tour taking its toll. Taking breaks seem to have no effect; you're in the exact same state when getting back in the boat as you were getting out of it 15 minutes earlier. Only a few things help at this point and one of them is the words of your coach. The person who has been there for you from day 1 of the training.
Leon Rothkrantz has been my mentor throughout this thesis. Despite his retirement, he is still going strong. Still up to date on everything; I never even bested him once in knowledge during the thesis. I want to thank him for all his involvement in the research and the fascinating stories about all the other projects he has been involved in. I am glade that by finishing this project, I am allowing him to spend more time on his retirement. Even though that will most likely mean the next project. Some people simply refuse to slow down.

75 - 88 kilometer

Moving past the 75 kilometres mark, tricks or distractions no longer work. You are totally stripped from all non-essential parts. It is just you, your training and the people that are closest to you. Distance signs seem to crawl by, everything hurts and all you can think is please let this be over. This is the moment you will find out who you truly are. It is in these moments, you find out how lucky you are with the people around you.
No one has supported me more during this part of the thesis than my parents. Never have my parent gave up on me; not once did they stop believing I could do this. They always wanted the best for me and helped whenever I needed it. Peter and Ineke Eigenraam, I want to thank you for everything you have done for me during and before this master thesis. I can't wait to start working and make the best of the opportunity that you have given me.

88 - 100 kilometer

An interesting aspect of the Ringvaart is that for the rowers there are mentally 2 finishes. The ultimate finish is of course Delft where the tour is done. But about 12 kilometres earlier, there is another finish. The sluice in Leidschendam forces the participants to get out and carry the boat to the other side. Due to the chaos this creates, every team gets about 45 minutes to make this happen. More than enough time to find a place for your boat, get some refreshments and seal up the wounds. Most participants consider these last twelve kilometres easy, because of the refreshing break.
Living in Delft adds another reason for this second finish. Most friends that were not able to cheer for you the whole way, decide to meet up in Leidschendam to congratulate you and cheer the remainder of the tour. The refreshing break and cheering friends make this remainder of the tour the easiest part of them all.
I would like to used this last section of the tour as a metaphor for everyone who was rooting for me during the thesis. A big thanks goes out to everyone who was there for me with small words of encouragement, best wishes and other methods to keep the spirit up. I cannot wait for the next phase in my life and enjoy all the things we have had to put on hold for a while.

ing. G.D. Eigenraam
Delft, Juni 2016

Abstract

Multi camera surveillance systems is a hot topic in computer science, artificial intelligence and security management in general. While cameras used to be low resolution analogue devices, nowadays they stream high definition images and come equipped with the processing power to reason about their environment. The field covers a wide variaty of researches from streaming large amount of data to reasoning about the behaviour. In this thesis we will address the problem of cooperation and task delegation within a distributed 3rd generation visual surveillance system. Multiple intelligent cameras will work together to solve the problem of detecting, tracking and analysing the behaviour of cars in a closed environment. As video surveillance systems grow more complex, the computational power distribution and data communication become a more important issue. Traditional centralized systems are no longer cope with the scale of processed data and inventive system designs are required to efficiently analyse the images at real time. The proposed system takes an hierarchical agent based approach to visual surveillance and distributes tasks over multiple intelligent cameras, while aiming for an equal distribution of computational demand of each camera. The tasks delegation applies an hierarchical model of agents and performs the analysis on the first layer of agents, which has all the required data available. The bottleneck of processing power demand and data stream within the global process is avoided by reducing the information resolution for each layer. The design concept of the hierarchical multi-agent approach to visual surveillance is tested in a simulated environment, which will only indicate the potential of object detection and tracking in real life environments. The main contribution of the research is to show the potential of applying the task delegation design method in distributed visual surveillance systems to achieve the scalability demand.

Contents

1 Introduction

The research performed in "Automatic Suspicious Behaviour Detection, a distributed approach to multi camera car surveillance" builds the distributed reasoning model for the visual surveillance system of the officer training facility of the Royal Dutch Navy in Den Helder. Previous work focused on the detection, recognition & mapping of objects in camera images [12, 70]. The mostly centralized approach however showed limited scalability. Decentralized approaches could increase the scalability, but lack the global overview needed for behaviour analysis. In this reseach, we investigates the possibility of cooperative intelligent cameras to detect specific behaviour reaching over multiple camera views.
The introduction chapter sets the scope for the research and explain the used approach. After a short introduction to the topic and background from the point of view of the research group, the scope of this research is described using the problem definition, research questions and scientific challenges. Knowing the scope, we list the method applied to reach these goals and finally the report structure of this thesis.

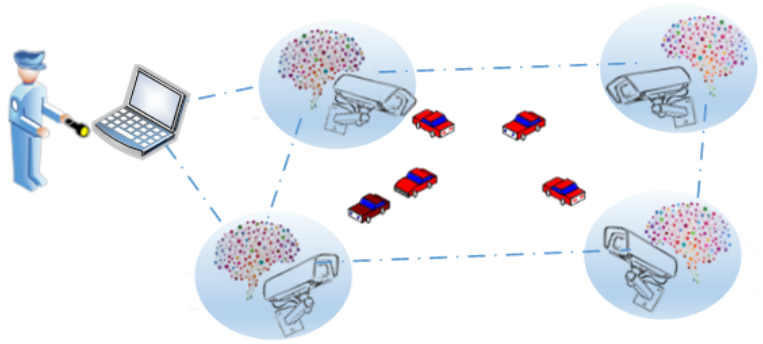

Figure 1.1: Guardian cameras model in current research

1.1. Topic introduction

Security is an important aspect of our daily lives. The safety of our society is threatened every day by actions ranging from petty theft to organized crime. The demand for security equipment has been growing to a billion dollar industry in North America as well as Europe. The global market for video surveillance alone was estimated to be 14 billion in 2013 and expected to grow to a 40 billion dollar industry in 2020 [36, 50]. Impaired with the growing demand for cameras is the means to store, distribute, visualize and analyse the images. Many commercial systems are already available, each competing to deliver that one feature that competitors do not have [9]. The days that visual surveillance systems only consisted of a camera and monitor seem ancient history.
The security on the officer training facility of the Royal Dutch Navy in Den Helder is managed by several security guards with an existing surveillance system. In their ongoing search to improve security on the premises, the interest in automatic reasoning systems arose and they asked to build a proof-of-concept reasoning system for their facility. The goal was to show the possibilities of automatic surveillance systems to detect car behaviour with the aim to improve the existing surveillance system. Previous work already showed many of the possibilities for individual intelligent cameras to detect & track cars, but proved to have scalability issues [12]. The next step in the search for security improvement was to find new methods to reason in large scale visual surveillance systems.

Figure 1.2: Typical controlroom is still governed by human operators with many monitors

Despite the technological progression, most surveillance systems are still based on human control as illustrated in figure 1.2. The fast growth of the number of cameras in the surveillance systems, also increase the demand for human resources to survey the environment constantly. This is often expensive or simple unavailable and introduces security risks.
Security surveillance means looking for the exception. The unwanted behaviour can occur at any time and any place and security guards are forced to monitor the environment constantly and watch hours of images just to detect the one unwanted incident. It is understandable that it is hard to stay vigilant, fight the boredom and keep attention to the surveillance task, making it even harder to detect the unwanted behaviour. In other scenarios, the surveillance task is difficult, due to many events that need tracking. Crowded environments and large areas force security guards to quickly switch the attention to different aspects in the surveyed area, putting an heavy mental workload on the guard. The overload of information will eventually influence the performance, making it easy for a person to slip the attention of the security guard, like in the populair puzzle game "Where's Waldo?". The automatic surveillance systems could potentially assist the guard in managing security, since it is such a resource demanding task and requires so much time and effort from security guards.
As the resource demand of automatic security systems grows larger, centralized methods for processing are no longer capable of processing all the information. The capacity of centralized processing units is limited to only a few cameras and often only support offline backtracking. Real-time analysis and tracking over multiple cameras often is not supported in large scale security systems. Next generation surveillance systems are challenged to find new techniques to distribute the tasks over multiple processing resources and cooperate with each other through well defined communication to achieve the goal of surveillance. By designing distributed reasoning, automatic surveillance systems would be able to track more objects over multiple cameras.

1.1.1. Guardian angels

In [59] the concept of a guardian agent was introduced to assist in potential dangerous environments. The human centred design applies software agents to assist operators and improve the performance overall. Particular when the human operator is inclined to make mistakes due to low vigilance or heavy (mental) workload. Dispite the limitations, the human operator is still the best choice for detecting suspicious behaviour and no automatic surveillance system will perform better. This makes the visual surveillance environment an excellent candidate for the guardian angel human centred design. The challenge for automated visual surveillance systems to function autonomously and for it to perform the tasks without human interference will not be overcome within a reasonable timespan. The visual surveillance systems definitely have grown in capabilities, but cannot nearly compete with the human reasoning. Instead of replacing the human security guard, the guardian angel design encourages to assist the human operator and increase the overall performances. By creating a decision support system, the security guard can be assisted in staying vigilance and process information faster. It has the best of both worlds, since the automated system remains vigilance and the overall security can still rely on the expertise of the security guard. It can act a tell tail for suspicious behaviour and provide summary of the behaviour to ease the evaluation of the security guard. But when the automatic surveillance system only does parts of the surveillance, how do we separate the responsibilities? What part is solved by security guard and what is solved by the system?
Figure 1.3 shows a diagram of the interaction between the world, the guard and the angel. Both the guard and the angel have specific means to perceive and interact with the world. The essential unique aspect of the guardian angel design is that the angel puts the guards interests first, instead of replacing his role in the world. By including the guard's mental state and the possibility to notify the guard of possible danger, the guardian angel principle improves performance by assisting the guard. The angel will assist first and act second. The design recognizes the potential of the human operator and helps the guard to achieve his goals.

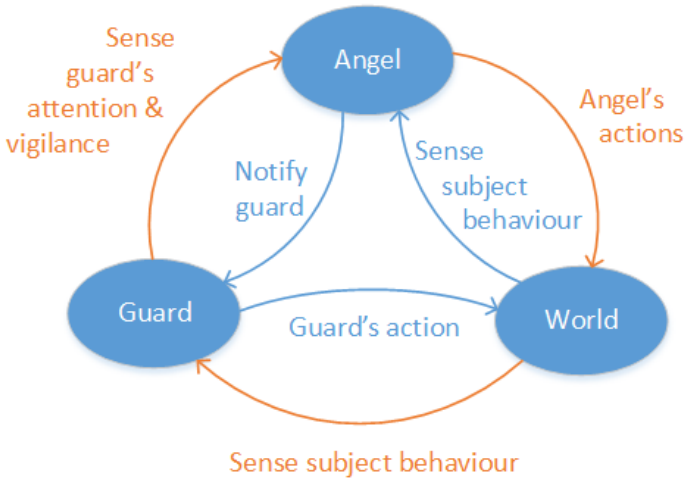

Figure 1.3: Guardian angel model in surveillance environment

Part of the reason why automatic surveillance systems still cannot compete with the human security guard is due to the ever changing behaviour within the environment. Guards constantly learn and adapt, because so is the behaviour within the environment. Automating security systems is still an open question; new protocols are still developed everyday and there is not one perfect system for all environments. But it does not stop the technology from growing more complex. A few years ago, video surveillance systems still streamed analogue black and white images over coax cables. Now we have wireless high definition cameras, streaming the information to advanced data centres for backup and automatic analysis[9]. Cameras themselves are becoming more complex and come equipped with temporary local storage, microphones, speakers and motion detection. Although security policies are still being developed, camera suppliers are racing to be the first to introduce new features and stand out from the crowd. The guardian angel principle wonders how the new technology best serves the human operators with their ever growing responsibility.

1.1.2. Cooperative cameras

With new technology come new challenges. Cameras are becoming more advanced, data centres are providing more options to safely store and easily monitor the images. Like the security guard, automatic surveillance systems also can become overwhelmed by the shire amount of data and have problems meeting real-time processing demands. Centralized automated system will clutter up and advanced detection algorithms have limited scalability. History thought us that the scalability issues are solved with distribution the computational power. But distributed approaches quickly grow in complexity and introduce new problems. Internal task delegation, communication and data distribution become an essential part of the design. The method of cooperation between the systems have just as much of an impact in the performance of the system as the algorithms for detecting, tracking and analyzing the behaviour in the world. In this research, we wanted to approach the problems that are introduced with the distributed approach of automated surveillance systems. We envisioned cooperative intelligent cameras, working together to assist the guard to survey the area. Imagen a building where the cameras communicate with each other and the security guard is only alerted when countermeasures are needed. Intelligent cameras could get rid of the need for centralized servers and proof to be more scalable than traditional systems. The design in shown in figure 1.4. We assume the human centered guardian angel design by building a system to the guard, instead of replacing him. Dynamic processing based on the guard's attention and vigilance is left out of scope, because the measuring of the mental workload is another research field entirely [32, 53, 65]. The same holds for the angel's actions within the world; these would only overcomplicate the research. We want to apply the agents to reason and cooperate in order to assist the guard by staying vigilant when the guard may not be and warn them when suspicious behaviour is detected.

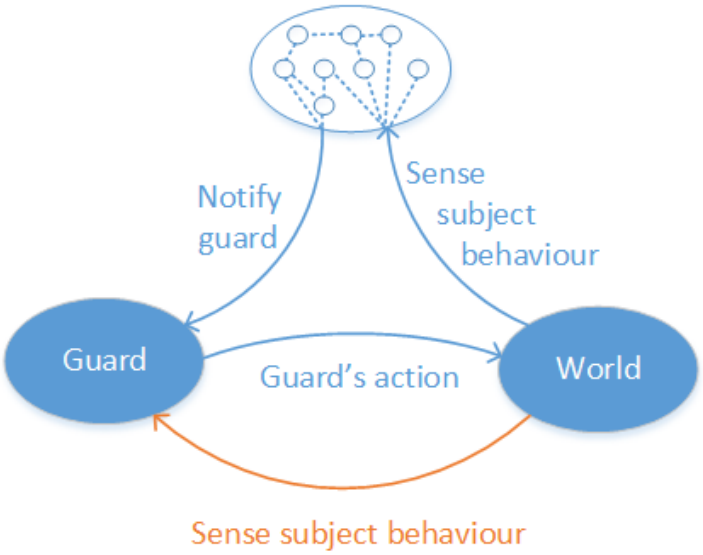

Figure 1.4: Guardian angels model in current research

1.2. Background

Understanding the relevants of cooperative smart cameras, requires us to look into the evolution of visual surveillance systems. This will be discussed in detail later in the thesis, but for now we will give a quick summary. Visual Surveillance Systems (VSS) are described in three generations. The first generation surveillance systems (1GSS) uses cameras to expand the sensory system. It allows users to see places with physically having to be there. Second generation surveillance systems (2GSS) made use of the digitalisation of the camera images to compress and analyze the images. The analysis is usually performed offline and on a centralized server, after the images were gathered. These systems have limited scalabilities and complexity of the analysis, due to the processing power of the centralized server. Third generation surveillance systems looks into methods of coping with large amounts of data with cloud services and decentralization of the behaviour analysis tasks.

Artificial Intelligence (AI) techniques have proven to be very helpful with both finding trends in behaviour and performing the countermeasures. The large amounts of data required in the surveillance tasks make them perfect candidates for assistance by software agents. But the increase in cameras within an average surveillance system, requires us to find new methods of reasoning and develop scalable reasoning methods. In the concept of smart cities and line this the development of the Internet-of-Everything, centralized methods of AI will not be sufficient to analyse the huge amount of data produced by the cameras. Analysing behaviour patterns in a city like Amsterdam cannot be performed with a centralized design, considering the city have over 2 million cameras installed.
In the past years, the Interactive Intelligence group of the Computer Science department of the Technical University Delft have been working on tracking and monitoring behaviour for different reasons (Managing flooding environment[60], analysing shopping behaviour[38], car parks[63, 70], ship traffic). The purpose of these systems is either to find tends in the behaviour or extend the capabilities of the security guard[59]. A few years ago we received a request to look at possibilities to upgrade the surveillance system for the Royal Dutch Navy officer's training facility in Den Helder. The upgrade would assist the security by detecting suspicious behaviour within the facility. Previous work has shown the possibility for cameras to detect and track objects over multiple cameras [70]. However, it also has shown a limit in the amount of cameras such tracking systems could support. This is an unsatisfying limit when the techniques would be enrolled to managing traffic over larger areas. New research would investigate if this limitation could be overcome by distributing the processing power over multiple cameras.
The current state of AI and robotics is insufficient for these tasks to be completely automated. The physical limitations and resourcefulness required for these tasks still demands human operators. The interaction of man and machine, in particular the task delegation between both parties plays an important aspect in the design of these supportive applications. The concept of the guardian angel encapsulate both the interaction between man, machine and the world [59].
Multi-camera surveillance is a very active research field, both due to the major advantages for managing security and the many scientific challenges in the field. Decentralized design for behaviour analysis, using cooperating intelligent cameras is only one of many aspects of a completely functioning automated visual surveillance system. The prerequisite processing steps before the behaviour can be analysed add many challenges to the research field. For example, the physical distribution of the camera's within the system has much influence on the possibilities of the system. Camera configuration and the state of the cameras are only a selection of the problems that need to be tackled before the system would be applicable. In this research however, we wanted to investigate how this knowledge can best be represented in a distributed reasoning system. The focus is on the design of the reasoning framework, so the problems faced in building a real life camera detection system are reduced to a minimum. This way, we could research how the system could reason about tracks and behaviour patterns of humans. Many of these practical problems are already discussed in previous work by the research group [12, 70]. Therefore, we decided to perform this part of the research in a completely simulated environment. The scenarios and entities of the training facility are used to design the simulation environment, but the practical problems are avoided to be able to focus on the task distribution.

1.3. Relevants

With growing number of cameras and images that need to be analysed within visual surveillance systems, the demand for human resources increases. Teams of guards will be working together to manage the security on the premises and the problem of coordination & communication naturally arises from the expansion of the team. Whether we talk about team of man or machines, we always improve performance when the coordination & communication is improved. The interaction between man and machine and the comparison between reasoning methods of (teams of) man or machines is what makes the current research so fascinating. The research covers key points of the Interactive Intelligence (II) research group at the Technical University in Delft (TUDelft). The Interactive Intelligence group aims to engineer empathy. To achieve this aim we combine research from different fields:

Agent-based reasoning

Aims to develop cognitive frameworks for various domains of applications, focusing on robots, human-agent/robot teamwork, serious gaming, agent-based simulation and negotiation.

Computational Intelligence

Aims to study machines that can learn through interaction.

Perceptual Intelligence
Aims to study human perception and improve automated sensory.

User-centered design
Aims to improve quality of life in contrast to replacing jobs with machines.

Over the past years, there has been a lot of research on agent based technology. An agent is a process with a certain level of autonomy and defined behaviour. Multi-agent systems are particular interesting for the Interactive Intelligence group, since it allows the study of group behaviour that arises from many individually controlled behaviour working together. The combined effort of many small agents can be designed to achieve common goals, while no one agent manages the process as a whole. Agents within smart cameras would hold certain spatial awareness, which allowed them to coordinate and equally distribute the surveillance task(s). Particularly the design of the agents and the effects of the design on the behaviour as one automated surveillance system could help engineers to develop better systems.
From a technical viewpoint, the research attempts to realize real-time detection, tracking and behaviour analysis, as compared to backtracking the behaviour. Most surveillance systems apply backtracking to see what behaviour had occurred. The cooperative cameras attempt to detect unwanted behaviour as it occurs.
From previous work [12] we find that the scalability of the system is not an easily achieved property within such systems and yet with the growing number of cameras within visual surveillance systems it is an requirement that becomes more important. When we look at the history of visual surveillance systems, the research also shifts from capturing the images (first generation surveillance systems) to distribution the processing load (third generation surveillance systems). A major advantages of multi-agent systems, compared to the traditional single process design is the high level of parallelism. Since every agent runs independently on a different processing unit, the total number of calculations can be much more and with the right design, the multi agent system has the potential to be a scalable system. The disadvantage of multi-agent system is de manageability of the software and debug process. Distributed systems are often hard to develop, due to the exponential growth in number of possible states. Agent based systems also have the autonomous property, which does not allow traditional white-box testing. Once the agent is running, internal processes cannot be analysed using the tradional methods. It is important for future development of multi-agent surveillance systems that we look at new methods of automated testing these agents.
The design proposed in this research takes the guardian angel approach, which is a human centred design, where software agents cooperate with each other and the operator to reason about and process the perceived behaviour of cars in a closed environment. The cooperation between man and machine is improved by gathering knowledge on the operators cognitive reasoning and adjust the provided information to the preference of the operator. The agent-based approach is chosen to both study the behaviour of cooperative agents and to look for methods to improve the scalability of the process. The scalability and real- time requirements is seen as one of the main scientific challenges in current visual surveillance system.

1.4. Problem definition

The topic follows in a line of research in automatic suspicious behaviour detection. It is important to separate the next and current step from the previous work and indicate where the previous work stops and this begins. We need to limit the scope of it to ensure the research creates a significant new topic. This is a challenge since the research follows directly from the previous work and weighs heavily on it. All the more reason why it is important to create a clear separation of the research topics and define what the current research will discuss. The problem definition is introduce as a guideline for the remainder of the thesis.
The main problem for this research is the design of a distributed model for reasoning about car behaviour, using multi-camera surveillance systems (MCSS). The model will be designed as a Decision Support System (DSS), assisting security guards by extracting tracking information and classify specific car behaviour. The topic is divided over the following sub problems:

- Design a model of cooperative smart camera reasoning for automatic suspicious behaviour detection as a decision support system for a human operator.
- Implementation of the model using JADE for multiple agent system and CLIPS for reasoning.

- Simulate the environment for the implemented scenarios using NETLOGO.
- Application to specific scenarios within the environment of the defence academy at Den Helder.
- Design an automated agent test framework to validate the performance of the proof-of-concept in the different scenarios.
- Testing different scenarios of one car travelling over the military area using different routes and show the ability to reason about the cars behaviour.

We emphasize once more that this research takes a different approach then one might expect from the research in MCSS. Traditionally the research focusses on designing algorithms for object classification, tracking or behaviour classification. Such research requires high definition cameras, located in controlled environment and treat multi camera solutions as a large variation of the single camera solution. For the current research, the cooperation, communication and task delegation between software agents is the main problem. The cameras are assumed to be able to capture the images from the environment without any missing information. Car are always visible and the detection process is simplified to bare minimum. The goal is to design a model of cooperation between the cameras to achieve the shared goal of suspicious behaviour detection.

1.4.1. Scientific challenges

Throughout the evolution of visual surveillance systems, the field stumbled upon different problems and the focus shifted from image distribution and processing to distribution processing and data fusion. The latter challenges form the focus of the current research. We will set the following scientific challenges for our research:

- Probabilistic reasoning framework for detection and recognition of cars.
- Share information and data fusion.
- Distributing tasks over multiple cameras.
- Design a scalable reasoning system.
- Communicate findings with security guards.
- Automate agent testing without violating the autonomy property of the software agent.

On top of these topics, the human centred design asks us to look into the psychology of surveillance. Sufficient knowledge is required about the human reasoning to design a decision support system.

1.4.2. Research questions

In order to validate whether the challenges are met, research question are composed. The research questions are guidelines throughout the thesis and in the end will be referred to as a measurements of the achieved goals. The following research questions are used in this research:

- Can a system detect cars from raw images and recognize the track within and between cameras?
- Is it possible to design a reasoning model in a hierachical way starting from multiple simple agents with limited view of the world at the bottom up to complex reasoning with global overview at the top?
- Is it possible to design an automated surveillance system for multi camera surveillance system?
- Is the developed model scalable?
- Is it possible to apply the developed model to real-life specific environments?
- Is it possible to automatically test the software agent's reasoning capabilities without violating the autonomy property?

1.5. Methods

The research topic will be investigated by building a proof of concept multi-agent behaviour classification system, which analysis virtual cars in a simulated environment. The inspiration for this simulation environment is gathered from the parking security management system of the Den Helder facility of the Dutch Royal Navy. From this case, we will extract the scenarios and build a simulation environment to support the re-enactment of these scenarios.
Before the system can be build, we will first perform an orientation in the research field. We will look into the related work to learn the commonly used approaches in multi camera surveillance system and behaviour classification. Equipped with the knowledge of the state of the research, we will set a framework for the case. The surveillance security management case is analysed through a run-down of the psychology of surveillance and specific behaviour classification tasks within the environment. Once identified, the tasks can be selected that will be the candidate tasks and use cases for automatic behaviour classification system. During this process, the requirements for the simulation environment and the proof of concept behaviour classification system will be separated and we will form a functional design for the final product. The functional design must incorperate all the components required to solve the problems defined in the previous paragraphs.
Next we will have the system design and build phase. The system will be designed for an environment with multiple intelligent cameras, however the implementation will remain in the simulated environment. The use of specific middleware will allow us to create an simulated environment that resembles the real scenario from the task delegation, cooperation and communication standpoint. The actual world model, including camera configuration, car behaviour and sensor data are simplified to minimum required complexity.
The experiments will be performed using automated tests. The simulated environment and virtualization of the proof-of-concept implementation make the topic a very abstract topic. By automating the scenario tests, allow us to make a reliable description of the reasoning and cooperative capabilities of the agents. Testing software is a mature skill with many available software and literature. Testing software agents however is a relatively new topic and methods are still being developed. The automated test environments will be customly build, which will both test the performance of our software agents and ccontribute to the development of automated software agent testing. We will test the system by running corresponding simulations. The tests contain feature and performance testing. The feature testing will show the systems ability to initiate the agents, setup the required communication between the agents and process the information. The performance testing focusses on the systems ability to extract the information, reason about the car behaviour and inform the guard about these findings. The system is mainly evaluated on its ability to automate the surveillance system, using a cooperative intelligent camera design. The performance on the car behaviour detection has lower priority, due to the unrealistic scenario that is created by the use of the simplified simulation environment.

1.6. Report structure

The report structure follows directly from the research method discussed in the previous paragraph. The orientation phase is discussed in chapter 2, 3, 4 and 5. The related work chapter looks at the evolution of visual surveillance system and represent the state of the art on the topics of video surveillance, car behaviour reasoning and agent-based technology. In the chapter context, we perform an analysis of the case, including a brief look in the local traffic surveillance philosophy, the multi agent design, feature extraction and case scenario definition. The orientation section finally results in the experimental design of the intended MCSS.
Chapter 6 and 7 respectively discuss the system design and implementation. Together they describe the building phase of the thesis. The system design begins globally by describing the components and roles and services of the agents. From there, the section goes over the feature extraction and behaviour analysis algorithms. The implementation chapter assumes the system design and goes into the technical details of the implementation.
The thesis is completed with the test results, conclusions and advise for future work in chapters 8, 9 and 10. Chapter 8 validates the functionalities discussed in the functional design. Chapter 9 summarizes the findings and evaluates the thesis based on the problem definitions. Finally, chapter 10 discusses the future work. It looks into how the gained knowledge can be used to implement the actual MCS and discusses what knowledge should still be gathered for the intended MCS to be realized.

I

Orientation

2

Related work

In the related work the state-of-the-art in the field of automated suspicious behaviour detection in analysed. A short history and evolution of the visual surveillance systems is given, to get a better understanding of the challenges in the research topic. From there, the analysis is split up into three topics in this field: (Visual) surveillance, reasoning systems and agent technology. Visual Surveillance Systems (VSS) contain the history of the research area and all the aspects involved in create the systems. The reasoning systems sections discuss the methods of behaviour analysis and data representation. Finally, on the topic of agent technology, we discuss the different cooperative models applicable and methods of testing the Multi-Agent Systems (MASs).

Figure 2.1: Research topics within the scope

2.1. Evolution of the visual surveillance system

Visual Surveillance Systems (VSS) are typically categorized into three distinct generations of which the 3rd Generation Surveillance System (3GSS) is the current generation [66]. The real history of VSS goes back to the first conflict amongst man. For a long time, despite advancements in weaponry to catapults, swords, and shields, the eyes and ears of warriors were utilized for surveillance. The invention of flying machines and the the start of the electronic age made it possible to design equipments that would give a tremendous increase to the range of the eye and the ear. The airplane enabled the procurement of information many miles in advance of friendly forces. [71] The evolution of the visual surveillance system has gained an tremendous acceleration with the Closed Circuit TeleVision (CCTV). It is considered to start the first generation of surveillance systems (1GSS).

Like many technology, the demand for automated VSS first came from the military. Rapid, complete, and precise information is needed to decide appropriate actions, based on the location of the enemy. Surveillance information had to be delivered to the correct commander when he required it and the information must be presented in a meaningful form to address the problem of information processing [52]. The fundamental intention of a surveillance system is to acquire information of an aspect in the real world. Military surveillance systems enhance the sensory capabilities of a military commander. Nowerdays, we send drones into dangerous areas to analyse the risk from hundreds of kilometers away. The intention remains the same and all of these modern techniques has been made possible through the building blocks of the first generation of visual surveillance systems. Even the most primitive surveillance systems gathered information concerning reality and communicated it to the appropriate users.

Generic surveillance is composed of three essential parts: data acquisition, information analysis and on-field operation. A surveillance system can be defined as a technological tool that assists humans by offering an extended perception and reasoning capability about situations of interest that occur in the monitored environments. Human perception and reasoning are restricted by the capabilities and limits of human senses and mind to simultaneously collect, process, and store limited amount of data [66]. While developing such new systems, the field progressed and included new problems with every new step.

Video surveillance has evolved from single analogue low resolution Closed-Circuit TeleVision (CCTV) system to distributed intelligent multi-sensor surveillance systems. Throughout the development of surveillance systems, other problems and scientific challenges rose and the focus shifted from plain image distribution to integration and communication of all types of information. The introduction of [72] gives describes evolution of intelligent surveillance systems. The table that summarises the evolution is reproduced in table 2.1.

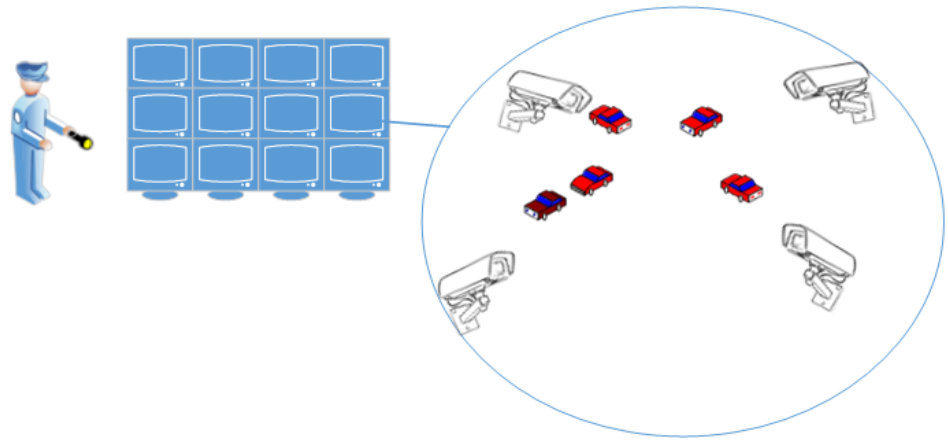

Figure 2.2: Typical first generation surveillance systems setup

2.1.1. First generation surveillance systems

First video generation surveillance systems starts in the 1960s with the introduction of the close circuit TV systems (CCTV). The technique is based on analogue signal and image transmission and processing. In these systems, analogue video data form a collection of cameras, which view remote scenes and present information to the human operators[66]. The main contribution of 1GSS is extending the perception of the human operator. It shows regions far outside the visual range of the operator and simultaneously shows regions far

away from each other, usually in the comfort of control room[57].
The principle of cameras and monitors used by 1GSS is straightforward and have been around long enough to be a very common technique. It has the advantage of being a mature technique that is easy to implement. The extend of the guards perception is both the advantage and the problem, because it significantly increases the workload of the guard. The main limitations of the 1GSS are due to the following points strictly related to analogue processing and transmission level[57]. The limitations include:

1. A large bandwidth is usually required that limits the number of sensors to be used
2. Analogue video is subject to noise in transmission and the stored information suffers from degradations in image quality during playback
3. On-line alarm detection for a large set of monitored sites is difficult as they are related to visual inspection of monitors by human operators with limited attention spans
4. Off-line archival and retrieval of information on significant events of interest is difficult due to the large amount of tapes to be stored and re-examined

Despite of the maturity of 1GSS, the systems are still being developed further. Research mainly focusses on the effective distribution and storage of the CCTV images. The conversion to digital signals is used to enhance the images and store them more efficiently, but no analysis is done about the symbolic content of the images.

Table 2.1: Summary of technical evolution of intelligent surveillance systems

1st generation	
Techniques	Analogue CCTV systems
Advantages	- They give good performance in some situations - Mature technology
Problems	Use analogue techniques for image distribution and storage
Current research	- Digital versus analogue - Digital video recording - CCTV video compression
2nd generation	
Techniques	Automated visual surveillance by combining computer vision technology with CCTV systems
Advantages	Increase the surveillance efficiency of CCTV systems
Problems	Robust detection and tracking algorithms required for behavioural analysis
Current research	- Real-time robust computer vision algorithms - Automatic learning of scene variability and patterns of behaviours - Bridging the gap between the statistical analysis of a scene and producing natural language interpretations
3rd generation	
Techniques	Automated wide-area surveillance system
Advantages	- More accurate information as a result of combining different kind of sensors - Distribution
Problems	- Distribution of information (integration and communication) - Design methodology - Moving platforms, multi-sensor platforms
Current research	- Distributed versus centralised intelligence - Data fusion - Probabilistic reasoning framework - Multi-camera surveillance techniques

2.1.2. Second generation surveillance systems

The technological improvement on digital image processing has led to the development of semi-automatic systems, known as second generation surveillance systems (2GSS). Most of the research in second generation

surveillance systems is based on the creation of algorithms for automatic real-time detection events aiding the user to recognise the events [72]. Automated visual surveillance is achieved through the combination of computer vision technology and CCTV systems. The difficulty still lies in the detection and tracking, which is required for behaviour analysis. The availability of automated methods would significantly ease the monitoring of large sites with multiple cameras as the automated event detection enables prefiltering and the presentation of the main events [66].

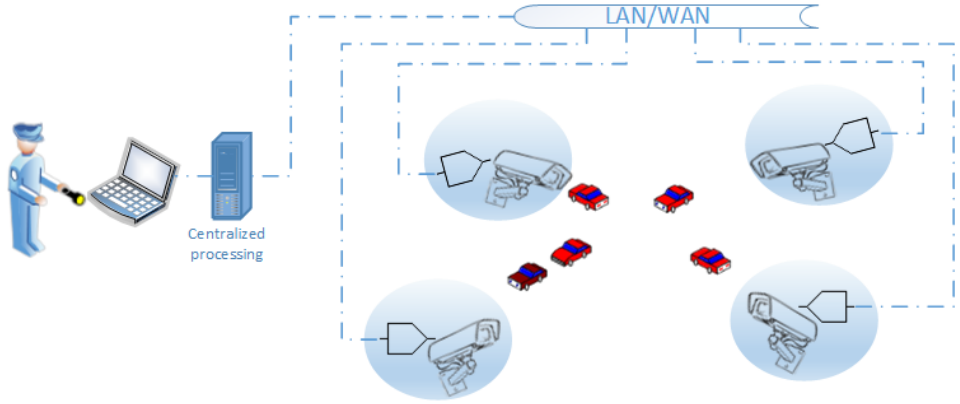

Figure 2.3: Typical second generation surveillance systems setup

2GSS is the first generation to begin analysing the behaviour within the environment. Typical 2GSS combine images from multiple cameras on a centralized server and perform analysis in a single process. The typical setup is shown in figure 2.3. This formed a bottleneck on both the network, as well as the processing power, as is also shown in previous work performed by our research group [12]. On top of that, the data fusion would set a heavy load on the control server. The development in surveillance cameras allowed the system to include multiple types of sensoring information, such as sound, identification cards, infrared detection, etc. The fusion of this data in one system and the distribution of the processing was the main motivation for the next generation surveillance systems.

2.1.3. Third generation surveillance systems

The main goal of thirds generation surveillance systems (3GSS) is to provide "full digital" solutions to the design of surveillance systems, starting at the sensor level, up to the presentation of mixed symbolic and visual information to the operators. From an image processing view, they are based on the distribution of processing capacities over the network and the use of embedded signal processing devices to achieve the benefits of scalability and potential robustness offered by distributed systems [10]. 3GSS goes hand-in-hand with the trends of cloud computing and Internet-of-Everything, which both depend on scalable systems that optimizes power consumption and utilizes decentralized systems. Diagram in figure 2.4 symbolises the typical 3GSS setup.
The cloud processing and storage shown in figure 2.4 is the topic of the 3GSS. The computation architecture defines the scalability, robustness and availability of the data within the surveillance system. Algorithms for processing of the images and analysis of the behaviour are still required, but the main concern with the 3GSS is the distribution of both data and processing power.

2.1.4. Modules involved in visual surveillance

The evolution of VSS make a distinct separation of the problems and challenges in each generation. However, it is an expanding progress and even a 3GSS cannot function without a imaging component or a tracking mechanism. Every VSS faces similar challenges, which has to be overcome within the design. Within multi-camera surveillance systems we distinguish five groups of technologies involved in 3GSS [75]. The groups of technology as shown in the diagram in figure 2.5. The arrows indicate the information flow between the different modules.

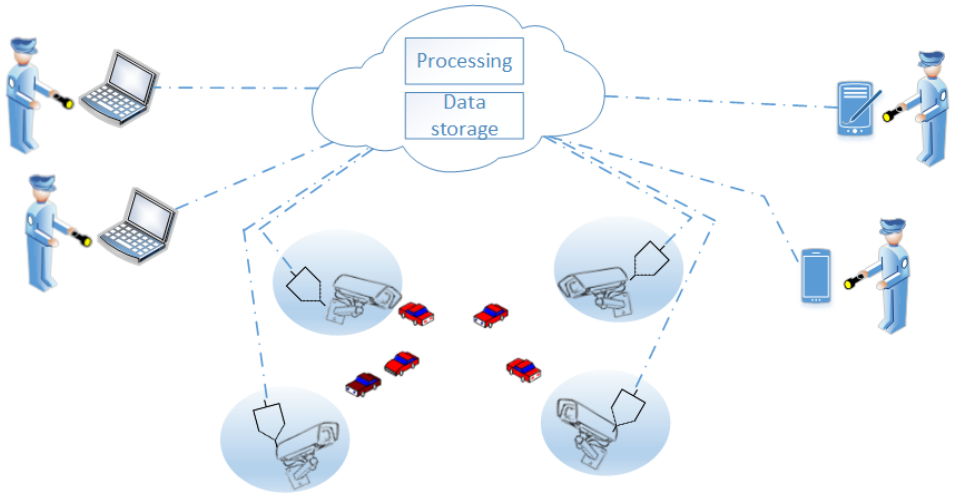

Figure 2.4: Typical third generation surveillance systems setup

Multi-camera calibration
Multi-camera calibration maps different camera views to a single coordinate system. In many surveillance systems, it is a key pre-step for other multi-camera based analysis.

Topology of a camera network
Topology of a camera network identifies whether camera views are overlapped or spatially adjacent and describes the transition time of objects between camera views.

Object re-identification
Object re-identification is to match two image regions observed in different camera views and recognize whether they belong to the same object or not, purely based the appearance information without spatio-temporal reasoning.

Multi-camera tracking
Multi-camera tracking is to track objects acress camera views.

Multi-camera activity analysis
Multi-camera activity analysis is to automatically recognize activities of different categories and detect abnormal activities in a large area by fusing information from multiple camera views.

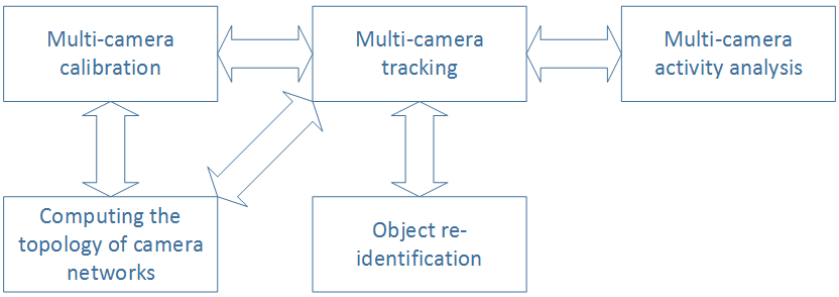

Figure 2.5: Technologies in intelligent multi-camera video surveillance [75]

Many existing VSS solve these problems sequentially according to a pipeline. This makes the software more manageable and eases the development process. However, sequential correlated processes may actually lower the performance, compared to the combined processes. Recent research works show that some of these problems can be jointly solved or even be skipped in order to overcome the challenges posed by certain

application scenarios [75]. The evolution of the surveillance systems suffer a similar fate. With each new generation, new problems arose, which can be attempted to be solved as individual problems. The performance of these choices unfortunately weigh heavily on the techniques used in the applied imaging techniques and tracking methods.
Intelligent multi-camera video surveillance faces many challenges with the fast growth of camera networks. As the scales of camera networks increase, it is preferred that the multi-camera surveillance system can self-adapt to the variety of scenes with less human intervention. Object re-identification and multi-camera activity analysis prefer unsupervised approaches in order to avoid manually labelling new training samples scenes and camera views change [75]. In turn, this may not always be possible, due to the reasoning applied within specific regions.
The three topics discusses further on from here on are just as much each others complements as the generations in surveillance systems. The diagram in figure 2.1 are drawn as overlapping topics on purpose. They topics also have a parallel with the generations of surveillance systems. The techniques for automatic video surveillance mainly concern the first and second generation and end with the tracking algorithms. With the introduction of digital cameras, it became possible to reason about the perceived images. The reasoning models are build on top of the tracking systems and apply logic and statistic methods to reason about the behaviour. Finally, agent technologies became an interesting research topic when the surveillance systems grow too large for traditional systems. This is a typical problem with the 3GSS. With every generation, the conclusions made in the previous generation would have to be reconsidered, while at the same time, the communication and task delegation heavily depend on the techniques used in the previous step.

2.2. Video surveillance techniques

Video surveillance techniques include everything done to extend the perception range of the security guard. The 1GSS mainly concern about the distribution of the (analogue) images to the user. There are some practical problems caused by the configuration of cameras over the region. These problems affect both the human and the automatic surveillance. We will quickly go over the practical problem and then continue with the techniques for automatic surveillance.

2.2.1. Camera configuration

The first problem for multi-camera surveillance is the visual region made available through the cameras. Obviously, when designing a VSS, it should cover as much of the visible region. Blocking obstacles and dark corners make it hard to detect objects within the region. Overlapping camera views make an inefficient configuration, but sometimes are the only way to look around obstacles. Designing a good camera configuration is an important aspect of the surveillance system overall. Once designed, the topology of the system is an essential piece of knowledge for interpreting the images [75].

2.2.2. Automatic surveillance

Automatic analysis of the images did not start until the 2GSS, due to the advances in digital image processing. A typical configuration of processing modules is illustrated in figure 2.6 [72]. Although the pipeline makes it easier to distinguish between the processing steps, any processing pipeline is unique. Depending on the problem, different steps are required and different approaches are chosen. Processing steps can be merged to solve them both in one step. Early automatic surveillance systems only set the goal of recognizing objects and solved that problem most efficiently. But in the process, they made it impossible to apply the technique to VSS that implement all the steps. We intend to show some approaches for each of the processing pipeline. Due to the different problems they all try to solve, the best approach for new VSS is more than the selection of the individual parts.

2.2.3. Pre-processing

The image interpretation process starts with images directly from the cameras. The pre-processing step enhances the quality of the frames for easier interpretation, starting with the object detection [16]. Like any classification process, the performance is dependent on the state of the input. Real life environment tend

Figure 2.6: Traditional flow of processing in visual surveillance systems [72]

to have many reasons why the images can be tainted, such as dirty cameras, passing animals and weather conditions. The capturing device itself also have its limits in terms of resolution and analogue-to-digital conversion process. The combination of external and internal factors for tainted images make every camera unique and the optimal pre-processing step is different for each situation. But we found two techniques that are reaccuring often in the literature: *noise filtering* and *background seperation*.
Noise filtering attempts to remove any noise in the images caused by the capturing method. The low-pass filter is an often applied technique to reduce the noise [58],[67]. Because the simulation does not include noise, we leave the subject of noise filtering for now.
Massimo Piccardi [45] reviewed about eight background subtraction techniques used for object tracking in video surveillance ranging from simple approaches, used for maximizing speed and restraining the memory requirements, to more complicated approaches, used for accomplishing the highest possible accuracy under any potential circumstances. All approaches intended for real-time performance. The techniques reviewed are: Running Guassian average, Temporal median filter, Mixture of Gaussians, Kernel density estimation (KDE), Sequential KD approximation, Co-occurrence of image variations and Eigen backgrounds technique.
Most segmentations methods use either temporal or spatial information in the image sequence. Most approaches use one of three methods[33]:

Background substraction
Background substraction detects moving regions in an image by taking the difference between the current image and the reference background.

Temporal differencing
Temporal differencing makes use of the pixel-wise differences between two or three consecutive frames in an image sequence to extract moving regions.

Optical flow
Optical flow uses characteristics of flow vectors of moving objects over time to detect moving regions in an image sequence.

Choosing the best method depends on the situation. Static cameras with constant lighting is best solved with a background subtraction method, whereas active cameras or dynamic environments can best be pre-processed using temporal differencing and optical flow respectively. The simulated environment used in this thesis has static structures, so we chose to apply background subtraction.

2.2.4. Object detection

Object detections include recognition of individual image regions, known as objects or patterns. There are many techniques to detection, too many in fact to describe all of them here. Pattern recognition by machine involves techniques for assigning patterns to their respective classes - automatically and with as little human intervention as possible. The most commonly used pattern arrangement is the feature vector, also known as descriptors. The key concept to keep in mind is that selecting the descriptors on which to base each component of a pattern vector has a profound influence on the eventual performance of object recognition based on the pattern vector approach [29].
Objects are mostly detected using shape characteristic detection, such as blob detection [11, 58], known colour or spatial histograms [35] or template matching [44, 78]. The challenging is to detect all the object, independent of posture, overlapping each other or other effects that impact the object representation.
Other approaches use the characteristics of movement to detect the objects. In [2, 77] the behaviour of ships is analysed. The system relies on regions of interest and verification of the realistic motion to detect the ships. Using the behaviour to detect the objects for suspicious behaviour detection is a dangerous method. Unusual behaviour may not be detected and will not be provided to the reasoning system. The ship detection method

works for this particular application, but may not perform well in the behaviour analysis problem.
The image from a video camera is a 2 dimensional image. Determining the position of the object required the image position to be mapped to the 2D or 3D environment representation model. Previous work in the research group presented a method for calculating the 3D position using the known configurations of the camera [12].

2.2.5. Object recognition

Detecting an object is only shows there is an object somewhere in the image. Recognizing the object as one particular person allows the system to apply rules specific to the target. There are numerous techniques to re-identify the object in different images. For example, [35] uses a voting mechanism between all detected objects to determine which person is detected. Without an object recognition method it would be impossible to track multiple people. Not all VSS require the system to track the individual objects, for example in the case of people counting [44] or tracking of unusual regions [69]. For our system it is important that the object recognition can track specific instance of cars. Else the complex behaviour cannot be analysed.

2.2.6. Track info extraction

Tracking information supply addition insight in the behaviour of an object. The position sequence allows the calculation of distance to other objects, speed, direction, acceleration, etc. In [68], we found that only supplying the security guard with tracking information on the objects already made a automatic surveillance system a valuable addition and reduced the intensity of the surveillance task.

2.3. Reasoning techniques

Digitalisation of the camera images made it possible to reason about the observed images. Reasoning techniques are applied to automatically analyse object's behaviour. Using the actions over time, the system is able to detect patterns within behaviour and classify unwanted behaviour.

2.3.1. Behaviour and activities analysis

In a surveillance scenario, they discriminate between normal and abnormal tracks. The detection of suspicious events has been the previous topic of our research group. In [63] an hidden markov model was used to assess the navigation behaviour of cars over the Den Helder facility. With the help of experts, the behaviour profiles could be made to assess the likelihood of unwanted behaviour. But they were not the only ones. Object and anomaly detection are researched in [3]. Their approach is based on modelling pixel level probability distribution functions of object speed and size from the tracks and was used for detecting local as well as global anomalies in object tracks. The research presented in [39] focuses on understanding human behaviour and interactions in complex dynamic scenes. They are able to find rules governing a scene like traffic sequences order, based on learning spatio-temporal dependencies in the scene. The first step is to learn dependencies between motion patterns and therefore extract local temporal rules of the scene. The second step is to jointly learn co-occurring activities and their time dependencies, generating global temporal rules.
As discussed earlier, in [69] the behaviour was assessed by learning the unusual regions within the image streams. In contrast to explicitly modeling specific unusual events, the proposed approach incrementally learns the usual appearances from the visual source and simultaneously identifies potential unusual image regions in the scene.
A smart surveillance system named CASSANDRA aimed at detecting instances of aggressive human behaviour in public environments is presented in [80]. A distinguishing aspect of CASSANDRA is the exploitation of the complementary nature of audio and video sensing to disambiguate scene activity in real-life, noisy and dynamic environments. At the lower level, independent analysis of the audio and video streams yields intermediate descriptors of a scene. At the higher level, a dynamic Bayesian network is used as a fusion mechanism that produces an aggregate aggression indication for the current scene.
A surveillance system that uses audio and video sensors to reveal and track the presence of an intruder in an off-limit area is presented in [48].The system is composed of a mobile agent and several static agents cooperating in the tracking task. The mobile agent is a vision agent composed of an omnidirectional vision system and a mobile robot. The static agents are acoustic agents composed of self-steerable microphone arrays and

a vision agent implemented on an omnidirectional vision system.
The multimodal workbench for automatic surveillance applications presented in [15] is centred on the shared memory paradigm, the use of which allows for loosely coupled asynchronous communication between multiple processing components. This decoupling is realized both in time and space. The shared memory in the current design of the framework takes the form of XML data spaces. This suggests a more human-modeled alternative to store, retrieve and process data. The framework enhances the data handling by using a document centred approach to tuple spaces. All the data is stored in XML documents and these are subsequently received by the data consumers following specific XML queries. In addition, the framework consists of a set of software tools to monitor the state of registered processing components, to log different types of events and to debug the flow of data given any running application context.
In the intended system, we want to learn the spatio-temporal dependencies in the scene, but we consider the behaviour different for each level. Similar to done in CASSANDRA, we want to find the suspicious behaviour triggers on each level.

2.3.2. Database

We found that the efficient storage and fusion of multi-modal sensory data is the least investigated topic within 3GSS. This is due to the work intensive task of creating a validation database as for example is done in [5]. The goal of the data for 3GSS is to efficiently store all information required to re-identify, track and analyse the behaviour of object. Depending on the methods used in the processing pipeline, different extracted features are important on each section of the process. In [6] an hierarchical database is proposed to store the data differently for each layer of analysis, including image framelet, object motion and semantic description layer. The proposed system gives an unique view on the storage of the data, however no attempt have been made to decentralize the storage of the data and distribute the information over the different cameras.
Logging the events on different simantic levels is valuable, but also allows automatic backtracking persons. Especially in large system, tracing a person's footsteps time consuming, work intensive task. There are already automatic backtracking systems on the market today [25]. However, we would want a real-time detection system that has the option of backtracking.

2.4. Agent technology

We choose to use an agent-based approach to the processes in the reasoning of the intelligent cameras. The agent-based approach allows us to focus on the cooperation between the cameras and their behaviour on an high level. The distributed approach for data processing and storage (as is the topic of a 3GSS) lies in the core of multi-agent systems. Added to the distributed approach, is a level of autonomy to the processes within the system. Agents have a spatial awareness of both the monitored environment, as well as the responsibility of each agent. Multi-agent surveillance system do not only know the configuration and topology of the camera, but also which agent is responsible for the data processing.
This section discusses different approaches in camera cooperation and task distribution. We review other approaches in the automatic surveillance domain and build an argument for the hierarchical approach used in the proposed system. Afterwards, we will discuss the topic of automatic testing in an agent-based environment. Due to the many parallel processes and autonomous nature of the software agents, we need to apply adjustments to the conventional methods of testing in order to test the agents.

2.4.1. Communication & cooperation

There are many different aspects to consider when it comes to the distributing the tasks between processes. On the one hand, an equal distribution of task load over all agents is desired, which prefers decentralized methods. On the other hand, the amount of data communicated should be both equally distributed and minimzed as well. Optimizing these properties depend heavily on the available hardware. Since there is no one simpel solution to this problem, all we can do is look at a few examples to understand the challenges and possible solutions.
[49] shows a cooperative approach within a multi-camera surveillance system. The system contains both static cameras and active cameras. Essentially, the cooperation of the cameras comes down to the calculation and communication of the relative position of an object, which are directly related to the angle and tilt

of an active camera. In addition, a new metric -Appearance Ratio (AR)- is defined to measure the reliability of each sensor and to fuse the measurements in a centralized manner. Unfortunately, due to this centralized processing, the work failed to reach the full potential of a distributed approach. It does show how the information from one camera can be communicated to other cameras to improve the surveillance tasks overall.
In [7, 8], we find an example of an agent based system, that uses the situation awareness to optimally use the available resources to automatically survey multi-camera environments with large amounts of cameras. Their approach defines clusters of intelligent cameras to dynamically allocate tasks to smart cameras. The framework for task delegation (named PoQoS [47]) combines power- and Quality of Service (QoS)-management in high performance distributed Intelligent Video Surveillance (IVS). It distributes tasks over the any device in the system, whose power- and/or QoS-parameters are dynamically configurable, named PoQoS Adaptable Units (PAU). The work shows the potential for multi-agent systems to perform large scale traffic surveillance. The hierarchical model for data processing allows for equal computation power demand distribution. Defining the algorithms within the tasks and designing the cooperation between the tasks remains a challenges on its own.
The problem of optimal task distribution plays an important role in many different research field. For the comparison between the different models, we are going to take a slight detour from the VSS domain and look at examples from the mobile agent cooperation. In [34] compared three models for scheduling refuelling activities for multiple mobile robots on a shared charger. The work compared centralized, fully distributed and hierarchical model. He found that “For small problem sizes, with high effifiency requirements, an off-line schedule can be found using the optimal approach, or proposed heuristic. For dynamic environments or large scale problems, the distributed, or hierarchical approach, using an adaptive threshold can be used to obtain reasonable results in real-time.” Decentralized method tend to find suboptimal solutions, but within shorter time. An hierarchical approach can distribute a lot of the planning and execution over the robotic team, thereby it retains the benefits of distributed approaches regarding to speed, flexibility, and robustness [27]. In the work of [34] the hierarchical approach outperformed the fully decentralized approach slightly on average and showed most potential in large problems. We decided to put this to the test by designing a hierarchical model for the agent reasoning.

2.4.2. Testing agents

The fast increase of complexity in behaviour of MAS is both its advantage and disadvantage. The cooperation of many simple agents can be used to solve complex problems. In nature for example, ants work together to gather food for the colony. While each ant only follows a simple reasoning system, the colony as a whole shows fascinating complex behaviour. This is similar to the multi-agent systems, were the individual tasks of a camera might be simple, yet together they manage the safety of the complete area.
At the same time, the growth in possible states when agents run parallel to each other, reduces the manageability of the software (agents). Ensuring the collective behaviour always does exactly what is required, can be a challenging task. Individual software agents can show bugs like any other piece of software, but even without them, the exponential growth of the number of states in distributed systems makes it hard to proof that the cooperative effort will always show the behaviour that was expected or required. Not only do we want to design the process, we will want to test the software as well.

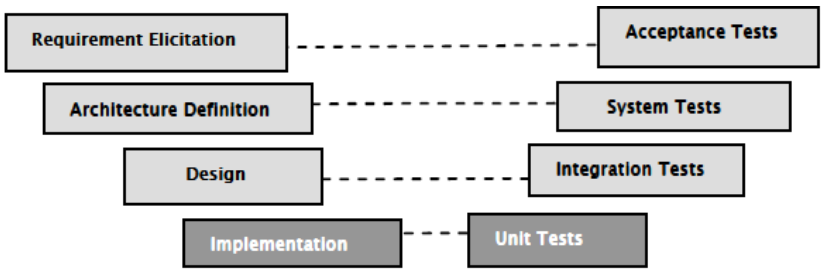

Figure 2.7: Development and Testing processes correspondence (Adapted from [51])

Testing is a skill on itself and should be considered in every step of the development process, as is shown in figure 2.7. For single process systems, the development cycle is reduced to small units of code, testing each

early in the development process. For Object Oriented (OO) code, the units go as far as testing each class separately. This ensures each unit works separately, increasing the chance that the software also works together. The same principle holds for testing software agents. However, testing each class within the agent separately is not allowed within the software agent principle. The default method of testing goes against the design principle of multi agent systems. The autonomy properties of the agents prevents us from testing the internal functionalities of the agents [74]. Agents are designed to communicate to each other through the proper communication channel and any other method to gather information goes against the autonomy of the agent.
Testing etiquettes tell us we should perform small fast tests, because the more classes are involved in the tests, the harder it becomes to trace the source of any potential bug. Ideally, the software is tested in units, because they are fast and the debugging is easy. However, multi-agent systems can only be tested as a whole and these tests are badly applicable in this environment. System tests (also known as End-to-End tests) are perfectly capable of showing how well the software works, but are often very slow and all discovered bugs in the software will take a long time to fix. Table 2.2 summarizes the properties of the different tests. New methods to reduce the class scope and duration of the tests would be very beneficial to the development process of MAS.

Table 2.2: Test method applicability

	Class scope	Speed	Applicable in single process code	Applicable in multi agent code
System / E2E tests	Large	Very slow	++	++
Integration tests	Medium	Slow	++	-/+
Unit tests	Small (single class)	Fast	++	–

In [13] an unit test approach is presented for MASs. The main purpose is to help MASs developers in testing each agent individually. It relies on the use of Mock Agents to guide the design and implementation of agent unit test cases. The proposed approach extends the existing JUnit test framework to allow the execution of JADE unit test cases on the JADE platform.

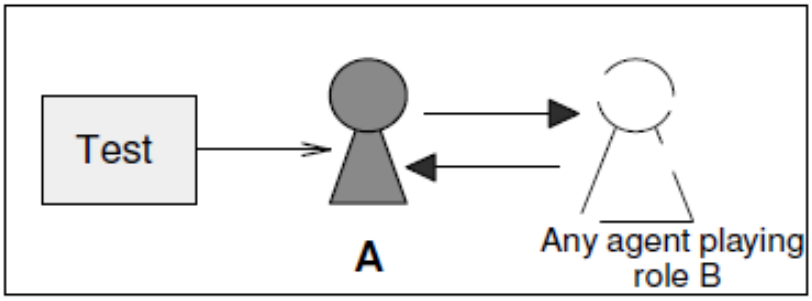

Figure 2.8: Test environment uses mock agents to interact with agents under testing

The approach relies on Mock Objects [46] to simulate real environmental resources. Mock Objects are regular Java objects that acts as a stub, but also includes assertions to instrument the interactions of the target object with its neighbours. The concept of mock objects and mock agents are combined to form mock agents. Mock agents are regular agents that communicates with the agent under testing (AUT) (as is shown in figure 2.8). By testing an agent in isolation using mock agents, the programmer is forced to consider the agent's interactions with its collaborators, possibly before those collaborators exist. We will be using the mock agents to test the functionalities of our system.

3

Context

The surveillance case handled in this thesis is the car surveillance on the Royal Dutch Navy training facility, located at Den Helder, the Netherlands. The goal is to design a DSS and automate (parts of) the detection and classification of suspicious car behaviour in a distributed environment. This task is currently performed by security guards and in order to reach the goal, we will automate parts of the reasoning process of the human operator. The aim is to develop the DSS, which assists human reasoning and is inspired by it in its design. In order to do this, we need a better understanding of the reasoning process performed by the security guard. In this chapter, we will look into the tasks of the guard, the reasoning steps involved in the tasks and gather the requirements of the automatic surveillance system. After this chapter, we will have a clear focus of the reasoning involved with VSS tasks in the Den Helder training facility.
The research starts with a generic view of the tasks within the local traffic surveillance. Once established, a cognitive model of the security guard is constructed. The cognitive model would assist in designing algorithms for automatic suspicious behaviour detection. Human intelligence is our main example of an intelligent system and proofs to be an informative design. The limitations the security guard must cope with, such as missing information are also problems for the cameras in the system. After we looked at the cognitive model of the security guard, the design design switches from generic to concrete and focuses on the traffic surveillance of the Den Helder Royal Dutch Navy facility used as a case for this thesis. The tasks are defined in scenarios, which serve as a guideline for the proof of concept application build for this research. The application will not be covering all the aspects of the traffic surveillance system and parts of it will be simulated. We will give a clear separation of the simulation and the application scope and define what behaviour the simulation environment will represent.

Figure 3.1: Research topics within the scope

3.1. Case introduction

The orientation starts with the environment in which the surveillance system will operate. As shown on the map in figure 3.2, the training facility is located in the Dutch town Den Helder, Noord-Holland. The facility is located near the harbour and houses offices, sleeping quarters, lecture rooms, navy supplies and the navy museum. The location is in between the city centre and the harbour, in particular the ferry to the island of Texel.

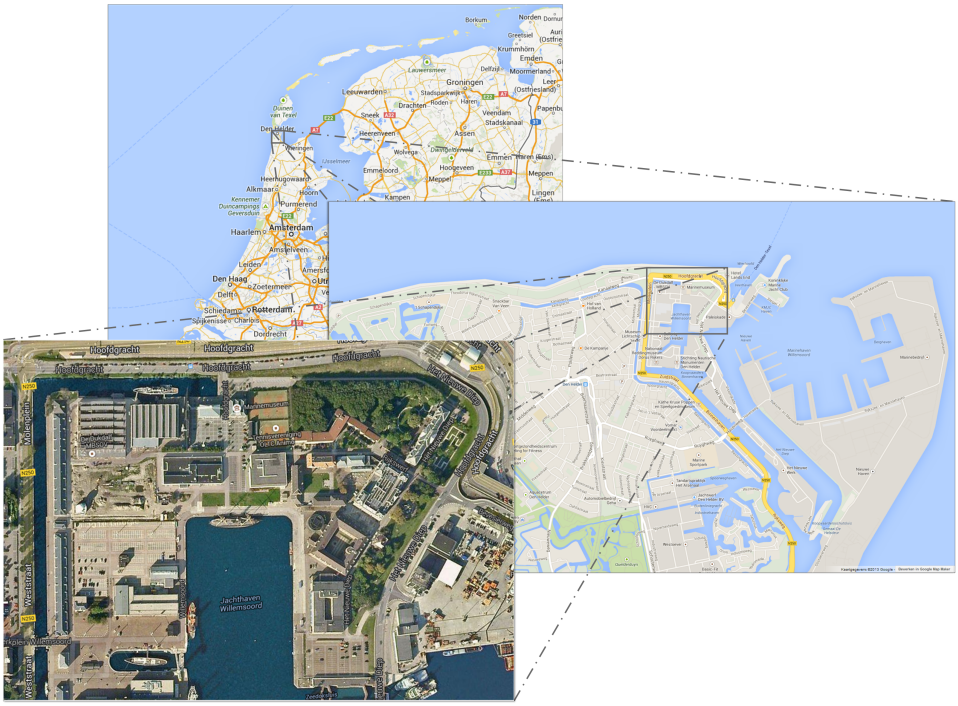

Figure 3.2: Den Helder, Noord-Holland

The current surveillance system consist of multiple active and static cameras distributed around the premisses. The images from the cameras are send to the control room, where a security guard is monitoring the activities as shown in figure 3.3. The security guard is responsible for the tracking of people and cars and interpretation of the actions of the observed objects. Due to a number of high value targets on the premises, such as the armoury or the offices of the Commandeur, security should always be vigilance for any attack or theft. However, the large number of cameras and the low ratio of attacks versus normal events make it hard to remain vigilance all the time.

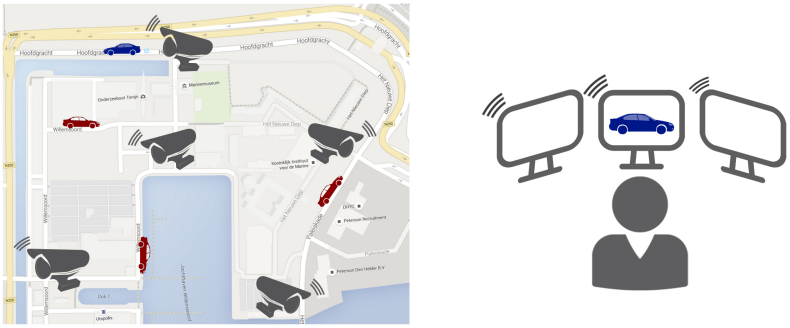

Figure 3.3: Current situation

The DSS is intended to be an extension of the existing system. It will have access to the same information and will have the video streams as input. For these video streams, the system will extract useful information for the security guard to be better at keeping track of the actions in the area. This is shown in figure 3.4.

Figure 3.4: Intended role automatic detection system

The analysis of the DSS is communicated to the user in three categories: *inform*, *warn* and *alarm*. The inform category is the information that is extracted from the actions that are not seen by the system as unwanted behaviour. This information is useful for the security guard to interpreted complex behaviour, such as scouting for a break-in or blocking an escape route. These are behaviours that require the understanding of the situation, a skill more present in human security guards. The inform category has for instance information about the track of a person or the time he has been parked. Warnings are the suspicious behaviour indicators. In these cases, the system has reasons to believe there is unwanted behaviour present or might occur in the near future. When the system is sure that unwanted behaviour is present in the environment, it will raise the alarm. An indication of how such a system would present the information to the security guard is shown in figure 3.5.

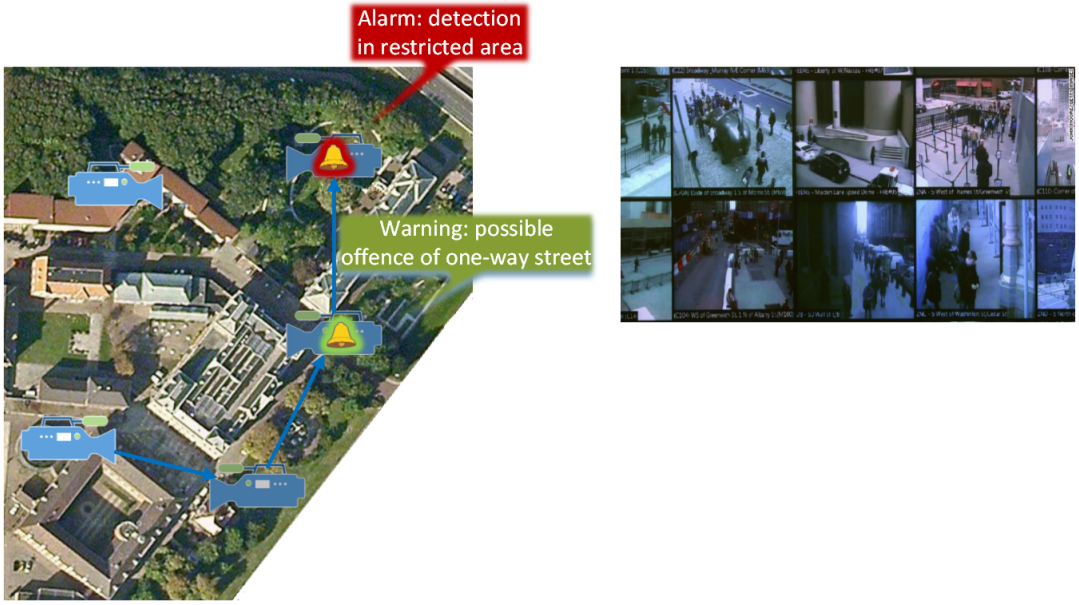

Figure 3.5: Goal interface

Now that we have an understanding of the case, we now discuss the specific contribution of this project. The current project is not the first from our department that looks into automatic behaviour analysis.

3.2. Models for evaluating performances

Since 2005, the research group Intelligent Interaction of TUDelft has looked into many different aspects of automated surveillance systems using smart multimodal cameras. The current work is a continuation of this research and focusses on a relatively new topic within visual surveillance systems, which is task load distribution and agent cooperation. In order paint a picture of how this topic became important, we will discuss previous work by the research group, summaries what we have learned from it and evaluate the work based on the criterias for the current research. This will show the focus points for the current research. In the past, we researched existing video surveillance systems in trains, railway stations, shopping malls, and military areas. In all those applications recorded video's (footage) were monitored by human operators in control rooms. To perform this monitoring job requires a lot of human effort and resources. Monitoring video recordings 24/7 is not a challenging job. The vigilance of human operators especially during the evening and in the weekends is not optimal. The upcoming terroristic threads requires 100% performance. Control rooms are not scalable. In a city like Amsterdam there are currently more that 2 million surveillance cameras installed. Monitoring by human operators is too expensive. There is a need for automated system.
The focus of automatic surveillance systems is an automated detection of aggressive behaviour. Currently, the human security guard is still irreplaceable. Primarily the technology is not advanced enough to completely replace the human security guard, but partly because of it, stakeholders often do not prefer it. Stakeholders often require that automatic surveillance systems should be modelled after human operator, since it improves the ease-of-use. This also allows a gradual introduction of automated surveillance systems, via (decision) supporting (semi-automatic)systems. To define the requirements of automated systems operators at work in the control room have been observed. The operators were required to think aloud. A list of possible important cases were discussed with the operators interviews. Detection of aggressive behaviour against employees, travellers, visitors and properties had the highest priority. The scenario's for aggressive detection of car drivers in military areas as listed in this thesis are also based on interviews of operators in the control room.
The existing camera network , conveying video data to the control room , monitored by the operators is the baseline. The question is how to support or improve the monitoring work of the operators. We list some possibilities:

- The cameras are located on places, such that human monitoring is optimal. In case of automated system location should be reconsidered.
- The quality of the camera should be improved to facilitate automated surveillance system. Cameras should be equipped with additional memory and processor to enable local processing. Even a microphone can be added.
- Usual only one operator is monitoring the video-data recorded in some area. In case of multiple cameras, we have to face the communication between cameras.
- One of the option is to consider distributed or decentralised processing of the data. One of the option is to model a camera as an agent who I able to perceive its environment, reason about observed data and take appropriate action. But the introduction violates for example the idea of centralised control.
- A problematic question is to create public domain annotated corpora of recorded data from surveillance systems to test different research methods. Simulated or acted data could be an alternative.

3.2.1. Criteria to compare surveillance systems

Automated surveillance systems can be designed in different ways using different models. In this section we describe different models used in past projects. To compare the different models we define at first some criteria. The different projects are ranked on this criteria using a five point scale. The final results are listed in table 3.1

Scalability
The proposed surveillance system should be able to handle the increasing amount of surveillance camera's.

Modularity
The proposed system consists of different detection units and detection units can be installed in every camera and should be able to communicate and exchange information with neighbouring units.

Multimodal fusion
The proposed system should be able to associate, correlate and combine data and information from multiple sources and multiple modalities.

Centralised versus decentralised
The proposed surveillance system should be able to handle the exponential grow of recorded data.

Vigilance
The proposed surveillance system should be able to analyse recorded video footage 24/7 and detect aggressive behaviour with a high detection rate.

Human model
The proposed surveillance system should mimic the human way of observation, feature extracting, reasoning and final classification.

Implementation on current camera networks
It is not an option to replace the whole existing video surveillance systems, here is only room for improvement of the quality of cameras and location.

Local/global detection
The proposed system should be able to detect aggressive behaviour on a local spot but also behaviour distributed over a large area.

Reasoning technologies
The proposed system should be able to reason about observed behaviour using rule based systems, Bayesian probabilistic reasoning and classifiers.

Experiments
The developed system should be tested using real life data, acted behaviour or simulated data.

3.2.2. Different models implemented in surveillance systems

In the framework of the smart surveillance project several surveillance systems were analysed. Every surveillance system has its own specific model. Next we discuss the successive models and discus their strength and weaknesses.

Single multimodal camera

In our first project Zhenke Yang [15, 54, 79] researched an aggression detection system in trains using single multimodal camera. A multimodal camera is a camera equipped with a microphone. In every train compartment four cameras were installed covering almost the whole compartment. Every camera was connected to a PC for data processing and to the machine-operators room for visual inspection. Special image processing software was designed to analyse the video recordings and to detect special aggressive behaviour. Special features extracted from the data was the amount of movement, the position of actors and distance between actors and special gestures. A special Bayesian network was used to reason about the extracted features to computable probability of aggressive events. There were good indicators of aggressive acts. The recorded speech signals were used for localisation of the speakers and speech processing. To record data and test the system the research team had access to a train compartment parked on a special place and a mock-up of a train compartment in one of the NS buildings. A team of stand-up comedians were requested to play special scenes. The topics of this scenes were the aggression detection topics as defined by the NS stakeholders.
The recorded data was annotated by observers in the control room and after training the system with statistical software and classifiers it proves that the system was able to detect aggressive acts in 70% of the cases. But we note the recorded data contains a limited scale of acts with overt aggressive behaviour. There was no transfer of knowledge and data from one compartment to the other. A fleeing man was difficult to track by switching from one camera to the other. The idea was that different cameras inside and outside the train the train communicate with each other so that an object can be tracked within and outside the train. One of main problems was that it was difficult or even impossible to attack an identifier to an object. Every camera was directly connected to the control room. This resulted in an overload of data at the central point. To connect and compare different data streams was too complex also because of synchronisation problems of different camera systems.

Network of distributed cameras

In a second project Iulia Lefter [41–43] researched a distributed camera surveillance system at a Military Area. Every camera covers part of the area (streets, parking places, entrance, exits, information desk) and sends its information to the control room. The main focus was to detect aggressive behaviour at the entrance of the area with toll gate and at the information desk of the safety guards. Special image processing and sound processing software was designed to detect aggressive behaviour and to generate a warning signal. In parallel all the data was forwarded to the control room.
Based on interviews with safety-guards special scenarios were developed of aggressive behaviour of visitors against the operator behind the information desk and detection of aggressive acts against material. All the scenarios were played by stand-up comedians. The recorded multimodal data was analysed using HOFT/SWIFT technology and different kind of classifiers. A difficult job was the annotation of recorded data using annotators of different expertise. An aggressive incident was clearly detected and classified, but the increasing threat before or after an incident was classified in different ways. It is of high importance that an aggressive incident will detected as soon as possible and not during the onset of the incident or after the incident. The agreement between annotators was about 70%. The detection rate of aggressive incidents was 65%. The results were discussed with operators in the control room and they were positive about a decision support system. They expect that the system could support their work during the night and during the weekends when there are less guards available and the amount of visitors of the military area is minimal.
Network facilities were not researched. Mainly because the network was out of date, the resolution of unimodal cameras was too low for image processing, not all areas were covered by cameras. But the main problem were security reasons. The surveillance rules and procedures of the safety guards are classified. Thanks to team members with a military background we were allowed to have interviews and discussions with the safety guards via this special team members.

Network of distributed cameras of different type

A third project was executed by Mirela Popa [55, 56] in cooperation with Philips. The goal was to research shopping behaviour in shopping malls with a focus on detection of theft. In a mock-up shop at Philips campus recordings were made of shopping stand up comedians. Via fish-eye cameras the trajectories of shoppers were analysed. Via charge-coupled device (CCD) cameras attached to the ceiling were used to monitor if shoppers put products from the regals in the basket or back to the regals again and not in the pocket for example The recorded interaction shopper-products movements were analysed using fusing Histograms of Optical Flow (HOF) with directional features. Between the several main research directions which contribute to human behaviour assessment, we used the approach proposed by Laptev et al. using space-time interest points (STIP) in combination with a multi-channel to recognise human actions. SVM classifiers were to recognize realistic human actions in unconstrained movies.
Thanks to Philips we had access to real life recordings of shopping people in a shopping mall. Again different kinds of cameras were used to observe the shopping behaviour. At the end we developed a prototype of a smart camera, a camera connected to a PC which was able to record, store, process and analyse recorded data and in case suspicious behaviour has been detected an alarm is send to the control room to activate safety procedures.
The focus in this project was on developing single smart cameras of different kinds and not on smart networks of surveillance cameras. The developed software was modular and could be installed on smart processors connected to the cameras. The performance of the system was reasonable. In 60-70% of cases of theft an alarm was generated. The system was 24/7 online. The cases of theft during the evening and robberies were detected automatically and the police was alarmed without dangerous intervention of shopping personal.
Network facilities are still under development. One of the researched option was face recognition at the entrance of the shopping mall using software developed within the group. The idea was in case a suspicious person entered the shop an increased alarm should be generated and the trajectory of that person should be tracked and his behaviour analysed in more detail. But because of privacy issues and failing communication facilities in the surveillance network, this option was not researched in more detail.

A surveillance system of a military harbour using AIS (Automatic Identification System)

An interesting sub-project performed by Krispijn Scholte [61, 62, 64] of the surveillance project was the surveillance of ship movements around the Military harbour of Den Helder using the AIS system. Every ship with a minimal size has to use a special transponder. Such a transponder broadcast information about the identity of the ship, its speed, position, heading. AIS data are public domain available via Internet. The idea

is that ships are able to detect ships in their neighbourhood and take actions to prevent collisions. But we preferred recording via a special antenna attached to a surveillance tower at the entrance of the harbour. This special antenna covers an circle area with a diameter of 60 km. Only with permission ships are allowed to enter the military harbour at Den Helder and to approach oil platforms. Because of increasing threats of terroristic attacks, illegal fishing activities and violations of ecological rules, ship movements are monitored by the Coast Guard in the control room. Via interviews t Via the recorded AIS we were able to record and analyse ship movements. In some areas ships have speed limits, have to keep their lanes, are not allowed to embark. Again via interviews of the operators in the control room we were able to extract knowledge about detection of aggressive acts. This knowledge was implemented in a rule based system and Bayesian reasoning system. The system was able to detect violations of shipping behaviour rules 24/7. The premise was that the AIS system was not switched off. But in those case also an alarm was generated if this happens on unusual places, outside the harbour or places to embark.

Intelligent multicamera video surveillance

From 2010-2014 several students [31, 60, 79] conducted their BSc or MSc thesis project in the framework of video surveillance system project. Different architectures and computational models were used. Different projects were based on ad-hoc approaches and knowledge from were hardly reused or combined. Most student prefer their own innovative approach

- In almost all projects cameras were modelled as agents. Cougar or Jade were used as agents framework.
- In many projects tracking of people was a dominant topic. Moving people at the TUDelft campus were recorded. Tracking of people was used to detect if people used forbidden tracks. Stand up comedians were requested to play aggressive scenes and the recorded data was analysed using different technologies.
- In a system of distributed surveillance camera all recorded data was send and stored at a central BlackBoard. Special agents were able to extract features from the recorded data, fusing of different modalities and removal of analysed data to prevent.

The most interesting results were summarised in some papers presented in Journal papers or Conference Proceeding.

3.2.3. Preliminary findings

The different projects are ranked on this criteria using a five point scale. The final results are listed in table 3.1. The columns of the table correspond with the defined criteria and the rows with the different projects.

Table 3.1: Summarizing table

Criteria / Projects	Scalability	Modularity	Multi-modal fusion	(de-)Centralised	Vigilance	Human model	Implementation	Local/global	Reasoning	Experiments
Single multi-modal camera	+/-	+/-	+	dec	+	+	+/-	local	+/-	+/-
Network of distributed cameras	+	+	+	dec	+	+	+	loc/glob	+	+
Network of distributed cameras of different type	+	+	-	dec - cen	+	+/-	+	loc/glob	+	++
A surveillance system of a military harbour using AIS	+/-	+/-	-	cen	+	+	+/-	global	+/-	+
Intelligent multi-camera video surveillance	+/-	+/-	-	cen	+/-	+/-	+/-	global	+/-	+

3.2.4. Conclusions

After reviewing the projects previously performed by the research group, we can draw a number of conclusions about the future projects.

1. Most of the listed projects end up with a prototype of a surveillance system. This prototype can be used as a proof of concept, but cannot be used in real environment. There is a need for a full implementation of a surveillance system.

2. There are only two projects (shopping mall, AIS system) where the developed surveillance system has been tested on real life data. Because of privacy and security reasons this was not possible in the other cases.

3. None of the developed system has been tested by human operators or matched with human operators. None of the operators was able or willing to take part in such an experiment. Most of the systems were discussed with human operators and labelled as promising. So far human operators believe that they outperform every automated system.

4. In most projects the reasoning module operates as a stand alone module. Only in the BlackBoard approach data of multiple cameras has been fused and used for reasoning.

5. Most project take either a centralized approach or only consider local behaviour. There is a need for decentralized approaches that are able to reason about global behaviour.

3.3. Philosophy of suspicious behaviour detection

Suspicious behaviour classification tasks is the first part of the security management of people within an localized area. Aside from the identification of the unwanted behaviour, security management also includes planning and executing of the course of actions to countermeasure the unwanted behaviour. In order to be able to identify the unwanted behaviour (and for us to design an automated system) we require a clear definition of unwanted behaviour. This definition is specific and can be different for each situation. The same holds for the plan of action to countermeasure the unwanted behaviour. The computer lacks the creative thinking to evaluate the behaviour without a clear definition of right and wrong. We depend on expert knowledge to define the behaviour for each situation and provide the restrictions.
But the problem of behaviour identification is even more complicated due to the restrictions caused by the way we perceive the area. The area is perceived by multiple cameras, allowing the guard to monitor the complete area at once. While a car is monitored, the guard is constantly evaluating the behaviour and gathering proof for a definitive classification of the behaviour. But in many cases, this absolute certain classification cannot be given, due to visible range or configuration of the cameras. In such cases the guard has to act, based on suspicious alone and decide how to gather. We devised a reasoning process performed by the guard. This reasoning process is shown in figure 3.6.

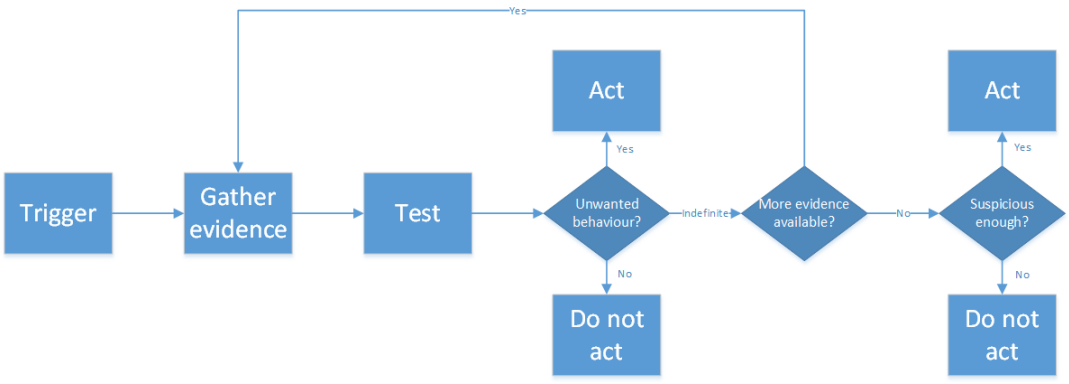

Figure 3.6: Guard's reasoning process

The reasoning process is the first time in this thesis we divide suspicious and unwanted behaviour. The difference between the these types of behaviour plays an important role in the reasoning method applied.

This is why we spend some extra time in the analysing the types of behaviour.

3.3.1. Deterministic versus probabilistic

Suspicious behaviour is a difficult term without proper definition. We define suspicious behaviour as a phenomenon that has a chance to be unwanted behaviour, but there is not enough proof to classify as such. Behaviour can arouse suspicion of being unwanted, but the further away we are from proving (or disproving) it, the more complex the behaviour becomes. Complex behaviour requires a creative method to gather the necessary information. The reasoning loop shown in figure 3.6 is part of this creative process. A guard can be triggered by the events on one camera, but may require the information from multiple cameras to proof the unwanted behaviour. Another option may be to switch on a microphone, when the camera images alone are not enough to gather the proof. The process of finding methods to proof the behaviour is the creativity of the guard and works best when the guard understands what he is looking at and knows what he can do to extends its knowledge.
The line between unwanted and suspicious behaviour is all about how the environment is perceived, including time, space and measurement type. Proving unwanted behaviour in a drivers actions may require a combination of multiple sensors, distributed over an area and measuring over a period of time. In the case of multi camera surveillance, we have limited coverage and detectability of actions. The setup of cameras may result in an incomplete coverage of the area, but also could have overlapping regions in the images. This is shown in the diagram of figure 3.7. The actions of a car outside the visible region have to be estimated, based on the available information. The same principle holds of actions that are undetectable by camera images alone, such as turning the engine off. Other sensors are required to get the full set of evidence.

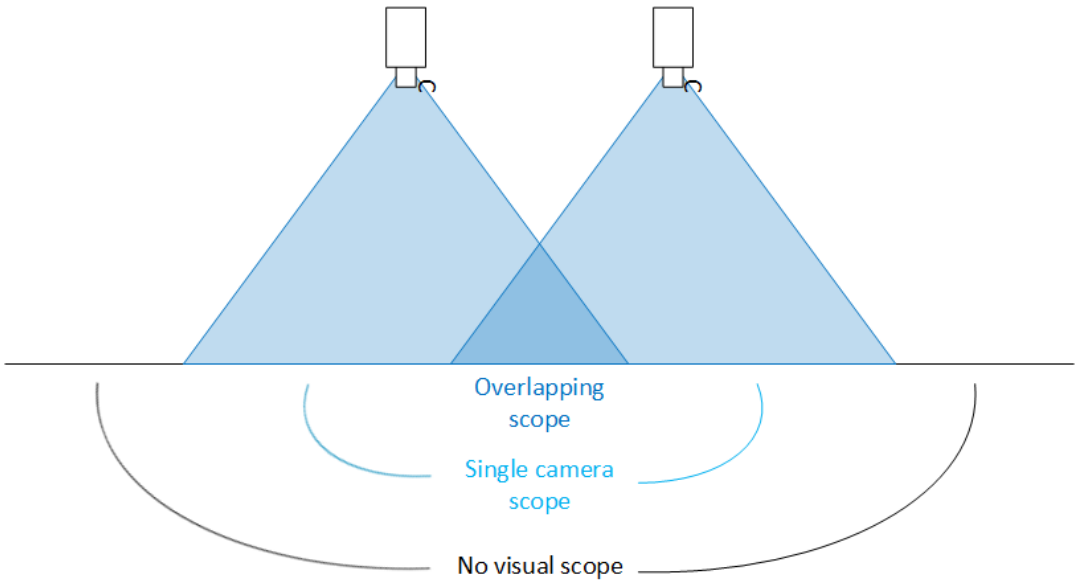

Figure 3.7: Perceivable regions

Suspicious behaviour is reasoning about the unperceived, may it be outside the time frame, measured space of measurement type. The known samples (Θ) are used to estimate the likelihood of unwanted behaviour $P(U|\Theta)$. The difference between unwanted & suspicious behaviour ultimately is the difference between deterministic & probabilistic reasoning.

$$\begin{aligned} \textbf{Unwanted behaviour:} \quad & P(U|\Theta) = 1 \\ \textbf{Suspicious behaviour:} \quad & 0 < P(U|\Theta) < 1 \end{aligned}$$

3.3.2. Implicit and predictive behaviour

We found two types of suspicious behaviour: *implicit* and *predictive* suspicious behaviour. Implicit suspicious behaviour is behaviour that could be unwanted behaviour, but there is evidence available to proof or disproof that the behaviour is unwanted. Predictive suspicious behaviour is behaviour that in itself is not unwanted behaviour, but is known to lead to unwanted behaviour in the future. The reason we separate these types of suspicious behaviour is that they both have a different method of handling.
Implicit suspicious behaviour is caused by a lack of sensory equipment. The main aspect of implicit suspicious behaviour is that the unwanted behaviour has already happened and the measurements arouse suspicion of this event. The actions of the security guard can only affect the consequences of the unwanted behaviour. Further investigation could proof the detection was correct, since the behaviour actually occurred. It requires the creative process of gathering evidence, even if it is as simple as going to the scene.
Classifying on the basis of predictive suspicious behaviour could prevent the behaviour from occurring or catch the criminal in the act. The downside of predictive behaviour detection is the validation, since the countermeasures may stop the unwanted behaviour from ever occurring.
From a technical viewpoint, classifying predictive suspicious behaviour is not much different from regular classification. Non-predictive classification would base the reasoning on information extracted from the observations. Classification of predictive suspicious behaviour is based on the predictions made from the observations. This is represented in the time diagram in figure 3.8. As we saw in section 2.1.4, it might be best to combine the predictor & classifier, but it will continue to have the predictive nature, included in the calculations.

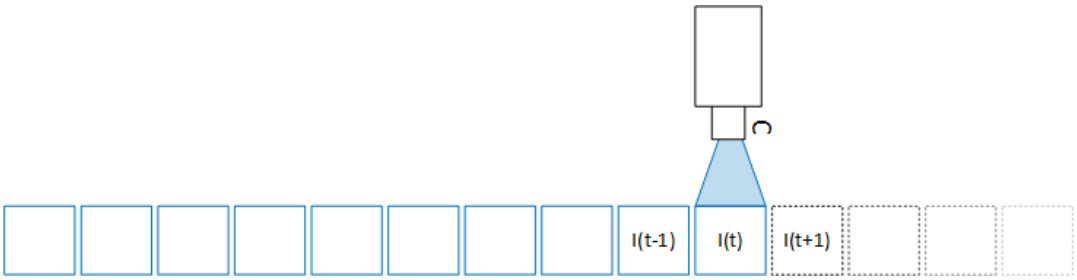

Figure 3.8: Reasoning about future events

Both types of suspicious behaviour are evaluated by the security guards. In fact, the reasoning is constantly changing, since the security guard is learning new aspects of the environment. For example, guards may find that cars entering from one entrance tend to park near the water, which is not allowed. Entering from this side would become a trigger and predictor for unwanted behaviour. Human operators are particularly good at finding these correlations, but this knowledge is hard to gather in an automatic system. Scenarios of unwanted behaviour can be changed to suspicious reasoning, when motivated by the increase of chance of detection. This creates a nesting doll effect, which for each step notifies a system with a larger perception or better suited conceptual model of the world. Again referring to the loop in figure 3.6, we find the guard is constantly changing the reasoning steps applied.

3.4. Cognitive model

In an attempt to define the human reasoning model, we further investigate how the human mind works during surveillance tasks. Through cognitive science we aim to model to mind and explain how it processes the information. the mind is a vastly complex process and any model will most likely be an oversimplification, but we can learn from it and use it to adjust the decision support system to best fit the operaters needs. Using the research in cognitive science, we will design a model of the reasoning process, applied by the guard, to detect the typical and atypical behaviour in the environment. We will then use the model as inspiration for the design of the automatic behaviour classification system.

3.4.1. Reasoning models

The reasoning steps introduced in the previous section is strongly inspired by the OODA-loop. The OODA-loop was introduced by John Boyd during his studies in gaining the strategical advantaged in war situations. The model helps to understand learning processes, which are applied both by friend and enemy. Boyd's research showed that in order to get the strategical advantage, one would be best off applying a proactive attitude and adapt quickly to the changing environment. The situation would only grow in complexity with every action taken by both you and your opponent. Only when all the information is available, would you be able to make the best decision, but by the time all the information is gathered and processed, the situation would most likely already be different. Stimulating proactive attitudes in the army would both decrease the time needed to process the information and lead to more actions that the opponent needs to process. This would slow them down, increasing the chances of gaining the upper hand in a potential dangerous environment. The OODA-loop is used to form the basis of this argument and is still used often in training Navy officers to learn strategic planning.
The reasoning cycle of the OODA-loop (figure 3.9) contains of four stages: Observe, Orientate, Decide and Act. During the observe stage, the agent observes the environment. The available senses are used to sense the environment and the predictions about the progress of the environment from the previous reasoning cycle are validated. The observe stage is followed by the orient stage. In this stage the agent uses his past experience, believes and conceptual model of the world to extract information from the observations. The extracted information also includes predictions about the future state of the environment. Simultaneously, the conceptual model adapted to the new observations, through a process of destruction and creation of the conceptual model, which forms the learning process. Once the believed state of the environment is constructed, decisions can be made about the appropriate actions. It is identified with a process of planning. Last, but not least the agent acts and uses the options available to manipulate the environment. The cycle is repeated continuously and due to the adaption, the guard learns and adapts to the ever changing environment. According to Boyd, there is a necessity to fast adaptation. The environment will only grow in complexity, which is a challenge for every agent in the environment. The party that adapts faster, can act faster and in turn make the conceptual model of the opponent outmoded.

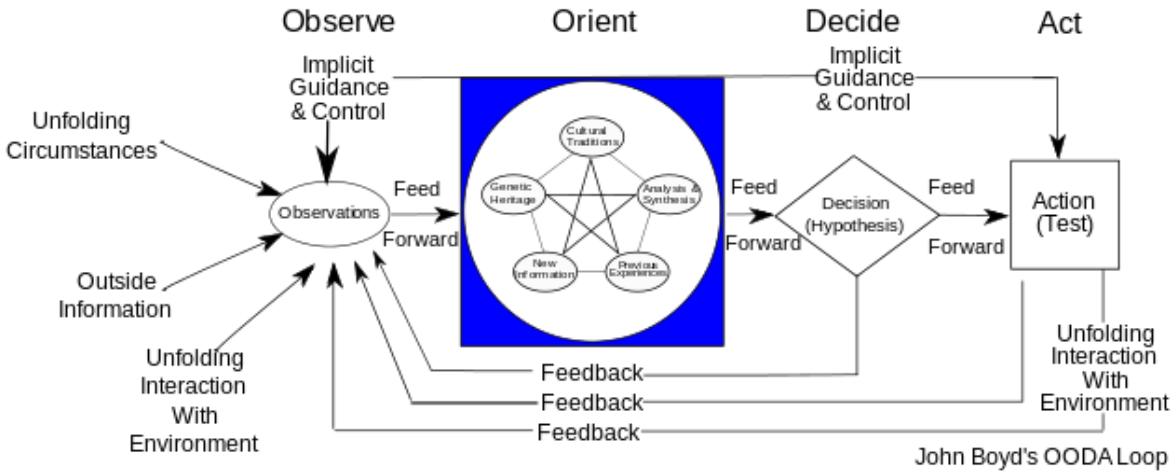

Figure 3.9: OODA-loop model

For our purposes, we will assume the security guard has variation of the OODA-loop in his reasoning cycle. The guard senses through the cameras distributed over the area. The images are interpreted from his conceptual model of the environment. The model is based on both the camera images and the knowledge of the environment. In contrast to the computer, the guard has probably walked around on the premisses and uses that knowledge to understand behaviour the images represent. Then the guard uses the knowledge to form a plan of action and decides where or not he should do something to counter-act the observed behaviour. And similar to the war situation, the sooner and more accurate the guard can detect or predict a crime, the better he would be capable of resolving it.

3.4.2. Reasoning complexity

Not all tasks require the same level of cognitive reasoning. While some behaviour is complex and require a true understanding of everything in the environment, other behaviour is straightforward that the guard could almost act instinctively. Rasmussen places human behaviour in three levels of performance: skill-, rule-, and knowledge-based performance as shown in figure 3.10. In the model, signs and symbols belong to two different universes of discourse: a sign is a part of the physical world of being, a symbol is part of the human world of meaning. Skill-based behaviour represents sensorimotor performance during activities that, after a statement of an intention, take place without conscious control as smooth, automated, and highly integrated patterns of behaviour. At the skill-based level, the performance is similar to that of a data-driven continuous system, conditioned by signs in terms of patterns in the sensory information or intentions, or both, supplied from the higher levels of control. At the rule-based level, the information is typically perceived as signs. Here the mind performs sequences of subroutines in a familiar work situation and is typically consciously controlled by a stored rule or procedure. The rules in Rasmussen's model can be learned empirically during previous occasions or communicated from instructions or cookbook recipe. The performance is goal-oriented, but structured by feed-forward control through a stored rule. After many repetitions, the behaviour (or subroutines within it) from a higher level can be learned in a lower level. In general, the skill-based performance rolls along without the person's conscious attention, and he will be unable to describe how he controls and on what information he bases the performance. The higher level, rule-based coordination is in general based on explicit know-how, and the rules used can be reported by the person, although the cues releasing a rule may be difficult to describe. The information perceived in the lowest performance levels is defined as a sign when it serves to activate or modify predetermined actions. Signs refer to situations or proper behaviour by convention or prior experience; they do not refer to concepts or represent functional properties of the environment.

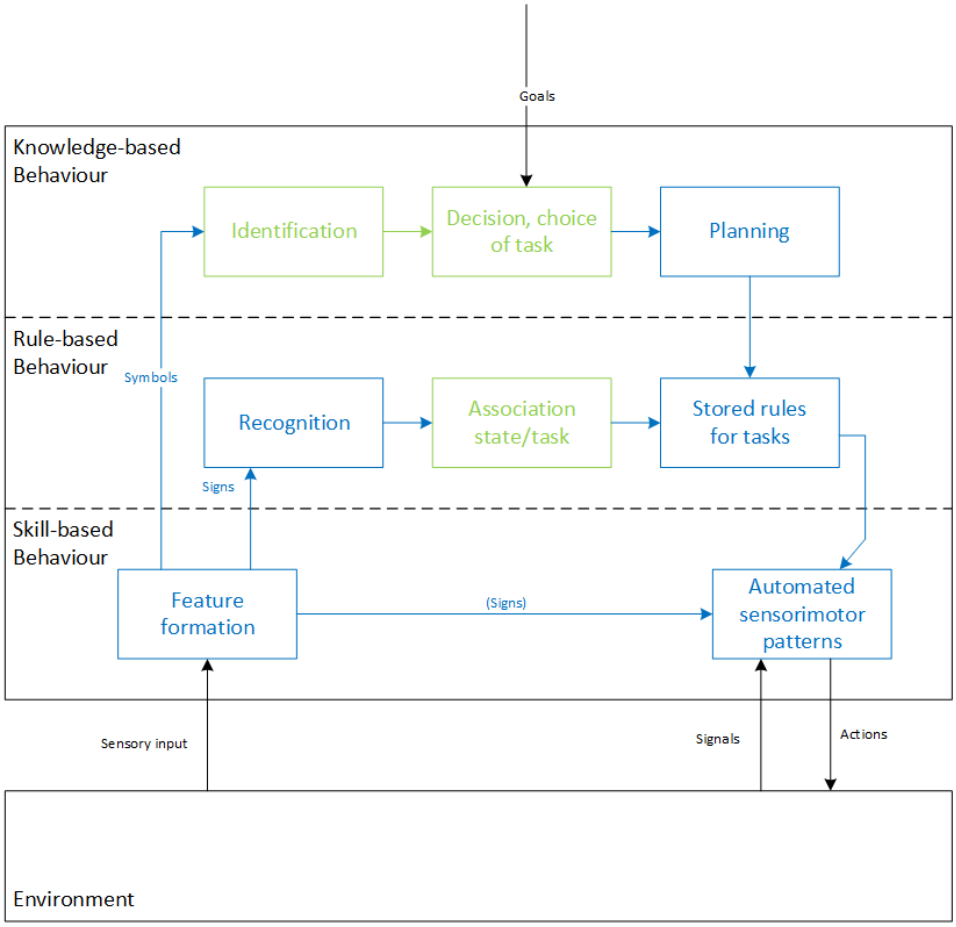

Figure 3.10: Three levels of human behaviour

The highest level of performance of human behaviour described by Rasmussen is the knowledge-based behaviour. Knowledge is here taken in a rather restricted sense as possession of a conceptual, structural mode. The behaviour at this level is based on symbols, rather than signs. Whereas signs refer to percepts and rules for action, symbols refer to concepts tied to functional properties and can be used for reasoning and computation by means of a suitable representation of such properties.
The model proposed by Rasmusses shows how human perform actions based on different levels of reasoning. At the same time, they form the inspiration for reasoning models in agents as well. We review three agent architectures with the purpose of showing the impact of reasoning complexity [4].

Rule-based reasoning agent

The simplest architecture we review is the rule-based architecture. The rule-based architecture is an instinctive system, which have no further planning component. It requires no higher cognitive functions. The state of the environment follows directly from the observations and through a process of matchmaking, the action is chosen. The advantage of this architecture is that it is usually fast. The disadvantage is that the rule are made in design time and the model does not adapt to chances in the environment. Also, because it has no knowledge on the reason for these actions, it will never detect opportunities to solve the problem more efficiently.

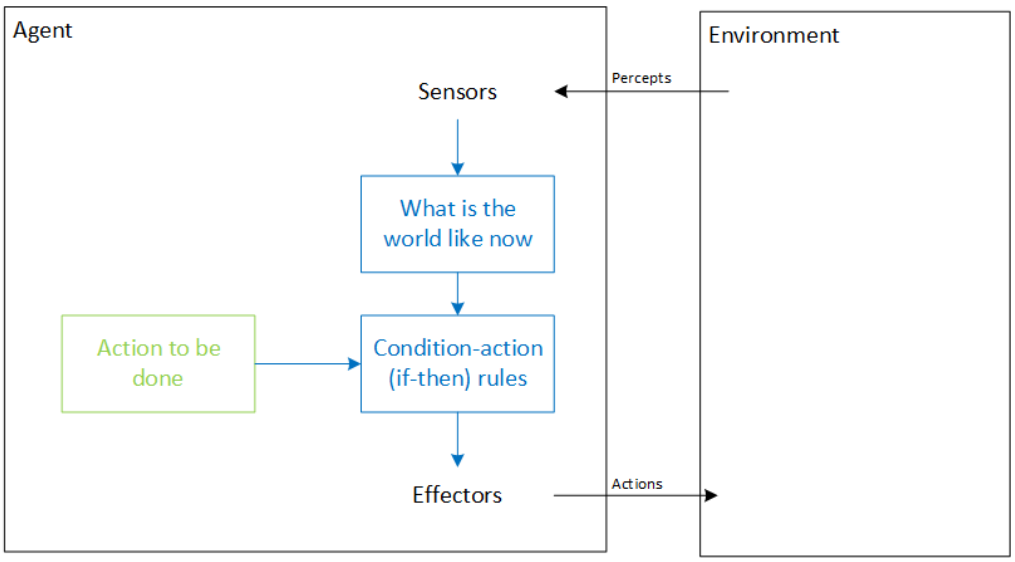

Figure 3.11: Rule based agent

Goal-based reasoning agent

The second reasoning architecture we review is the goal driven agent. It contains a more complex model, because it includes the goals in the reasoning. By knowing how the agent could influence the world and evaluating the outcome with the goals of the agent, the agent chooses the set of actions which best fit its goals. Although this comes closer to the human reasoning method, still the architecture contains no learning mechanism. If the agent is wrong about how its actions will influence the environment (or about the influence of external factors), the agent will never adjust this error and make the same mistake over and over. It is more flexible than the rule-based architecture, but not as adaptive as OODA-loop model and in a dynamic environment it will eventually start making errors.

Learning reasoning agent

Finally we review the learning agent. It is the most complex of the three models and is characterized with a learning component and critic. The critic is used to evaluate the performance of the model. The actions (as a result of the decisions made based on the standing conceptual model) have resulted in a new state of the world and the critic is able to evaluate the new state in terms of performance. The learning components uses the performance scale to adjust the conceptual model so it would hopefully perform better in the future. The learning agent is the most complex of the three reviewed models and most capable of adjusting to the

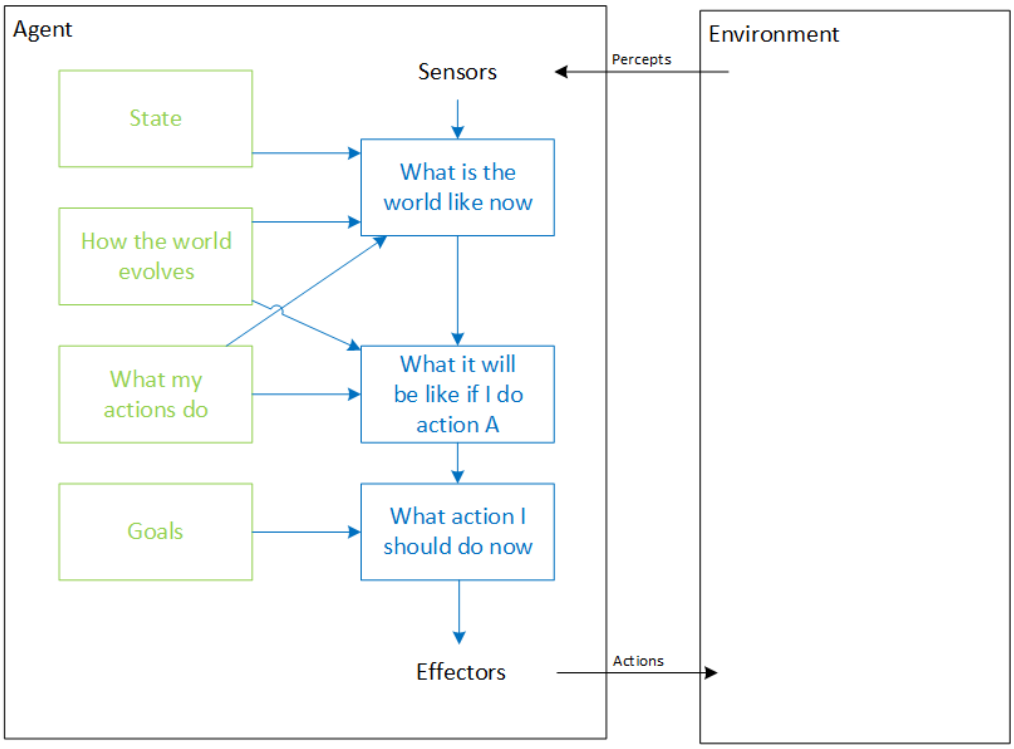

Figure 3.12: Goal driven agent

changing environment.

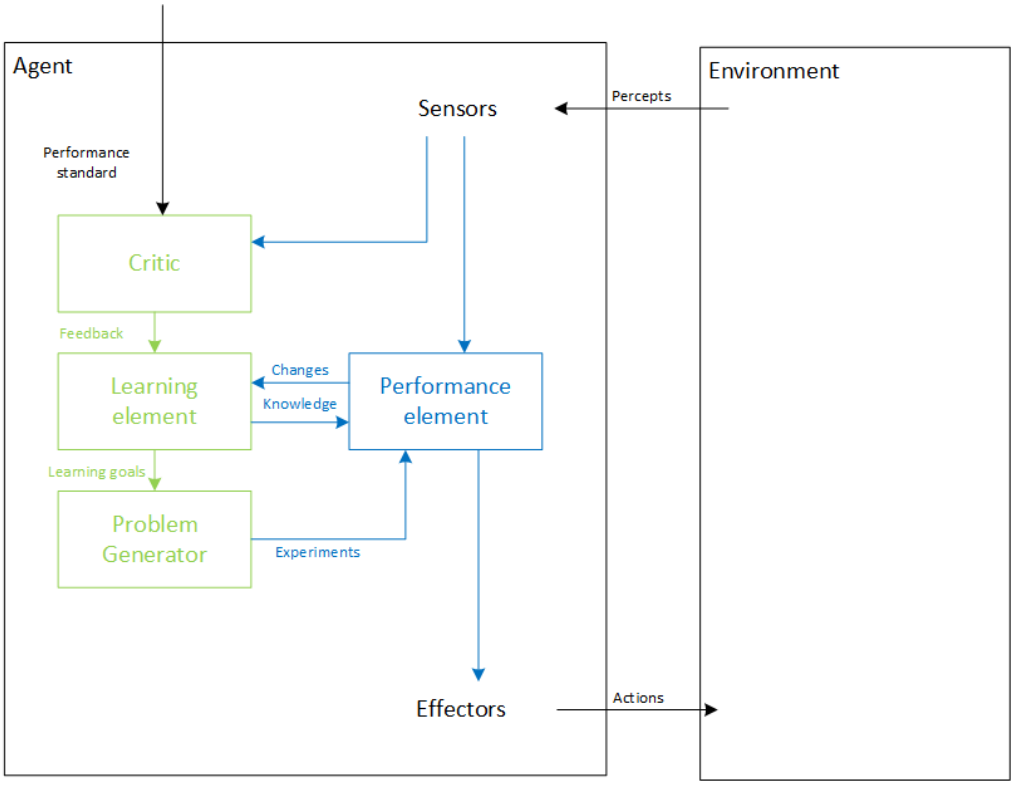

Figure 3.13: Agent with learning capabilities

The remaining question is which of these models would perform best. Unfortunately, there is not one single answer to that question. Performance is first of all measured in terms of correctness, meaning the lowest error rate. All systems depend on how well the sensors can sample the environment. Apart from the sensory capabilities, each system has its own dependencies for the correctness. The error rate of the rule-based architecture depends on the initial rules. For the goal based architecture, the error rate mostly depend on how well the predictions of the environment reflex the truth. The learning architecture is the only architecture that would really adapt to the chances in the environment, but the error rate depends on the initial settings and the extend to which the learning component can actually improve upon the error rate. Critics of gradient models says that such systems can only garantee to find a local optimum, not the global optimum. This would mean there could always be another system (such as a rule based system) that could outperform the learning architecture.
The OODA-loop model is an example of learning agents. While it is able to learn and reasoning on a high level of cognition, it is possible for complex mechanism to apply a more simple reasoning approach to a task (as shown by Rasmussen). For example, muscle reflexes in our body are solved within any interference of our higher cognitive thoughts. Keep in mind, that the human mind could (to some extend) learn to suppress these reflexes if it is aware of them and able to predict that they are coming. Thus, the human mind could adjust the reasoning method to our needs.

3.4.3. Task complexity and error rate

Finding the right approach for a task is a creative process, since there are often many reasoning methods to conclude the same solution. Earlier we saw how the chosen reasoning complexity for the problem could influence the performance of the security guard. We mainly described performance as the level of mistakes made. In this section, we will see how reasoning complexity also affects the performance of the system.

Mental workload and multitasking

Security guards have to perform many small tasks at once. They have to track multiple cars, classify and predict their behaviour and decide the proper actions. At some point, some of these tasks will be ignored due to the workload and cases of unwanted behaviour is bound to be overlooked. If we could identify what tasks are hard to combine or introduce relatively much workload, then they would form the candidate for automation. Unfortunately, it is hard to find an objective measurement of mental workload for a single task, due to the large difference between the way people reason. Since everyone reasons differently, equal tasks can be easy for some and hard for others, depending on their approach. Wickens investigated the effects of multitasking on the workload and has constructed a model to show how much combining multiple tasks affect the mental workload [76]. Readers that are known to his work will immediately recognize the 3D cube to symbolize the different parameters in the model (figure 3.14). The input parameter indicate the type of sensory input they require. The stages input indicates the processing stage the task requires, rather similar to the stages we saw in the OODA-loop model. And finally it includes a reasoning parameter, which indicate the type reasoning involved in the task. Wickens found that when combining tasks, that require similar resources affects the mental workload. The more two tasks share the required cognitive resources, the more the mental workload increases. For example, having to track many cars at once or having to guess the intensions of many subjects.

Attention and vigilance

The last concept we will discuss on the topic of human cognition is the attention and vigilance. In the previous paragraph, we already discussed the mental workload and the chance of overloading. Likewise the guard is in danger of a vigilance decrease. Computer analogues of human cognition are helpful, but it gets to a point where the analogue no longer holds. A decreased vigilance due to boredom is a good example. Unlike computers, human operators can become bored after performing repetitive tasks over a long period of time. The vigilance decrease will eventually result in detection misses. Having an automated system guard and notify easily overlooked items would be of great assistance. Basically, the mondane tasks of tracking and low-level reasoning would assist both by reducing the mental workload and remain vigilance during relatively boring intervals of the watch.

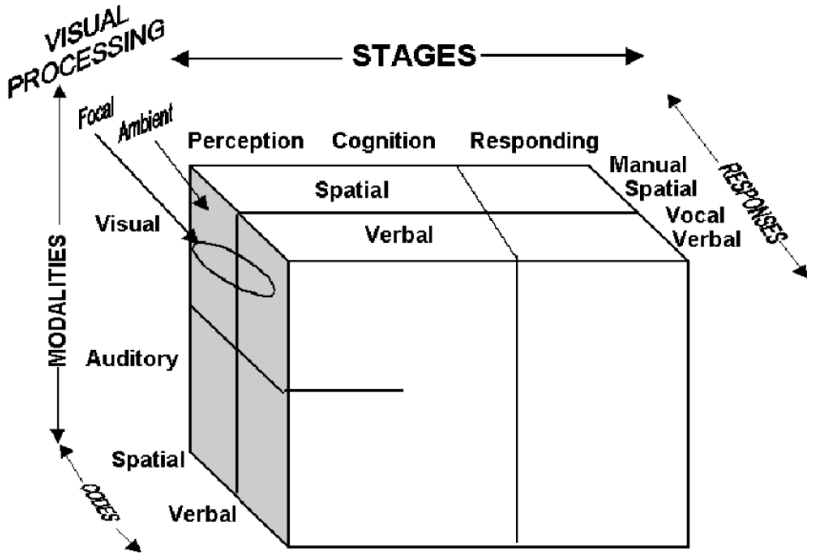

Figure 3.14: The 4-D multiple resource model

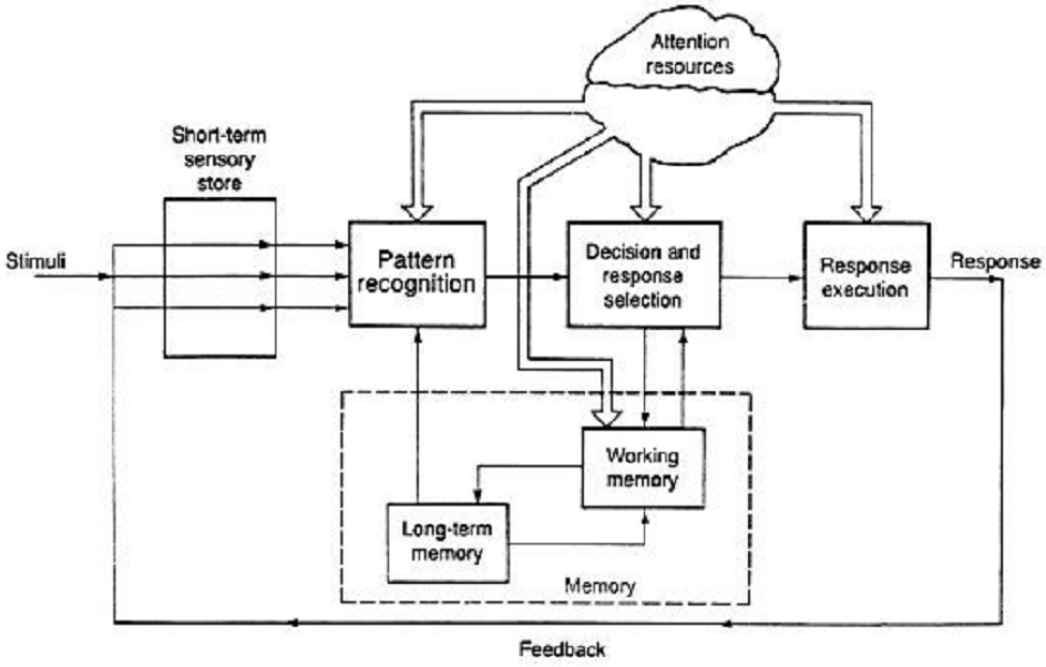

Figure 3.15: Cognition attention resources

3.4.4. Man-Machine interaction

After evaluating the cognitive model of our security guard, we can draw a number of conclusions. John Boyd showed the need to quickly adapt to the chances in the environment and the need for quick action. At the same time, the number of mistakes should be minimized. Faster reasoning can be achieved by applying another (often simpler) reasoning method and it is up to the creative learning process of the guard to find these new reasoning methods. The overal performance of multiple tasks are particular hard when the task share a cognitive resource demand. Especially tasks with a high demand for working memory tend to fail when the attention and vigilance decrease due to overload or under-load (boredom). Using the knowledge we gain here, we should be able to find the best candidates for automation.
The guardian angel principle states that we should assist the guard, rather than replace him. The guard will

not be able to watch all the monitors at the same time. Small incidents will be hard to monitor, due to the large amount of info presented to the guard. At the same time, tracking all cars is also near impossible for an human operator and is usually done through back tracking after an incident. Real-time tracking would ease the work tremendously. For now, we can stick to the simplest reasoning method applied within the software agents. We will stick to the rule-based approach and if proven successful, the system can be upgraded to more complex reasoning methods. The distributed nature will make the system complex enough to create within this thesis. Still, such a system would already be of great assistance to the guard and higher levels of reasoning will not be necessary until the rule-base system proofs insufficient.

3.5. Multi-agent systems

The design of our 3GSS is based on the concept of smart cameras. The smart cameras will be implemented using cooperative software agents. The cooperative effort of the agents result in the detection of suspicious behaviour. Software agents are processes with an advantaged level of autonomy and are aware of their spatial distribution. The stability of the individual agents is independent of the other agents. It is possible agents depends on input from other agents, but it features as sensory input and will only affect the reasoning within the agent, not the stability of the agent. In this research, the principle of multi-agent design is considered to be a design concept. The advantage of agent based design lay in the ease of design and analysis on a functional level. It allows for the system to be described on a functional level, without discussing the technical details required to realize this complex system of parallel processes.

3.5.1. Hierarchical agent design

Multi agent system is a powerful approach for solving all types of tasks. But it the complexity soon grows, when adding more and more agents. Keeping control and ensuring the performance of the system is hard. The growth in complexity is due to the exponential growth of the number of states the system is in. A few agents working in parallel soon create complex behaviour as a group and the number of possible outcomes become to big to control. Homogeneous design of the agents may help to cope with this growth, but the number of states still grows exponentially.
The complexity of multi agent reasoning systems is related to the cognitive reasoning models. In comparison to the single agent system, agents in the multi agent systems each have a limited view of the environment. The agents must work together and share information to gather the evidence required to classify the behaviour. Simple localised behaviour can be detected with the information of just one camera, where as complex behaviour requires a track over multiple cameras. The complexity of the behaviour classification is related to the amount of agents involved in the reasoning.
Simular to the mental workload model of Wickens, agents may form a bottleneck in the reasoning process. Agents that are responsible for processing large amount of information, could clog and slow down the whole system. Well designed multi agent systems are able to spread the workload as much as possible. This increases the scalability of the system and the speed of the classification. In relation to the findings of John Boyd, the system performs best when the reasoning is kept simple and well distributed over the agents.
This leads to an adjusted reasoning model, we will use as a guideline for the design of the multi-agent reasoning system. The reasoning model is applied at design time and the agents will not learn during the executions of the process. The reasoning model for each agent is shown in figure 3.16.

The most important aspect of the reasoning model is the decision on the amount of evidence that is required for this classification. In this decision, a trade off is made between the classification error and the execution time. Distributing the data will require more processing power from a limited resource. If the error decrease is not significantly reduced with the respect to the added computation time, the reasoning will be performed locally. Unfortunately, the trade off is never fixed and has to be evaluated for each classification problem.

3.6. Generic processing pipeline

In line of the effort to find the right reasoning method, we want to understand the differences in reasoning methods applied by man and machine. For the design of automated suspicious behaviour detection system, we define a generic processing pipeline, applied by both human operators and software agents. The steps

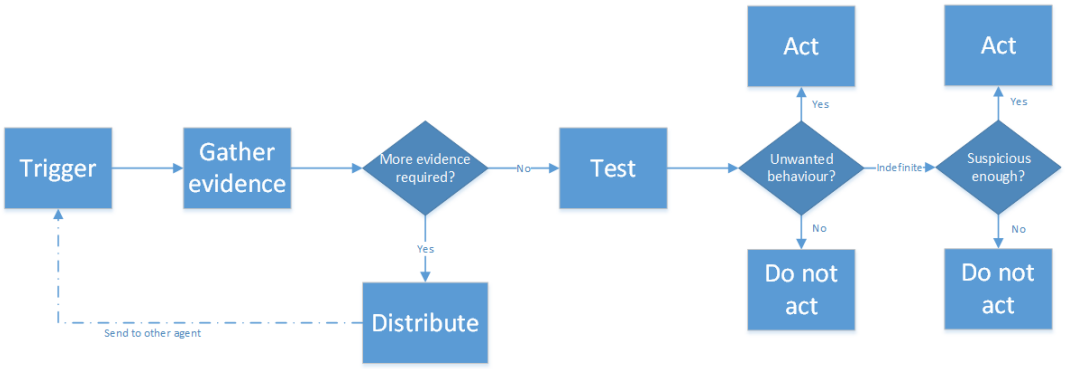

Figure 3.16: Agent design reasoning steps

within the generic pipeline have been individually discussed in the related work. The generic pipeline is based on the idea that each agent (human or software) gathers and extracts all the required information reasoning about the suspicious behaviour. The extracted information from one agent can serve as a trigger for another agent. For the system to operate, it should be clear how the tasks are distributed between man and machine. Once defined it is important to define how the information is communicated between the parties. The human cognitive process considers this step as a creative process, which continuously searches for more information. For the automated process, the design is reshaped to a static pipeline with triggers to start the information extraction process. In the related work, we already found the three (pre-)processing steps for automated VSS: detection, recognition and tracking. These steps are shown in figure 3.17. Here we will discuss the relation between the human approach and the automated approach. This will form the basis for our model of cooperation within the system and between the system and the operator.

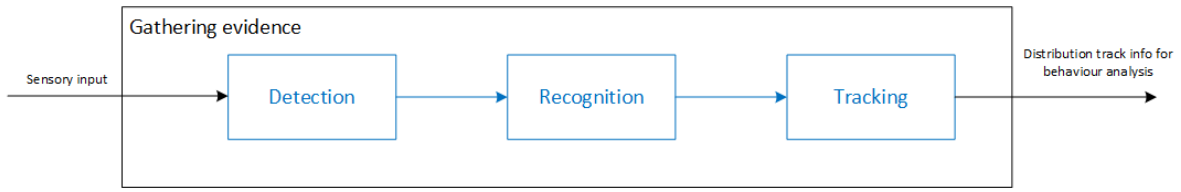

Figure 3.17: Processing steps in automated visual surveillance systems

3.6.1. Detection

The contrast with the human processing method, the automated detection of cars is far less flexible. We saw in the related work, that the detection of cars required methods including noise filtering, lighting settings and shape properties in order to detect the cars. The automated detection system has to be told very specifically what to look for and tends not to look for a conceptual understanding of the car. Deep learning techniques do challenge us to consider when machines are thinking on a conceptual level [40]. However, such techniques require large datasets and computation power in comparison to rule-based systems. Fast and efficient DSS still rely on simple techniques to process the data and in our case detect the cars.

3.6.2. Recognition

Although a lot can be told from a single image of a camera, but (car) behaviour includes all the actions within the region. In order to analyse the behaviour over time, the security guard must track the car over multiple images and cameras. Looking at the theory of attention from Wickens, we find that tracking a car requires relatively much working memory, meaning it is hard to track multiple cars at the same time. Also, the raw sensory information does not tell the specific car, its location or speed, but it just represents the pixels in the image of the camera. We have to go up to a conceptual level to understand the images showing a specific car at a location and it requires processing before other features can be extracted. All of these components combined way heavily on the mental workload of the security guard and will affect his attention and performance. Creating an automatic tracking system would already be of great assistance to the security guard.

3.6.3. Feature extraction

The track is the automatic result of the recognition process, which supplies a set of locations over time. Operators use this track to extract information relevant to the behaviour analysis. Similar to the problems within detection, extracting the track information itself it no problem, but monitoring many cars would create a heavy workload. And as we saw with the recognition process, the smart cameras need the detections from multiple cameras to extract the information. We found that the following features are used the most to analyse the behaviour

Location

The most basic feature of the detections is the location of the detection. Every location holds its own set of rules and regulations. The fact that a car is detected at a location already raises a suspicion of the intentions of the driver.

Speed

Every road in the Netherlands has a speed limit. The feature already allows the detection of unwanted behaviour. A speed of zero shows the car has stopped and maybe even parked. The speed of a car also allows the system to predict future behaviour.

Acceleration

The acceleration gives an indication of the intentions of the driver. Drivers who are in a hurry tend to have many speed changes. Patrolling or scouting cars would have low acceleration.

Heading

The heading shows the lane in which the car is driving. It is used to validate the car is driving on the correct lane or if it is honouring the one-way street rule.

Rotation

Like the speed, the difference in heading over time can be calculated. The rate in which the car rotates gives insight in how aggressive the car is driving. The higher the speed, the lower the expected rotation speed.

Duration & distance

The duration of the track from start to finish shows how goal oriented the driver is driving and how well the driver is familiar with the environment.The duration can be measured in driving distance as well as travelling time. We assume the (uninterrupted) duration is low, while the distance between start and current location is high. If the car shows different behaviour, the car is driving around longer than expected. This could mean the driver does not know the area of is looking for weak spots in the security. Either way, it is suspicious.

The reasoning is also influenced by the scale and time frame used to analyse the behaviour. For example on a small scale, cars may need to avoid obstacles or navigate their way through a maze of one way streets. For the large scale analysis, these turns will not affect the conclusion, because the track does not divert much from the starting position to the goal position.
All of these aspects must be combined to find the best viewpoint to classify the behaviour. And like in human reasoning, the best would be the reasoning process that optimizes based on processing time and error rate. We want to find the set of features that is easiest to extract and process, while the result has the lowest percentage of misclassifications. This is a different combination for each behaviour.

3.6.4. Task delegation

Dynamically assigning the tasks is complicated in an automated environment. The creative process of gathering evidence performed by the human operator is a highly complex mechanism. For the software agents to perform similar actions, all agents need to be fully aware of the context and available data in the environment. This would require a level of awareness and resourcefulness of the software agents that is too complex for this research. Instead, the processing pipeline is predefined and the roles and tasks of the agents will be determined on design level. The gathering of evidence is a static, predefined pipeline and the creative process is performed beforehand by integrating the expert knowledge into the pipeline design. Detection, recognition and feature extraction will be the best methods to reduce the workload of the guard. Simple behaviour analysis will have to stay vigilance.

3.7. Scenario definitions

The case study for this thesis is the security management of the training facility of the Royal Dutch Navy at Den Helder. This section describes the type of behaviour we can expect. We created a list of expected scenarios in the environment, separated as approved and unwanted scenarios. This list is by no mean exhaustive, but it should cover the most typical scenarios in this environment. Using these scenarios, we will discuss feature extraction in this case.

3.7.1. System components

The system itself contains 3 global components: *cameras*, *monitors* & *classification system*. The classification process is an addition to the existing system and will not replace the features. It will gather additional information about the behaviour of the people in the monitored environment. The classification component is designed as a distributed system. Each camera will have a process will have a dedicated process for distributing the extracted information from the camera images to other processes designated to perform parts of the classification process. The distributed system is an agent-based design and will include cooperating software agents with a shared task of analysing suspicious behaviour. The cameras communicate with the classification process by the agent designated to each camera.

3.7.2. Typical behaviour

The environment can be roughly divided into two regions: restricted and non-restricted. The restricted region is located on the east side of the Willemsoord harbour and is accessible for Navy personnel only. The main occupation in this area is related to training and education of future officers. The region includes lecture rooms, offices for researchers, strategic planning facilities, offices for supporting personnel, sleeping quarters and public places for ceremonies.
The non-restricted region of the environment is open for public use. It is located on the west side of the map. There are a whole range of activities that can be expected. The main activity for visitors is to visit the navy museum. Apart from that, the parking spaces could be used for anything within a walking distance, including the ferry to Texel, site-seeing, entertainment within the harbour and even visiting the city centre of Den Helder. The main activity for students is to attend the classes or visit the sleeping quarters.

Looking at the map in figure 3.18, the two regions are separated by buildings, fences and access gates. Any intruders would either have the climb over a fence or navigate around an access gate and no one could enter the restricted area without noticing it. The access gates are opened by the key pass or remotely by the security guards. Cars will only be able to enter if the access gates are open.
Over the complete region, the regular traffic rules apply. For the case scenario, we apply a more strict interpretation of the traffic rules and state that is only allowed to parking within the parking spaces. People are expected to park their car and walk to their destination, without to much detour from their route. Especially within the restricted region, everyone is there for a reason. The approved personnel are expected to take an intentional route to their destination and will unlikely take the time to go for any site-seeing.
The typical behaviour changes over time, due to the visitor hours of the museum and curfew of the students. Visitors mostly there for entertainment and if the weather is a demotivating factor, less visitors are expected during that time. Also the visitors will need spare time from work to visit. This shows that the expected time

Figure 3.18: Navy Facility with restricted and non-restricted regions indicated

for visitors to visit the environment is during holidays, with nice weather and after work hours. Of course the exception is made for special events, such as the harbour days or public activities of the navy. The training facilities of the navy work with regular time tables and also have workdays, weekends, holidays and office hours. In fact, students that are stationed at the facility are under a night curfew. Everyone leaving or entering the restricted region after 0:00 AM must report to the security guard and can disciplinary actions to be taking by their officers. As a result of this, the expected behaviour of the students chances over the course of each day. During the day, students are expected to follow courses or study. Student can arrive and leave as they please. Student usually go outside the region for dinner and can stay there until they decide to go back. Over the course of the evening student will arrive back from their activities and will drive to the sleeping quarters. Once the night curfew is up, students are no longer expected to arrive in the region.
Based on the typical behaviour, we were able to define series of scenarios that form the basis for the use cases in the automatic surveillance system. The scenarios are separated in approved and unwanted behaviour.

3.7.3. Approved behaviour

The scenarios of approved behaviour indicate the scenarios that are expected to happen and are considered normal. The total of eight scenarios cover the most likely approved scenarios. These scenarios combined still do not cover the complete set of behaviours, but give a good first impression of the likely behaviour.

Scenario 1: Students driving to their classes

The officer students attend classes in the designated buildings. They tend to park their car in the area of these buildings. Sometimes the students walk from their sleeping quarter to the classrooms. Since we are looking at car behaviour, walking to the classes is not analysed.

Scenario 2: Students drive to their sleeping quarter

Navy officers in training are often required to stay at the facility during their training. The training facility include sleeping quarters. The students are expected to sleep here and be on the premiss before midnight. Typically, the students arrive before their training start and park the car near their sleeping quarter. During their spare time, they are allowed to come and go as they please, as long as they are back before midnight.

Scenario 3: Tourists parking to visit museum
The navy museum is located close to the training facilities. Many people visit the museum during the day and park their car near the entrance of the museum. The visitors and students sometimes share the parking spaces, but only if there is nowhere else to designated parking spaces left for the students.

Scenario 4: Commandeur driving to strategic offices
The VOKIM headquarters can serve as a strategic headquarter. During this time, commandeur is expected to arrive and depart regularly. The cars of the commanding officers are authorized to drive to the entrance of the office and ignore the one-way driving lane restrictions.

Scenario 5: Teachers driving to lecture room
Teachers will arrive and depart for the lectures and other activities. The behaviour is similar to the students arriving to follow courses.

Scenario 6: Tourists parking to visit other events
The main attraction on the premisses is the museum, but the parking spaces are relatively close to other activities such as the cinema, bowling alley and city centre of Den Helder. It is still a relatively long walk to these attractions and they all have their own parking places. This is unlikely the visitors will not visit the museum, but it does happen.

Scenario 7: Taxi's dropping off students
The arriving students could arrive by means of other transport. The students could for example use the public transport to get to Den Helder and continue by taxi or have a drink in the city centre, depriving them from the opportunity to drive themselves. The taxis are not allowed on the area for authorized personal and usually drop the students off at the gate. When it is inconvenient for students to walk the remainder of the route, the taxis sometimes use the students authorization to continue on the closed off area.

Scenario 8: Museum employees arriving and leaving
The car behaviour of museum personnel is very similar to the behaviour of the visitors. They arrive and drive to the parking space nearest to the personnel entrance. The only difference is the knowledge of the environment, which allows them to drive straight the target parking space, without much navigation involved.

Other scenarios
Apart from the eight scenarios, there are numerous exceptions. These exceptions include (but are not limited to) training on the parking lot, supplying of gift shop, supply and gathering of ammunition and maintenance. These situations divert from the regular behaviour and will most likely be misclassified by any system. Because these incidence happen very few times, the misclassification is accepted in these cases. In fact, the exceptional behaviour is partly suspicious and misclassification will keep the security guard vigilant.

3.7.4. Expected unwanted behaviour

Next to the typical behaviour, the environment also has behaviour that is unwanted for the region. The detection of (precursors of) this behaviour is the main task for the surveillance system. Like the list of approved behaviour, this list is not exhaustive and many other actions could take place on the premisses. Our task for now is to map the most common ones. The most common scenarios for unwanted behaviour are:

Scenario 1: Drivers ignoring rules to take a shortcut to the cabins
Students in a rush to get to their destination, tend to ignore some rules. These conveniently ignored rules include: one way streets, no driving zones, no parking zones and speeding.

Scenario 2: Terrorists stealing from the armoury
The navy armoury is the storage for many items that should not end up in the wrong hands. We mainly focus on organized crime, stealing complete boxes of guns and ammunition. Stealing such amounts requires cars capable of transporting the weight and will most likely be parked near one of the exits of the armoury. Large cars parking near the armoury will likely cause suspicion for the guards, which the criminals will avoid as long as possible. Other suspicious behaviour such as speeding or ignoring one way streets will be postponed until the criminals suspect they are detected.

Scenario 3: Exploring the area with bad intentions

Intentional crimes require planning and knowledge of the area, if the criminals attempt to increase the chance of success. As part of the preparation, knowledge of the environment such as camera locations, exit routes, alarms and routines are just as valuable for the criminals as it is for the security guards. One method to obtain the information is by scouting the area.
The scout will attempt to blend in with the other behaviour, not to arouse suspicion with the guard. A prepared criminal will be hard to detect, because it will not break any simple rules, such as disobeying traffic rules. The difference in behaviour of the criminal is the goalless driving. Anyone with a destination in the area will take the shortest known route to the parking space near the destination. The duration in relation to the absolute travel distance will be low compare to goalless driving. Anyone with a relatively long duration time of the route is suspicious in this scenario.

3.7.5. Illegal actions

The scenarios describe the most threatening course of actions for the environment. At the same time, anyone could disobey the law for any number of reasons. We list a few illegal actions that may or may not be part of an intentional crime, but are illegal non the less. They are less complex to detect and should on itself cause an alarm.

Illegal entry

Certain areas are off limits for any car. Driving there will cause an immediate alarm.

Illegal parking

Certain areas are designated for parking. Everything outside these areas is considered a non-parking area.

Speeding

The speed limit depends of the location. Acceding the speed limit should raise an alarm.

Ignoring one-way street

Driving the wrong way in an one-way street is a clear illegal action.

3.7.6. Simulation

The automatic surveillance system designed for this research will not be applied to real life surveillance and will use a simulation environment to simulate the car behaviour. Complete implementation of the automatic surveillance system include multiple research topics, including image processing, camera mapping, car classification, image object tracing, user interface design, which would take several years. This research focusses only on the communication between the cameras and distributed reasoning. It looks for the design aspects involved in implementing a multi agent reasoning model in an distributed environment where the behaviour can only be classified using multiple sensors. Real life cameras bring quite some practical problems and chances are the classification would already introduce errors. Image processing, 2D-to-3D mapping and the tracking problem is reduced to the minimum to ensure the collected data is as accurate as possible before we apply reasoning. This accuracy is only guaranteed in a simulation environment.
The downside of using a simulation is the effect on the reasoning pipeline. The reasoning methods will always be designed specifically for the input. It can only be used for the real life situation when the simulation represents the actual environment as much as possible.
The goal of the simulation environment is to come as close as the original case as possible. The simulation will include all the scenarios described in the previous section. The image processing is minimized by generating a very simple environment, were the car is immediately spotted within the image. The mapping problem that includes calculating the position in a three dimensional map from the images is completely avoided by placing the cameras straight above the road. Each camera still needs to know where it is placed on the map.
A problem with using a simulation environment is the data gathering. Since it is a simulated environment, no data can be collected to form the statistics for the reasoning system. All of the knowledge used in the model will come from the analysis done so far in this chapter.

4

Simulation & tools

The previous chapter defined the knowledge context of the intended automatic suspicious behaviour detection system. The next step is to define the tools to build it. Numerous tools will be used to build this system including simulation tools to represent the environment and supply the camera images. In this chapter, we will discuss these tools and show how the simulation environment is extracted from the Royal Dutch Navy training facility.

Figure 4.1: Layout of the used tools

4.1. Project aspects

The goal of this thesis is to design an automatic suspicious behaviour detection system, using multiple intelligent cameras. We separate the project aspects into four categories: behaviour analysis, world simulation, automatic behaviour detection and testing. These categories are shown in the overview in figure 4.2.

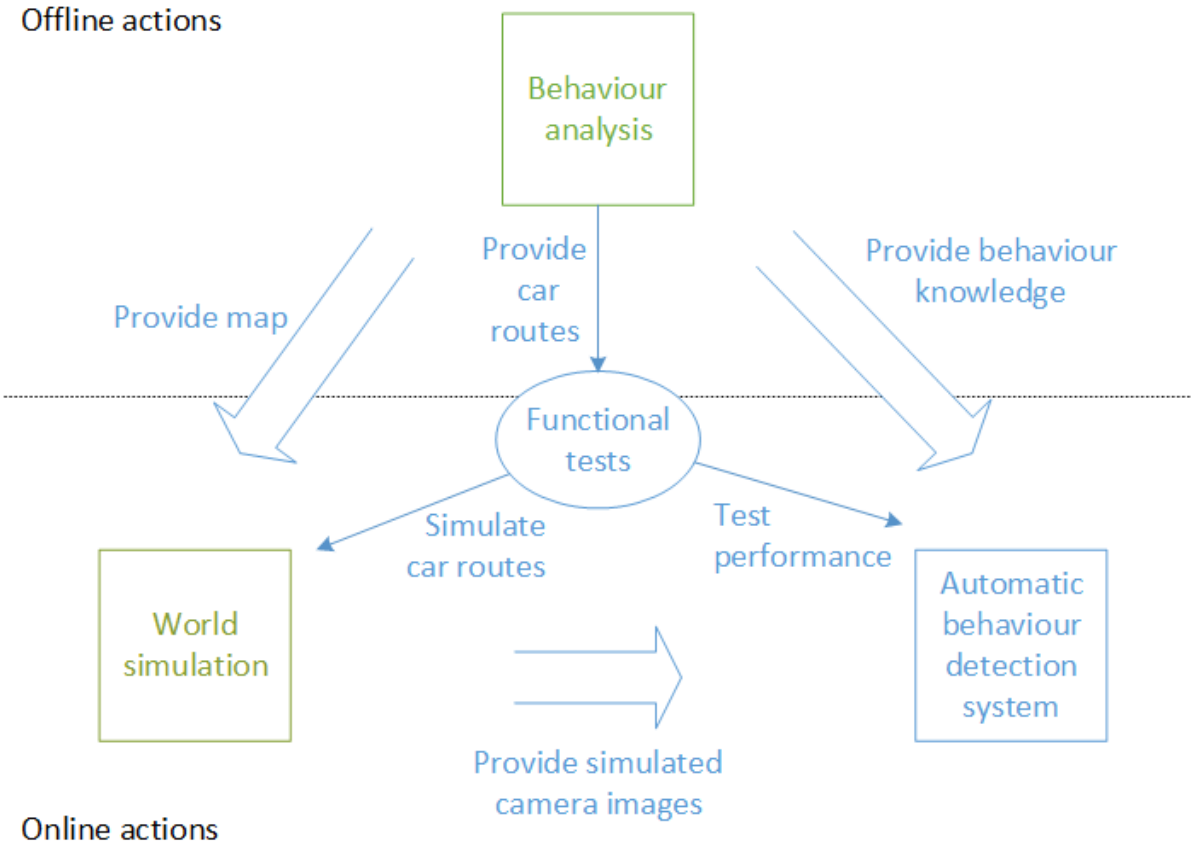

Figure 4.2: Project components and interactions

First and foremost, we will using a simulation environment for the car behaviour. The map structure is inspired by the map structure of the facility in Den Helder, but will lack the resolution and some the practical problems of the real-life situation. Real-life environments are influenced by many problems, including dirty cameras, weather conditions, camera alignment, etcetera. Using simulation environments ensures the problems do not occur and allows us to work from the reasoning model outwards. We will build a platform for multiple cameras to cooperate and reason about the behaviour in the environment. Once the platform is proven successful, it can form the basis for designing the real-life implementation of the environment. It allows us to focus on the reasoning and cooperation model, rather than spending time on the practical issues surrounding real-life cameras.
Using simulation instead of real-life environments does take away the chance to analyse the car behaviour. No data can be gathered from the historical videos to detect common routes or typical unwanted behaviour precursors. We rely on expert knowledge to define these patterns. The behaviour is analysed by building the map and simulating the behaviour in an analysis tool. The offline analysis provides us with three pieces of information. It creates the static representation of the environment, that is used for the world simulation. This map contains the structures represented as the simulation tool requires it. Internally, the behaviour analysis tool will use the knowledge of the map to generate car routes that can be used to simulate specific behaviour. The world simulation itself ensures all the virtual cameras have images available as-if they were actual cameras, looking at the environment. The images exist of both static and dynamic information, so that the automatic behaviour detection system will have to process the input images similar to the real-life situation. At the same time, the simulation does not have the practical problems we discussed earlier.

The automatic behaviour detection system is the main topic of this thesis. Within the system, cameras will work together to solve local, regional and global behaviour detection problems and will assist the security guard by providing him with the discovered information.
Finally, the systems performance is tested automatically, using test cases. The test cases require the world simulation and automatic behaviour detection system to execute synchronous in order to gather reliable test results. We decided to automatically test the system, since the result of this thesis will only be one chain in the shackle. Ultimately, the Royal Dutch Navy wishes to create the full system. In this thesis, we only lay the groundwork for building the multi-camera surveillance system and we wanted to be clear in showing what the software could and could not do for them.

4.2. Integrated development environments

The project tasks discussed in the previous section will be developed using a collection of tools. We will start by summing up the the integrated development environments (IDEs) used to program the code for the tools.

4.2.1. MathWorks MATLAB

The behaviour analysis will be performed in MATLAB. MATLAB is a high-level language and interactive environment for numerical computation, visualization, and programming. It allows for analyzing data, development of algorithms, and creating of models and applications. The language, tools, and built-in math functions enables exploration of multiple approaches and reach a solution faster than with spreadsheets or traditional programming languages. MATLAB can be used for a range of applications, including signal processing and communications, image and video processing, control systems, test and measurement, computational finance, and computational biology.
MATLAB will be used in this research to gather the knowledge about the common paths in the simulation environment. Due to the simulation environment, it was not possible to gather the common paths for the training phase. Instead, we will use a scripted system to calculate the characteristics of approved and unwanted behaviour. The knowledge is used to define the rules of the automatic detection of unwanted behaviour in the system. The knowledge is then used to randomly generate the test tracks.

4.2.2. NetLogo

NetLogo will be used to simulate the car's behaviour during the experiments. NetLogo is a programmable modelling environment for simulating natural and social phenomena. It is particularly well suited for modelling complex systems developing over time. Modellers can give instructions to hundreds or thousands of "agents" all operating independently. This makes it possible to explore the connection between the micro-level behaviour of individuals and the macro-level patterns that emerge from their interaction.
Even though NetLogo is commonly used for simulating large amount of agents and research the behaviour of cooperative agents, we will only use it to simulate a car and collect the images of the cameras. Since we are not using real cameras in this thesis, we needed a method to generate the images and simulate the processing pipeline. NetLogo allowed us to do so. NetLogo is also chosen for the potential for future projects. Although we only study the cooperation of smart cameras for the controlled environment of one car, future projects could simulate multiple cars in the same environment. NetLogo provides the processing pipeline with an image stream for a configured set of cameras, which can just as easily show the images of multiple cars.

4.2.3. Eclipse

The smart cameras, agents and software will be developed in a Java environment [20]. Eclipse provides IDEs and platforms nearly every language and architecture. It is known for the Java IDE, C/C++, JavaScript and PHP IDEs built on extensible platforms for creating desktop, Web and cloud IDEs. These platforms deliver an extensive collection of add-on tools available for software developers [18]. Eclipse allowed us to setup a runtime environment for the smart cameras. It is used to program and test the intended automatic suspicious behaviour detection and covers all that make the smart cameras smart.

4.3. Software & Libraries

The IDEs allow us to program the software, but we also need to program the pipeline. This section sums up the third party software and libraries will be used within the thesis to run the smart camera system.

4.3.1. Oracle Java

As discussed previously, the smart cameras will be running in a Java environment [20]. Java is a programming language and computing platform. Java code is written in .java file. This code contains one or more Java language attributes like Classes, Methods, Variable, Objects etc. Javac is used to compile this code and to generate .class file. Class file is also known as "byte code". The name byte code is given may be because of the structure of the instruction set of Java program. Java Virtual Machine (JVM) like its real counter part, executes the program and generate output. Because the code is executed on the virtual machine, the software is platform independent. Assuming the operation system has a JVM written for it, the software run the same on all devices (figure 4.3).

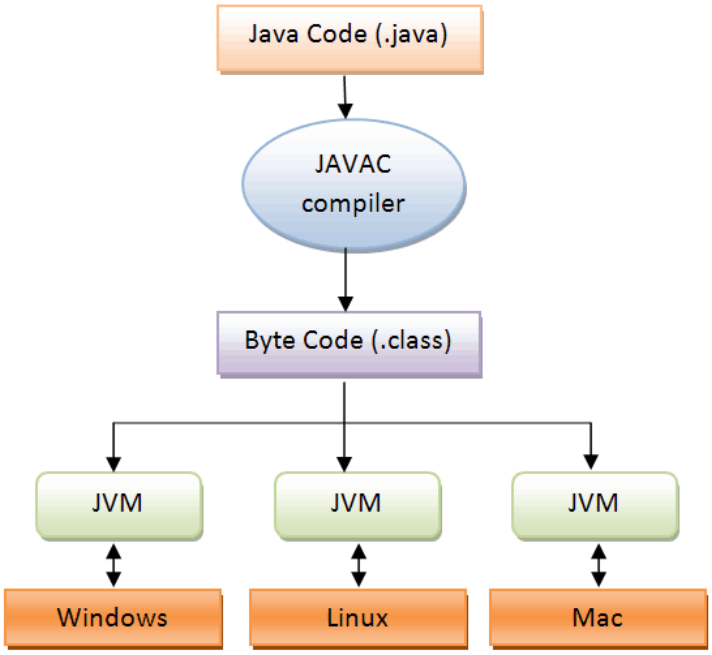

Figure 4.3: Java software is platform independent, due to the Java Virtual Machine made for many operating systems

The Java Virtual Machine (JVM) allows us to develop the software for any platform that supports it and can run cross-platform if needed. The smart cameras will be simulated, but designed for real world environments. Thus the system is required to have a JVM available. Fortunately, these JVMs are developed for a wide range of chipsets, including the Linux, which is the operating system of the Raspberry Pi [26]. This would make the Raspberry Pi an excellent choice for implementing the smart cameras.

4.3.2. CLIPS

The behaviour analysis will be performed using the expert rule based system CLIPS [17]. CLIPS is an expert system platform, designed execution rule based reasoning. CLIPS provides methods of initiating facts and rules and provides the process of reasoning. CLIPS is available as a Java Native Interface (JNI) and the reasoning engine can be managed from the Java code.
CLIPS is an expert system, which uses rules to deduct information about the prior knowledge. The knowledge base is filled with the facts of the world. When it starts to run the reasoning, CLIPS will go over all the rules that apply to the knowledge and update the knowledge base with the newly deducted information. The new information can again trigger other rules and CLIPS continues to run the epochs, generally until no more

rules apply. The CLIPS API allows us to perform these steps, as long as we provide the knowledge.
Facts with fixed structure can be defined using fact templates. The templates can be defined using the 'deffact' command. Facts with defined template are sure to have the same structure. We used the fact templates to ensure the Java classes and CLIPS facts could be exchanged.

4.3.3. JAVA Agent DEvelopment Framework

The design for the smart cameras will be implemented using an agent-based approach. JADE (Java Agent DEvelopment Framework) provides the software framework for the management of the agent's lifecycle and communication [19]. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set of graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same operating system) and the configuration can be controlled via a remote user interface. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language.

Agent platform

Besides the agent abstraction, JADE provides a simple yet powerful task execution and composition model, peer to peer agent communication based on the asynchronous message passing paradigm and many other advanced features that facilitates the development of a distributed system.
Within JADE, agents live on top of a platform that provides them with basic services such as message delivery. A platform is composed of one or more containers. Containers can be executed on different hosts thus achieving a distributed platform. Each container can contain zero or more agents [19]. The agents can communicate within the containers through local identification, but also across containers through global identification. The containers can be connected, froming a cross platform multi-agent system. We use this property to distributed the agents across the smart cameras. Even though the implementation will not run on real cameras, the transformation from virtual to real smart cameras only requires effort in configuring and addressing the containers on smart camera platforms.

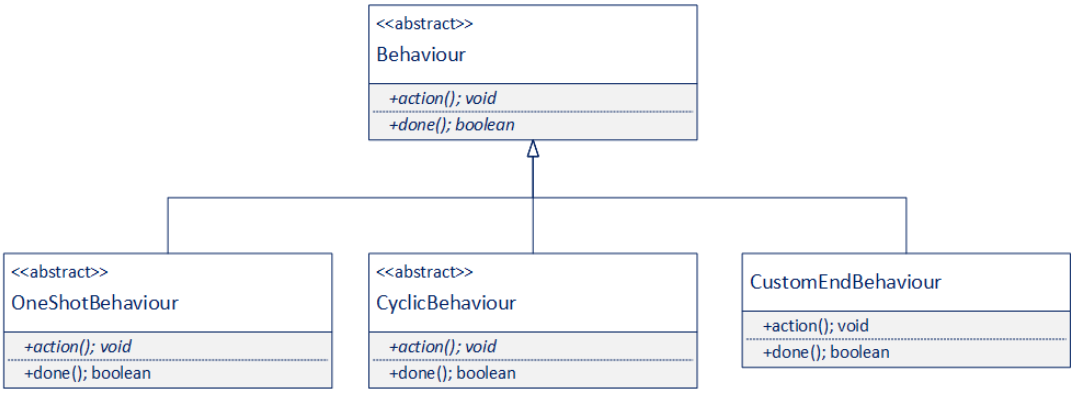

Figure 4.4: Cycle types of behaviour

Agent behaviour

JADE provides a framework for the agents to act and process the information. All agents can have multiple processes running simultaneously, known as behaviours. Behaviours are simular to Java threads, but have a custom implementation of thread, optimizing them for large amount of threads. With respect to the cyclic behaviour, there are three types of behaviour: one shot, cyclic and custom cyclic behaviour. The three types are shown in figure 4.4.
The behaviours contains sequential actions, performed within the *action* method. The cyclic behaviour is added with the use of a *done* property. The one shot behaviour runs once and is deleted after execution. Cyclic behaviour is designed to run indefinitely. The custom cyclic behaviour can be implemented as the

user prefers, but is particularly designed to be executed a finite number of time. Examples of such behaviour is acting on a specific way until an environmental events or behaviour with multiple phases. All agents behaviours are added to the behaviour pool and scheduled by the JADE framework. The path of execution applied by the scheduler is shown in figure 4.5.

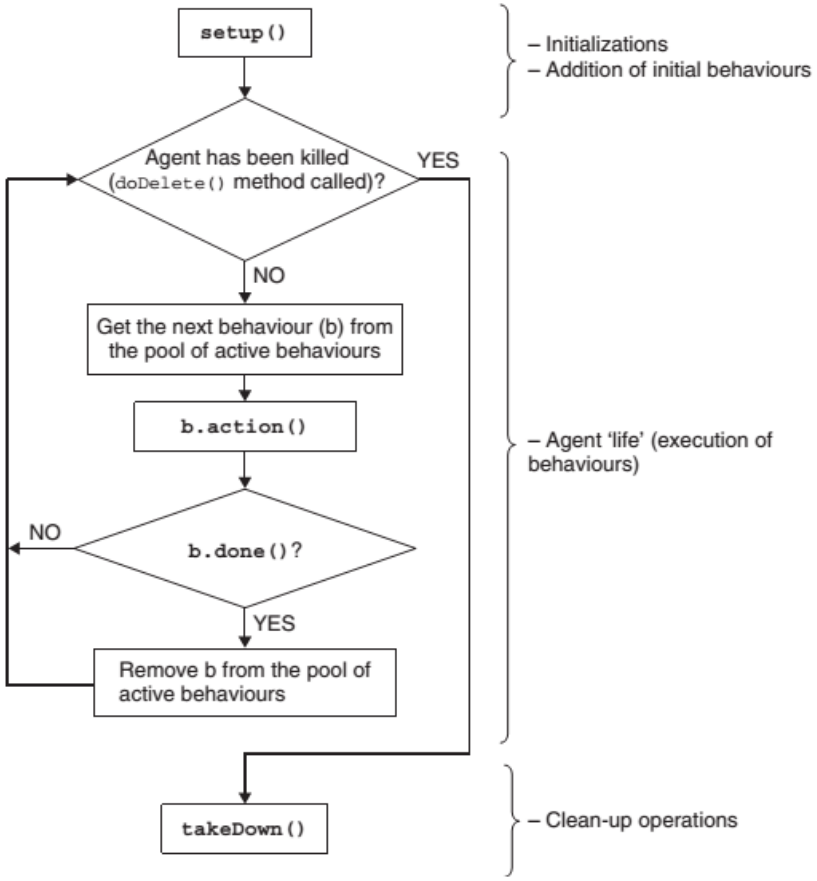

Figure 4.5: Agent thread path of execution

4.3.4. JUnit testing

The simulation, virtualisation and lack of graphical user interface make the intended automatic suspicious behaviour detection system a very abstract topic. To make the environment more tansionable and the project more manageable, we will automate the testing of the system. The tests will allow us to validate the scientific challenges set for this thesis. An often applied method for automatic testing of software is unit testing. JUnit is a unit testing framework for the Java programming language. JUnit has been important in the development of test-driven development, and is one of a family of unit testing frameworks collectively known as xUnit that originated with JUnit [22].
JUnit testing framework uses test cases to test the functionalities of the developed system. Test cases provide a starting point for the tests to perform and refer to what tests are performed within the test cases. The annotation are used to tell the test runner what functions are used. The annotation include (amongst others):

BeforeClass
Function is called at the beginning of the test case.

AfterClass
Function is called at the end of the test case.

Before
Function is called before each test in the test case.

After
Function is called after each test in the test case.

Test
Function is considered a test within the test case.

The tests are regular Java functions, will have the same relations with the classes under testing as the main function with have in normal program. Within the tests, the test developer can use the provided overloaded assertion methods for all primitive types and Objects and arrays (of primitives or Objects). The assertion methods validates the calculated value against the expected value. Using a collection of the assertion methods, the functionalities of the system can automatically be tested.
Testing Multi-Agent Systems (MASs) requires a different technique for testing. For JADE, the JADE agents will run on the JADE container on a separate application or even device. In [13], a framework is provided to test JADE agents within JUnit tests. The framework uses mock agents to test the agents under testing (AUT) through communicating the results over the normal channels. The test result is collected from the mock agents and finally tells the JUnit test if the functionality is working properly. The communication between the test environment and the mock agent is done with the JADE feature A2Ointerface, which stands for agent to object interface. The use of this feature is discouraged for most application, since it works against the autonomy property of the agents. In the suggested agent test framework [13] they make proper use of this feature, since the mock agents is not part of the functionality of the MAS. The mock agents are used as an entry point in the MAS and do not change the behaviour of the agents under testing. We will be using the testing framework suggested in [13] to build our own extension to the JUnit testing framework and be able to test the functionalities of the automatic behaviour detection system.

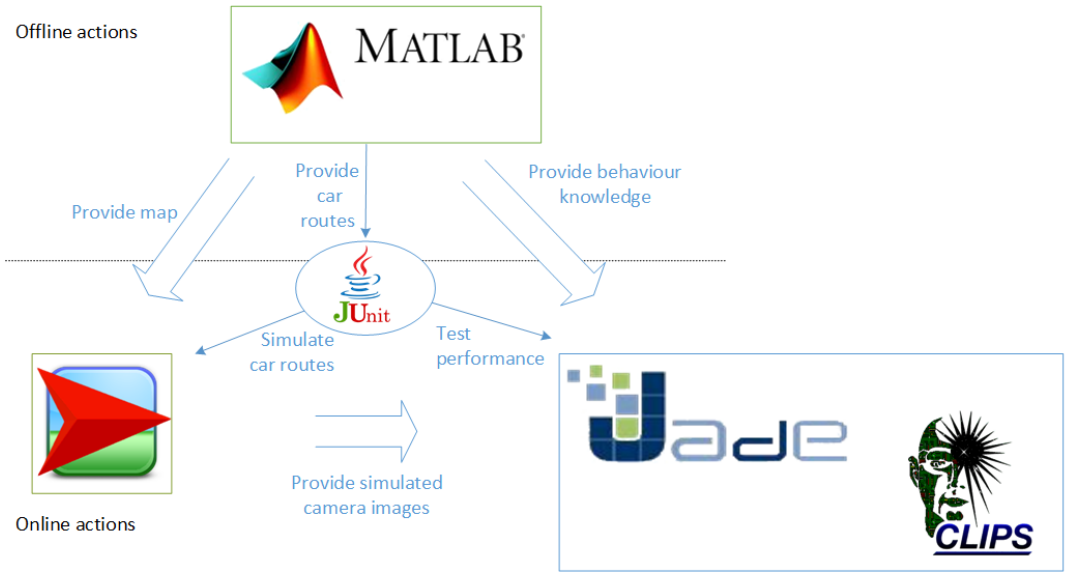

Figure 4.6: Responsibility and dependency of the tools

4.3.5. Interaction between the tools

We finish this section on software & libraries by describing the interaction between the software & libraries and provide an overall view of each tool's purpose. The diagram in figure 4.6 shows the used components and their interaction.
Matlab will be used for the offline analysis of the map and gathers additional knowledge about the car behaviour in design time. The outcome of these analysis is exchanged with both the world simulation and the automatic behaviour detection system. The world simulation uses the tracks generated by the Matlab analysis to simulate either the approved or unwanted behaviour. Within the automatic behaviour detection system, the knowledge about the parameters from the analysis are used to estimate the level of suspicion.
The world simulation (or NetLogo environment) acts out a test scenario and simulates a car driving along a configured track. While the car is driving, images are created for each camera and broadcast over a specific

TCP/IP port. In real world scenarios, the images would be send to the dedicated hardware within the smart camera, using a similar interaction.
Finally, the intended automatic suspicious behaviour detection system will run the 'smart' component of the smart cameras and runs the distributed cooperative agent platform for processing and reasoning. Each process linked to a camera will be assign the associating TCP/IP port and listens to the image stream arriving from the NetLogo world simulation. It uses the knowledge from both the experts and the offline analysis to estimate the level of suspicious and notifies the human operator of the findings by use of an graphical user interface. The user interface is part of the intended automatic behaviour detection system, runs on a control terminal, which for practical reasons is scheduled to be placed in the existing control room.

4.4. Simulation abstraction

The intended automatic suspicious behaviour detection is mainly created to study the processing structure of cooperative smart cameras. It will not be applied in a real world environment. The practical challenges in real life environments are beyond the scope of this research. At the same time, we saw how much the processing pipeline and measurements device affect the classification performance. There for it is important that the simulation environment represents as much of the case environment and goes through the same processing steps as real life systems would. We will now look at the simulation abstraction made of the case scenario and explain how the simulation represents the real world environment.
The real world environment is stripped to the essential components needed for simulating the behaviour we want to detect. The first result of the simplification was the reduction from a three to a two dimensional map. Previous work within the research group showed it was possible to map the 2-D images from the camera to 3-D coordinates [12]. Using the map given in figure 3.18, we composed a new map that would be easier to simulate and would be able to represent the scenarios used in the case.

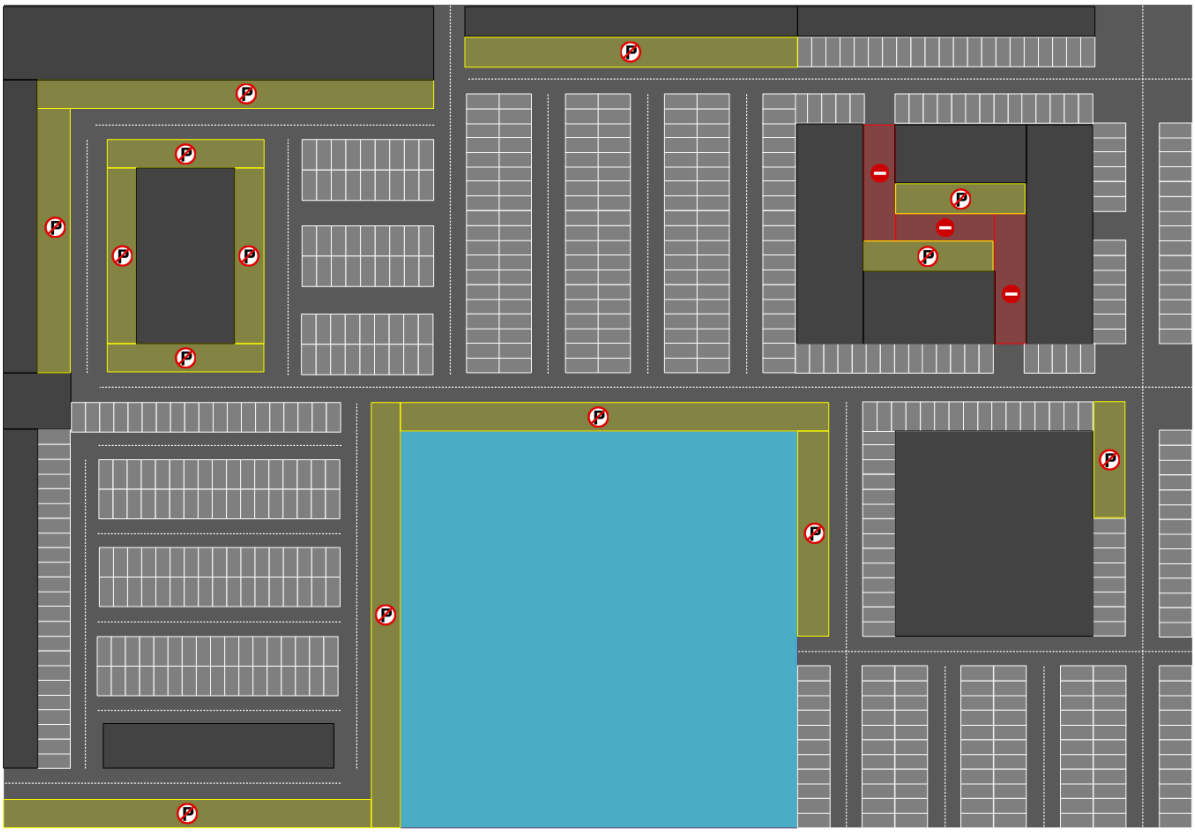

Figure 4.7: Simulation abstraction

The components that remain within the simulated map are:

1. structures
2. roads
3. parking spaces
4. cars

Other components in the environment are small local variation with no structural impact on the behaviour of the car. The behaviour of cars are naturally affected by this simplification. The roads and parking spaces are accessible for the cars, but the structures are not. The simulation environment must not only support allowed behaviour, but also must be able to simulate unwanted behaviour. This is implemented by giving all roads accessibility for the car. Roads in no entry zones (shown by the no entry symbols on the map) can still be driven on. No parking zones have similar configurations.
In real world situations, the cars can drive on roads in any orientation. The simulated car can only have one of eight orientations. When driving straight, the car is always heading north, east, south or west. Only when the car is making a turn will the car have one of the four intermediate orientations. The map originally did not have the grid like roads, so the map has been adjusted to fit these guidelines. Unfortunately, the access gates are no longer a part of the simulation. This would both overcomplicate the simulation environment. The intented VSS only contains visual information and multi modal sensory input is left out of the scope of this research. Since the identification system used for the access gates is not part of the VSS, we decided to remove the access gates all together. We also decided to keep the speed changes out of the speed for now. The car will only move one step or be parked. The result of the simulation abstraction is shown in figure 4.7.

4.4.1. Camera installation

The cameras will be distributed equally over the map. The mapping from camera images to environment map is more of a necessity than a scientific challenge in this research. We intended to keep the mapping as simple as possible. This is solved by placing the cameras directly above the simulation environment, looking down. The 2-D images map directly on the complete map of the environment, with the use of relative positions for each camera. The camera distribution is shown in figure 4.8. The JADE design with the containers of different platforms and Java virtual machines allows us to be able to port the simulated environment directly onto real environments.
We settled for 8-by-8 images for each camera, with cars the size of 2-by-1 pixels. The map is resized to a matching 72-by-56 locations, which gave us complete coverage of the map with the use of 63 (virtual) cameras.
In order for the simulation environment to properly imitate the situation in the real world environment, the simulated cameras should have the same limitations as the hardware cameras will have. We assume the cameras are connected to dedicated hardware that is able to stream the images, perform the processing and communicate with other cameras through a standard TCP/IP network. Although the simulated environment will not have this hardware, the software is designed to work on both the real world environment, as well as the simulated environment.

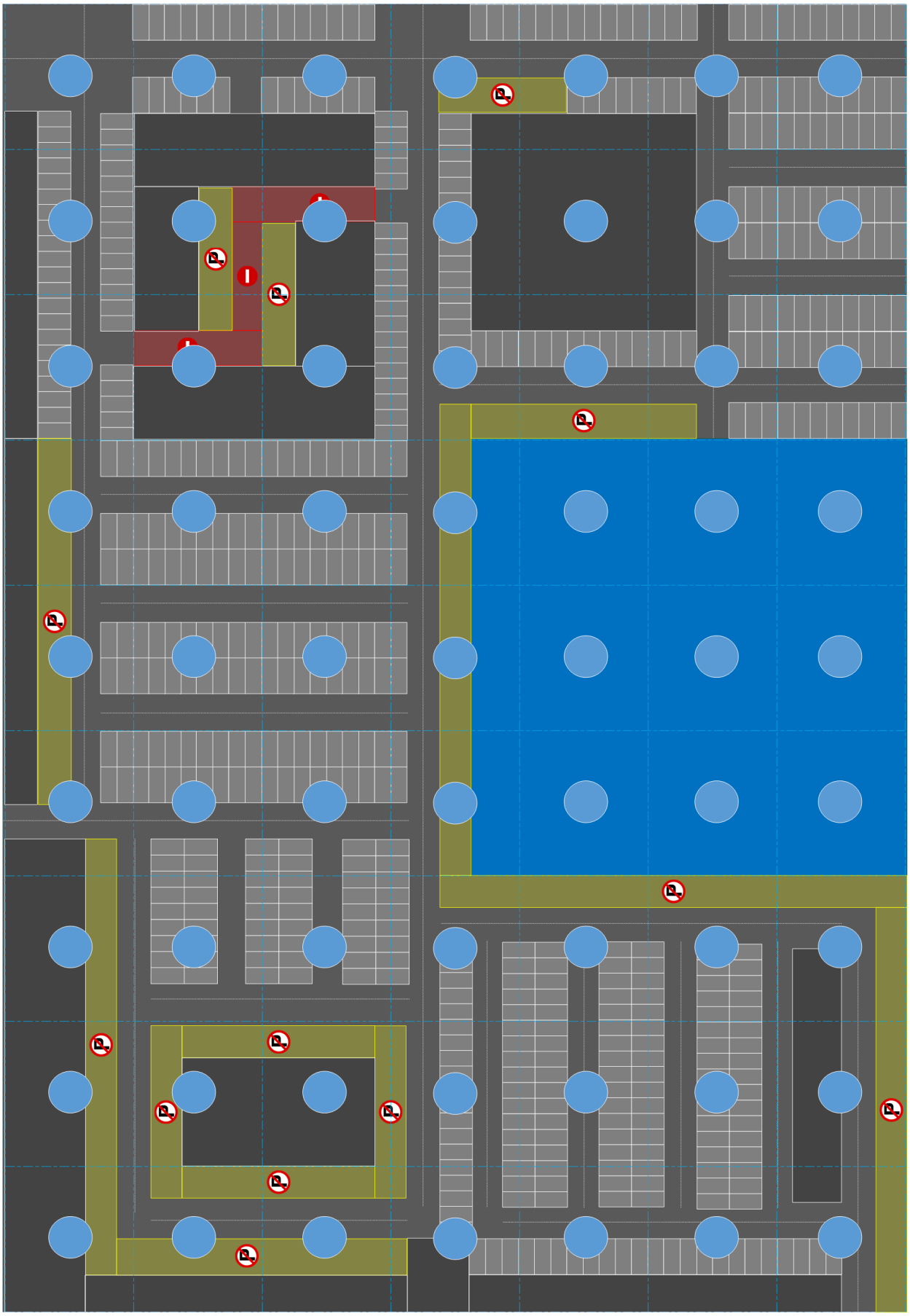

Figure 4.8: Camera distribution within the simulation environment

5

Experimental Design

The intended automatic suspicious behaviour detection system will be tested using a collection of experiments. Different scenarios will be acted out in the simulation environment and the response for the system is tested. In this chapter, we describe the experimental design of the intended automatic suspicious behaviour detection system. First the scope of the software will be defined by providing the functional design of the intended automatic suspicious behaviour detection system. Once defined, we continue with describing the experiment scenarios that will be used to test the system.

Figure 5.1: Wordcloud for experimental design

5.1. Functional design

The knowledge gathered in the previous sections form the basis of the design for the automated multi camera surveillance system. This section summarizes the basis of design by supplying a functional design. The use cases of the automatic behaviour detection system are shown in the use case diagram, given in figure 5.2.

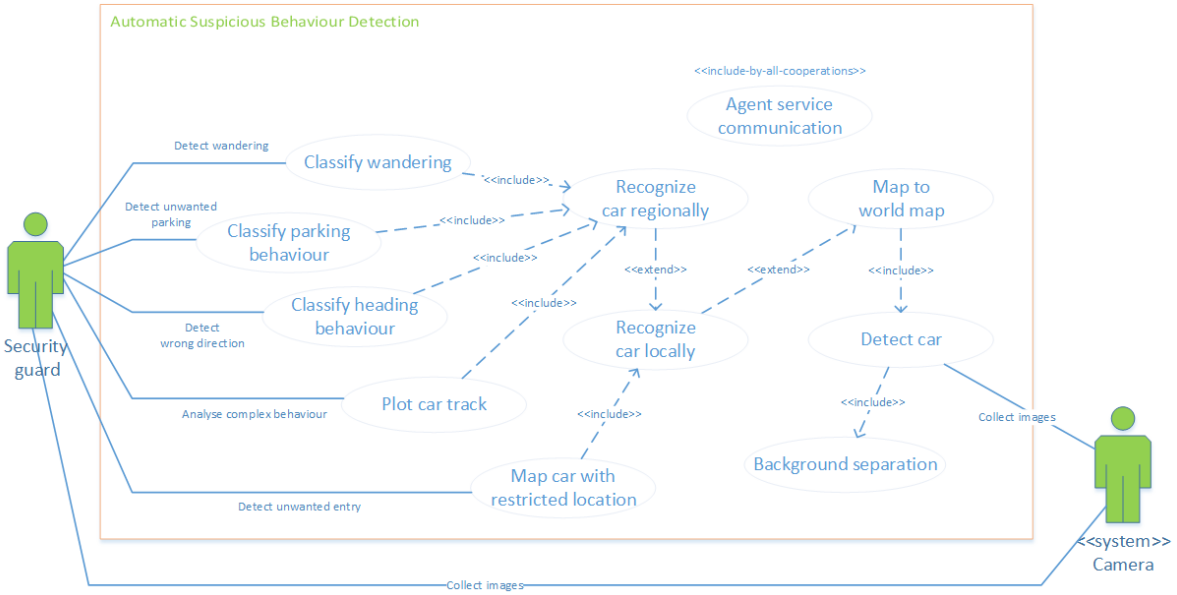

Figure 5.2: Use case diagram for the intended automatic suspicious behaviour detection system

5.1.1. Main use cases

The automated multi camera surveillance system has five main use cases, which will be implemented. These include typical classification cases in the driving behaviour in the simulation. The typical cases are based on the cases we also find in the security management of the driving behaviour on the officer training facility. The classification use cases include:

Classify entry behaviour

This use case focusses on the detection of a car within a region of interest. Smart cameras that detect cars in restricted regions must send an alarm to the security guard. Cameras must know where (within their view) it is not allowed to enter and match the detection.

Classify parking behaviour

Not all locations are assigned as parking lots and such locations are marked as illegal parking. The track must be analysed to detect parking actions and the behaviour must be validated for illegal parking. Parking on the edge of the camera view requires the information to be shared between cameras. Smart cameras that detect cars parking illegally must send an alarm to the security guard.

Classify heading behaviour

This use case is also known as the one-direction case, since it detects the illegal driving direction within a one way street. The track must be analysed to detect the driving direction and the behaviour must be validated for illegal headings. The behaviour can scope over multiple camera views and often requires the information to be shared between cameras. Smart cameras that detect illegal heading behaviour must send an alarm to the security guard.

Classify wandering behaviour

The environment expects visitors to know the environment and drive directly to their destination. Any diverging behaviour raises suspicion and creates a threat. Since no explicit illegal actions are detected, the system cannot raise an alarm. The system is expected to warn the security guard of the threat level of the behaviour. The wondering car must raise suspicion and the system must warn the security guard.

This behaviour typically happens distributed over many camera views. The system must have a method of communicating the information between the smart cameras.

Backtrack car track
Even when no suspicious behaviour is detected, the storage of the track already can be of great assistance to the security guard. Collecting the track information, requires a communication protocol to gather the normally distributed data. The system must provide the track information in full detail, which requires the communication protocol to gather it.

5.1.2. Supporting features

The main uses cases rely on supporting features to function properly. Because the main use cases are dependent of these features, all of these features must be fully functional in order for the classification processes to work. We define the following six features (in order of decreasing dependency):

Agent service communication
The cooperative intelligent cameras require a method of communication the information. The agent service communication defines the method of communication.

Background separation
The process requires the separation of the objects and the background in the images.

Detection
Detect the cars on the camera images. The process requires the separation of the objects and the background in the images. The goal of the detection process is to find the shapes of the cars in the images.

Map to world map
The detections only finds the location of the cars within the image. The location has to be mapped to the world coordinates in order to connect the detections on different cameras and apply location specific reasoning.

Local car recognition
Tracks are formed using a collection of detections for each car. Local car recognition ensures the detections that belong to a specific car are linked together to form one track.

Track information extraction
Additional features are extracted from the set of detections and stored as meta data for the tracks. The meta data is used to analyse the car behaviour.

Regional car recognition
Tracks involving multiple camera views require information on past detections from other cameras to recognize the detections as one track. Through the agent service communication, the information is distributed and the car should be recognized (again) on the follow-up intelligent camera. Additional features are extracted from the set of detections used to analyse the behaviour.

Extracting the features, classifying the behaviour and communicating the information is the heart of this system and thesis.

5.2. Experiments

We will perform tests to validate the performance of the automatic suspicious behaviour detection system. The tests are designed to test individual features and overall performance. The feature tests will test the main features described in the functional design. The reasoning tests validates the performance on the detection of suspicious behaviour.

5.2.1. Feature tests

The feature tests are designed to check individual features for the agents. The features are the building blocks for the overall system to operate. All of the tests have to pass in order for the system to be fully functional. We will describe what is tested and how the test validates the functionality.

Service communication (TEST-01)

Services are used to communicate between the software agents or cameras. It uses asynchronous procedure calls to provide services from one agent to the other. The service system is responsible for the management of the sessions. The feature tests for the service communication will show the system's capacity to do so.
All tests involve a service providing agent and service calling client agent. These agents will be created for this purposes. The service communication is tested with the following tests:

A - Call mock function without return value
The server agent will provide a mock service and the client will call the mock function. The service system will provide the communication and calling of the procedure. The test succeeds when the mock function is called.

B - Call mock function with return value
The server agent will provide a mock service and the client will call the mock function. The response will be predetermined and send back by the mock service. The service system will provide the communication and calling of the procedure. The test succeeds when the expected response is received by the client agent.

C - Call mock function with failing response
The server agent will provide a mock service and the client will call the mock function. The mock service will be set to fail, leaving the client waiting for a never to receive response. The service system will provide a fail safe mechanism to notify the client of the error. The test succeeds when the expected error is received by the client agent.

D - Initiate reasoning parameters
A simplified version of the smart camera process will be created to test the initiation process. The client process will start its behaviour by requesting initial parameters from the service providing process. Once the parameters are received, the smart camera will start a dummy process with the use of the initial parameters. The dummy process will show the system is able to initiate (and start) a smart camera process.

E - Communicate perceived or extracted information between agents
This tests is designed to test the smart cameras ability to set up a peer to peer connection between to smart cameras, based on the received initial parameters. The test succeeds when the second camera received an example measurement from the configured camera.

Car detection (TEST-02)

Car detection requires all smart cameras to preprocess the images and detect the shapes that represent the cars. This feature runs one each camera and does not require sharing information between the cameras. The problem is the same for each smart camera.
The car detection process require a delegation process and a smart camera. In addition, this is the first test that requires the simulation environment to stream the camera images to the automatic suspicious behaviour detection system. The system communicate using the service communication feature. The following tests will be applied to the car detection feature:

A - Detect car at random location completely covered in one camera view
Cars that are completely visible should be detected without any problems. The smart camera must be able to separate the car from other (background) objects, pinpoint the location and map it to the relative location. For the test, the simulation will park a car on a location within the camera image. The test succeeds when the car is detected on the right location. The cars will be simulated at random locations within one camera.

B - Detect car at random location completely covered in one of multiple camera views
This test is similar to the previous test. The locations of the car can now be picked within any camera view. They will still be completely visible within the image, but it is unknown on which camera the car will present itself. The test shows that all the cameras are now functional.

C - Detect car at random location partially covered in one camera view
Cars that are partially visible are harder to detect and cannot be detected with the corresponding orientation. The smart camera must be able to separate the car from other (background) objects. A margin of error is to be expected in the (relative) location. For the test, the simulation will present a car on a random location on the border of the the camera image. It is known beforehand, which camera will have this image. The test succeeds when the car is detected within acceptable margin of the actual location.

D - Detect car at random location partially covered in one of multiple camera image
Again, we perform the same test on the detections on the edge, but this time the simulation shows the car at any random location on the edge. Here we test if all the cameras are able to perform the car detection with cars on the edge of the camera view.

World mapping (TEST-03)

World mapping is applied the map the detection from the relative location within the image to the world coordinates. The feature requires the car detection to function properly. The world mapping problem is the same for each smart camera and only differs in initial parameters. Distributing the initial parameters is part of the automated task delegation process. The car detection process requires a delegation process and a smart camera. In addition, this is the first test that requires the simulation environment to stream the camera images to the automatic suspicious behaviour detection system. The system communicate using the service communication feature. The following tests will be applied to the world mapping feature:

A - Map coordinates of car detection of one camera
The smart camera must be able to map the coordinates of the detection within the image to the coordinates in the world map. It uses the relative location of the camera to perform this mapping. The test succeeds when the car is mapped to the world coordinates. The simulation will show the cars on random location, within the view of one specific camera.

B - Map coordinates of car detection of all cameras
Now we pick a random location from anywhere on the map and present it to the MCS. The system must be able to map the detection to the world coordination system, using the received parameters. The difference with the previous test is again the number of cameras involved in the test.

Car recognition (TEST-04)

Detecting a car is immediately followed by matching the detection to specific cars. In the car recognition, the gathered knowledge about the known cars is used to match the detection to the cars. It is the first feature that requires the cooperation between smart cameras (in specific situations). For the car recognition tests, we use two smart cameras and a delegation process. The feature is tested using the following tests:

A - Recognize car within one camera view
The core of recognition feature is tested by simulating a car driving through one camera view. The tests succeeds when the car is created on the first detection and matched to the following detections.

B - Recognize car within one of multiple camera view
The core of recognition feature is tested by simulating a car driving through one camera view. The tests succeeds when the car is created on the first detection and matched to the following detections.

Extracting tracking information (TEST-05)

The extracting of tracking information is heavy dependent on the recognition feature. But once a series of detections are connected as a track, the series can be used to calculate additional features that represent the car behaviour. The features discussed in paragraph 3.6.3 sum up the features used in the automatic suspicious behaviour detection system. The tests for this feature are similar to the tests for car recognition, but focus succeed when the parameters are calculated.

A - Recognize car within one camera view
The system must extract the following features from the track: *speed*, *acceleration*, *heading*, *rotation* & *duration*. The duration is calculated in both number of detections and time span. For this test, a car will be simulated to drive through the view of one camera. The test succeeds when all the extracted features are correct.

B - Recognize car within one of multiple camera view
The system will also be tested for all cameras, in staid of just one. The tracks will now be generated in different camera views and the system MCS must be able to extract the information from all of them.

Recursive car recognition (TEST-06)
After the car leaves one camera, it will move over to the next camera. The cameras must share that information and recognize the car, even though the latter never saw the car before. The regional car recognition test validates the MCS's ability to re-recognize the car, when it is driving across the view of multiple cameras. The feature is tested using the following tests:

A - Recognize car within neighbouring camera views
The recursive car recognition feature is tested by simulating a car driving through two neighbouring camera view. The tests succeeds when the car is created on the first detection and matched to the following detections, including the ones in the latter camera view.

B - Recognize car over distant camera views
Cars can eventually navigated through the complete set of camera views. To test this feature, a car will drive from one corner to the other corner with all the cameras initiated. The complete set of detections should show up and collected as detections from one car.

Recursive extraction of tracking information (TEST-07)
Not only the recognition must be supported over multiple camera views, the extraction of the tracking information must remain precise as well. The simulation will be similar to the recusive car recognition test, but this time, the extracted track information is validated. The test case include the following tests:

A - Extract information from cars driving within neighbouring camera views
The system must extract the following features from the track: *speed*, *acceleration*, *heading*, *rotation* & *duration*. The duration is calculated in both number of detections and time span.

B - Extract information from cars driving over distant camera views
The world mapping allows the system to track the car over multiple camera views. Because these coordinates are all relative to the world coordinate system, the features can be calculated similar to features calculated from the detections within one camera view. For this test, a car will be simulated to drive through the view of two neighbouring cameras. The test is successful when the features are calculated from a track covering multiple camera views.

5.2.2. Reasoning testing
The reasoning is the last feature in the pipeline and covers the main use cases for this thesis. All the use cases are tested individually for their performance. In contrast with the feature tests, the reasoning tests evaluate the performance and do not require to pass completely. We aim for a high detection rate for the unwanted behaviour.

Illegal entry (TEST-08)
Illegal entry is the detection of cars within an assigned region of interest. For the test, we will select 5 regions in which the car is not allowed to enter. 10 scenarios will be creates and for 5 of them the car performs the illegal action. The performance is indicated by percentage of correct classification.

Illegal parking (TEST-09)
Illegal parking is the detection of cars parking within an assigned region of interest. For the test, we will select 5 regions in which the car is not allowed to park. 10 scenarios will be creates and for 5 of them the car performs the illegal action. The performance is indicated by percentage of correct classification.

Wrong direction (TEST-10)

Only in one region in the Den Helder case has an one direction street. For this use case, we will compile two scenarios in which the car is driving the wrong direction. Eight scenarios are applied in which the car uses the same road, but in the correct way. The performance is indicated by the percentage of correct classification.

Complex behaviour (TEST-11)

The previous tests validated the ability to detect unwanted behaviour. But when behaviour becomes more complex, it looses the option of absolute detectability. The behaviour no longer is unwanted, but suspicious and has a threat level. Suspicious behaviour is the set of events that combined arise suspicion. The performance of the system cannot be evaluated using qualitative analysis. The following scenarios will be tested and analysed.

A - Detect aimless driving

The test include 4 scenarios in which the car will be driving aimlessly through the environment. Scenario 1: driving local circles. Scenario 2: driving alone the edges of the environment. Scenario 3: looking for parking space. Scenario 4: Taxi dropping off student.

B - Driving near high risk structures

The KIM and armoury are considered high risk structure. Both will be tested with a car driving past it multiple times and car parks near it.

II

Building the system

6

System & software design

This chapter focusses on the system design choices. It describes how the system components will work together in order to automatically detect and show the unwanted and suspicious car behaviour. We will first review the system components, which the design is based on. Then we will look at the internal structure of the agents and the processing pipeline. From there, we will continue by looking at the roles and services of the agents. Finally we discuss the reasoning applied by the agents.

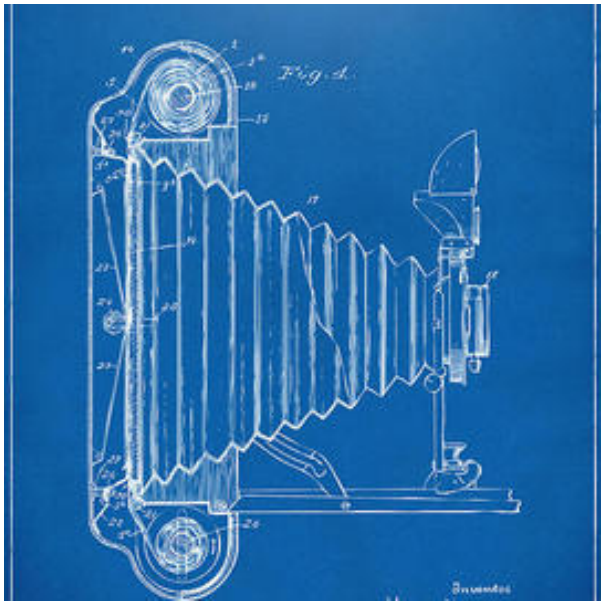

Figure 6.1: Blueprint of early photo camera

6.1. System components

The first step in the system and software design is choosing the hardware components for the intended automatic suspicious behaviour detection system. The last paragraph of chapter 3 already hinted towards the components found in the system. Figure 3.4 shows three components groups: *cameras*, *reasoning system* and the *terminal*. The reasoning system will be the main topic for this thesis. It consists the collection of intelligent cameras that together form the automatic detection system. The reasoning system is intended to be an extention of the multi camera surveillance system and within the setup, the original and added functionalities of the VSS are separated. In the following paragraphs, we will describe the system components and the interaction between them.

6.1.1. Intelligent cameras

Multi camera surveillance systems have many cameras streaming images to the control room. To create a system in which intelligent cameras cooperate to detect the suspicious behaviour, we need cameras with reasoning and communication capabilities. When we speak about intelligent cameras, we refer to cameras that are able to perform both tasks of capturing and reasoning. Intelligent cameras will have all required components in one machine, but for the design it is easier to separate the two functions.

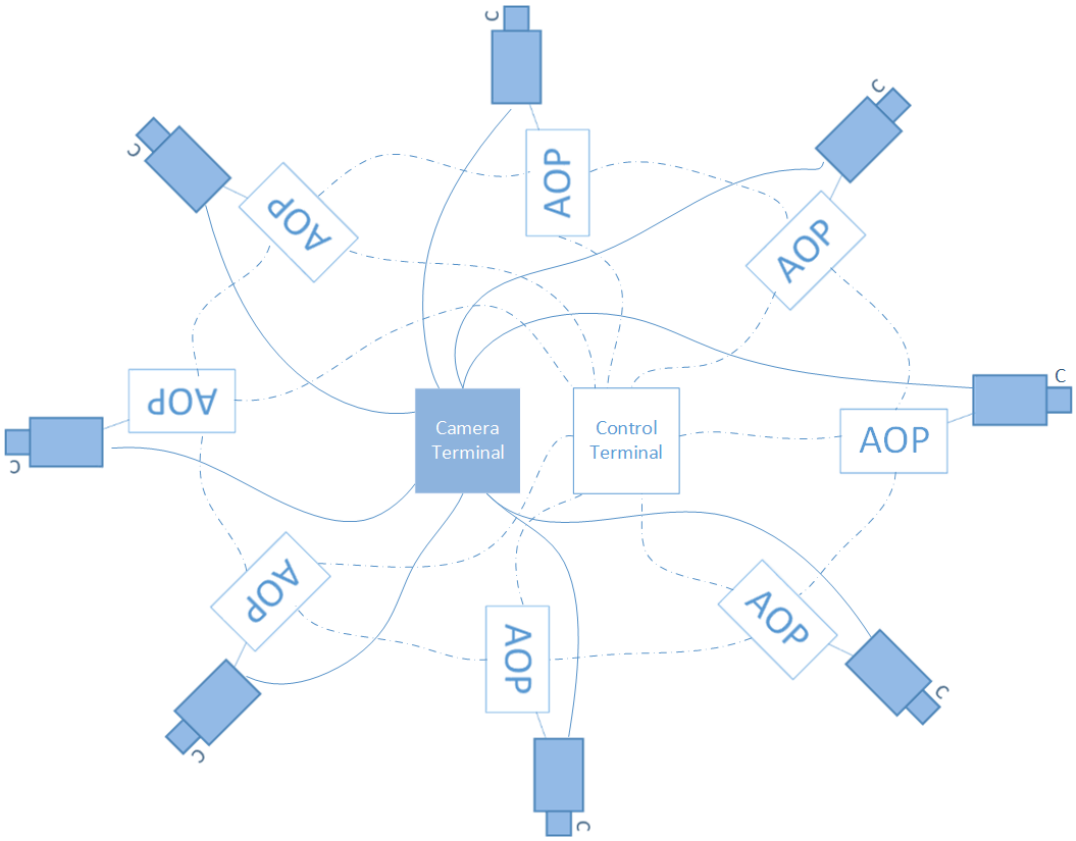

Figure 6.2: System components

The image capturing components will be referred to as the camera. The cameras are extended with reasoning and communication capabilities by assigning software agents to each camera. The software agents requires a platform to operate on and thus intelligent cameras are equipped with additional hardware to perform the calculations. The camera stream images to the agent, which in turn start processing the images. Future cameras may be equipped with dedicated hardware for these purposes, but for the course of this research, we will assume the platform is any programmable platform, which supports a Java Virtual Machine (JVM). The JVM will be the platform for the JADE container, discribed in chapter 4. A raspberry pi is a good example of such a platform [26]. The system components are shown in figure 6.2. We will assume the cam-

eras are connected in the same local area network (LAN), but the network could also be implemented over the internet using Virtual Private Network (VPN). All the cameras are able to communicate directly to each other. Many of the connections between the agent operating platform (AOP) are left out of the diagram to avoid unclear overview.
The application will be build by using simulated cameras, meaning the image capture device will be virtual. For the simulation it is important that the simulated input represent the real life output as much as possible. We assumed a basic camera only produces images with a fixed frequency and no further meta data about the image is available. The interface between the camera and the agent's operating platform can differ from device to device. We decided to use an UDP socket connection, because it can send the images to both an existing terminal and the agent operating platform.

6.1.2. Terminals

The terminals are the gateway for the security guard to monitor the events in the environment. The camera terminal is the existing monitor of the environment, often a series of monitors showing the camera streams. The control terminal for the automated surveillance system contains both the visualisation of the detected events and the delegation service. The combination of the terminals form the user interface as proposed in figure 3.5.

6.2. Guardian agent framework

The developed software agents have an unified method of Digital Signal Processing (DSP), agent-to-agent communication and internal data and event logging. For the purpose of this application, a framework was developed to supply each of the agents with these shared features. In this section, the design of the guardian agent framework is discussed. The next section will provide the design of the individual agents, using this framework.

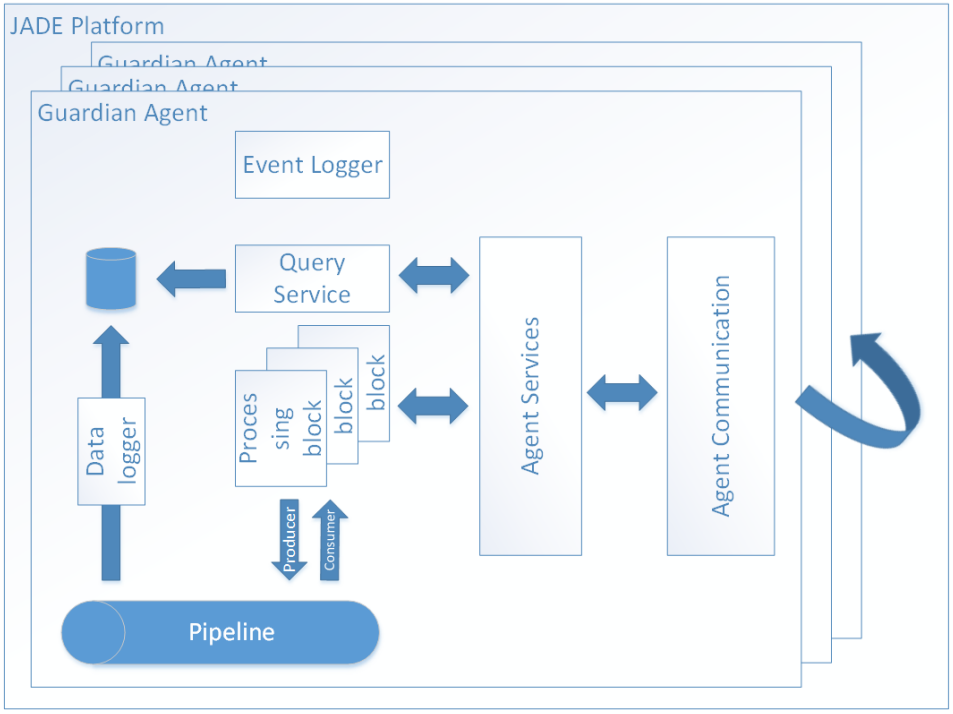

Figure 6.3: Agent components

6.2.1. Guardian agent design

JADE agents are always created from the same base design. This design has been discussed in chapter 4 and can be summarized as extensions of the agent class, which have programmed behaviour. The intended automatic suspicious behaviour detection system will contain different agents. Since all the agents process information and communicate in a similar fashion, the generic guardian agent base class was developed to supply each of them with the framework to perform these actions. The added features include the pipeline structure, data and event logging and the services support system. An overview of the software components in the base class are show in figure 6.3.

As previously discussed, the guardian agent is an extension of the JADE agent. The relations between the guardian agents and JADE agents are represented in the UML class diagram given in figure 6.4. Using this structure, any guardian agent can be executed as a JADE agent in the JADE platform. The initiation and behaviour model of the guardian agent is also similar to regular JADE agents. The guardian agent framework uses JADE behaviours internally to execute the events resulting from the data transactions and services requests. The framework provides DSP modules and services support to define the behaviour of the agent. Since the framework is based on the JADE agent, it shares the same agent life-cycle.

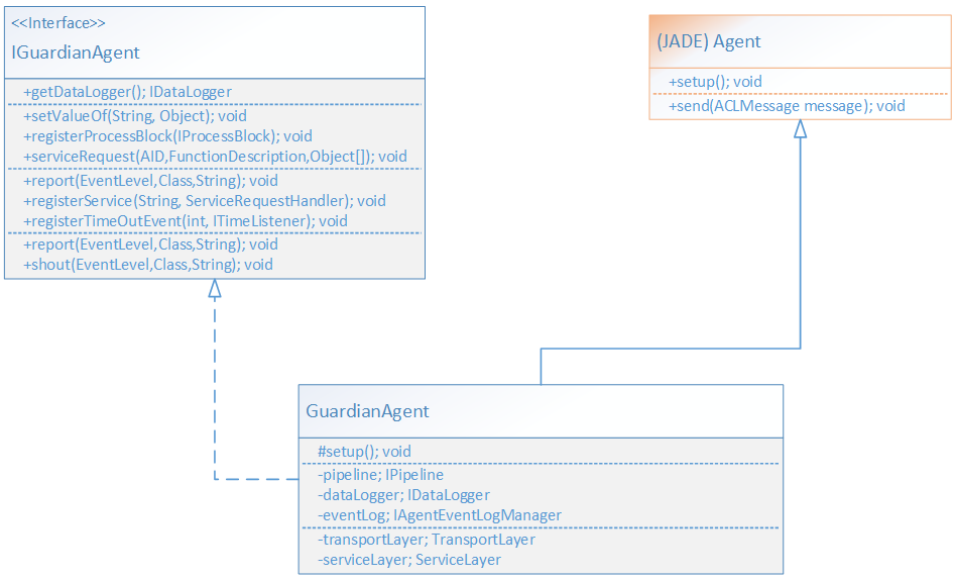

Figure 6.4: Guardian agent class diagram

6.2.2. Digital Signal Processing Modules

The DSP pipeline is the heart of the design and passes through the agents. Each agent contains a pipeline, in which data can be stored. The pipeline is setup to support a producer/consumer method for processing blocks. Every processing block can subscribe to updates of variables and/or supply new values of variables. The data distribution method is inspired by OLE for Process Control (OPC), where OLE stand for Object Linking Embedding [24]. OPC is a technique to exchange data between systems. Every datasource is packages in a subscription package and clients can subscribe to be notified of any changes in the datasource. OPC Data Access (DA) clients will receive the latest value from the OPC DA server, while they are subscribed. Historical data is considered an additional service and can be implemented in many different ways. We designed the system such that every agent contains a pipeline, which processing blocks can subscribe on [30]. It is even possible for processing blocks to be executed within different agents. Data update events that are send to other agents, are communicated through services calls, which will discuss in the next paragraph. Historical data is stored locally and together with the other agents form a distributed database. The data is accessible through the query service.

The components used in the DSP modules are shown in the diagram in figure 6.5. The modules can produce data, consume data or perform both. The production of data is symbolized by the 'setValue' operation available through the guardian agent. The consumption of the data is performed on any set event of the subscribed data. The operation follow a subscription design for event handlers. Each DSP module also needs a

trigger policy. The system supports three types of triggers: *periodic*, *time-out event* and *set event*. The periodic trigger is an event generated with fixed time-outs between the events, a time-out event is one-shot event generated after a given number of milliseconds and the set event is a data-driven event. It is left up to the implementation to choose the policy that fits best with the specific DSP module. Any DSP module that is not triggered by an data set event, can use JADE Behaviours to trigger the processing.

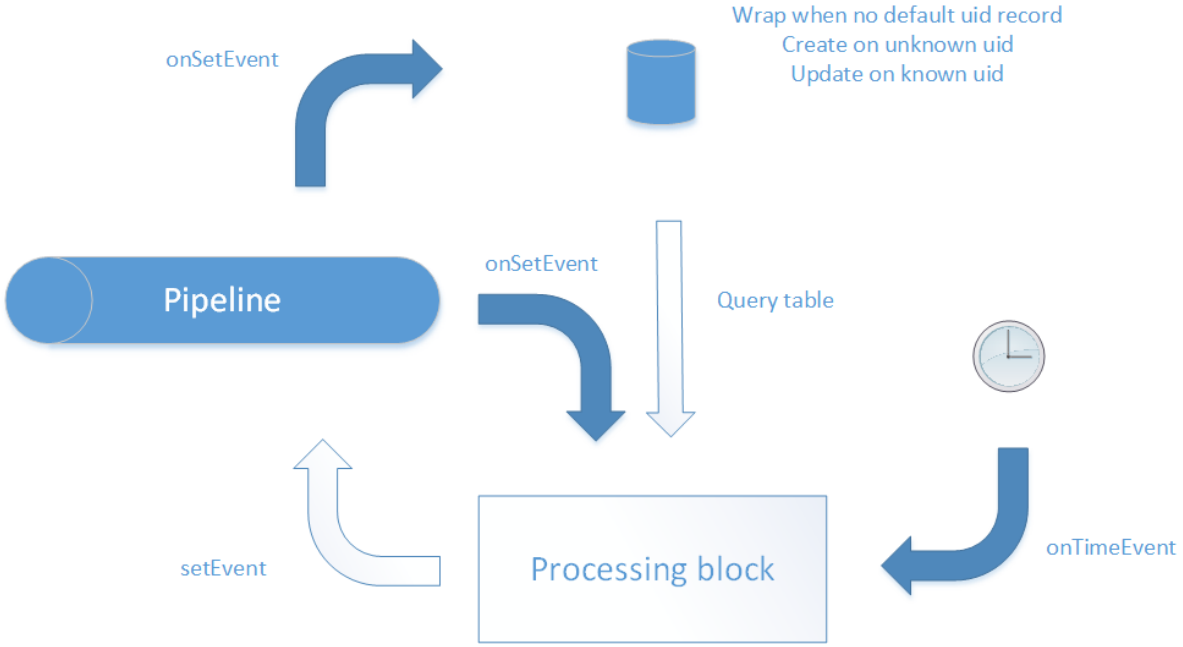

Figure 6.5: The pipeline components provides both memory management and data-driven events. Every set value event is stored in the agents memory and triggers the onSetEvent on the processing blocks. If needed the historical data can be queried from the agents memory. In addition, the agents provides time events in the form of both periodic and one-shot events.

6.2.3. Agent-to-Agent communication

We decided to use services to communicate between agents. All guardian agents are expected to use this method instead of the JADE communication protocol. Guardian agent services are asynchronous procedure calls that are made available to other agents. A layered encoding system build on top of the existing communication method provided by JADE allows procedures to be called remotely. Each agent can support different services. The services are registered with the agent's service system, which will manage the triggering of call events and sessions during runtime. Due to this extension, we now have a system on which every platform can run any agent and any agent can support its own set of services.

The service management is responsible for managing the logistics involved in the service calls. It follows a layered design pattern, which is often applied in communication protocols. The three layers involved in the design, inspired by the OSI-model, are shown in figure 6.6. Building upwards from the bottom, the transport layer is responsible for the distribution and receiving of the messages. It uses the communication channels available within JADE to send the messages back and forth to the other agents. Among others, the JADE communication channels allows Serializable Java classes to be send between agents [21][73]. This property is used to send transport layer packages to the other guardian agents. The session layer manages the services of the guardian agent. At the initiation phase of the agent, the supported services will be registered in the session layer and incoming services calls will be redirected to these registered services. In case the service generate a response, the session layer ensures the response is send back to the requestor. On the requestor side, the session layer makes sure an event is triggered when the response is received. The session layer only handles service requests and response. The translation from procedure call to service request has to be done by the service itself. There is however a predefined process, which will be discussed in the implementation chapter. By using the three layered design of communication, we are able to set-up processing streams and distribute detected events. JADE provides many methods for communication, some very closely related to our implementation. However, we felt these techniques were less suited for a DSP stream environment used in this design.

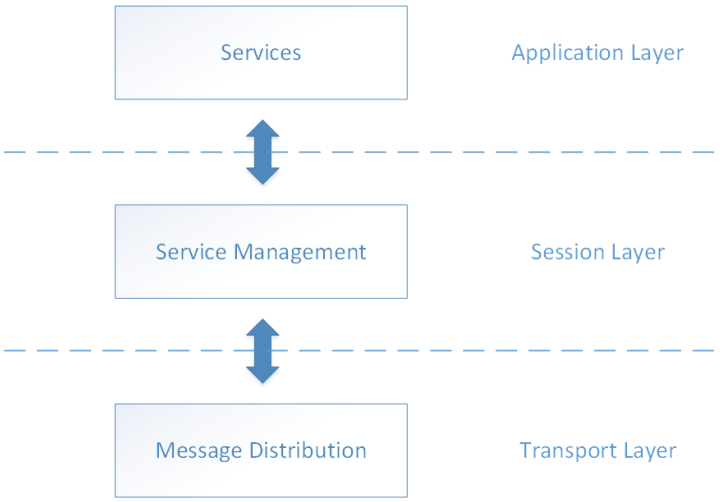

Figure 6.6: Service management layered design

6.3. Agent roles

We discussed the system components, the agent's method of information processing and communication. The next step is the design of the agents themselves. In this section, we will look at the roles of the guardian agents. We describe how the tasks are delegated and list the expected DSP modules for each agent. The next section will describe what processing is applied within these DSP modules.

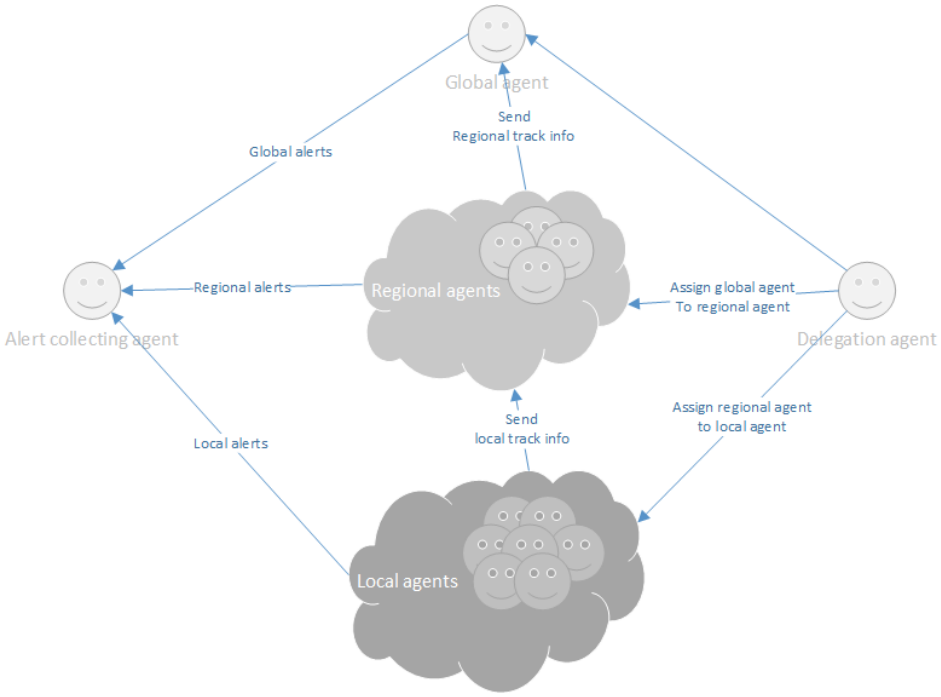

Figure 6.7: Agent rules and processing flow

6.3.1. Hierarchical reasoning

The processing pipeline of the automatic suspicious behaviour detection system follows a hierarchical design. Many agents perform local processing steps, while every layer in the hierarchical design has less members. Automatically, the design is a mixture of homogeneous and heterogeneous agents. There are five different types of agents. In alphabetical order these are: *Alerting Agent*, *Delegation Agent*, *Global Reasoning Agent*, *Local Reasoning Agent* and *Regional Reasoning Agent*. Within the agent type, the agents has a homogeneous design and the initial parameters only influence the conclusions, not the reasoning methods. These initial parameters are assigned by the delegation agents, named as such because in doing so, it ultimately delegates the tasks to the agents. Agents of the same type contain homogeneous behaviour and only differ in there processing, based on the area they are in. An overview of the types of agents and relationship among them is shown in figure 6.8.

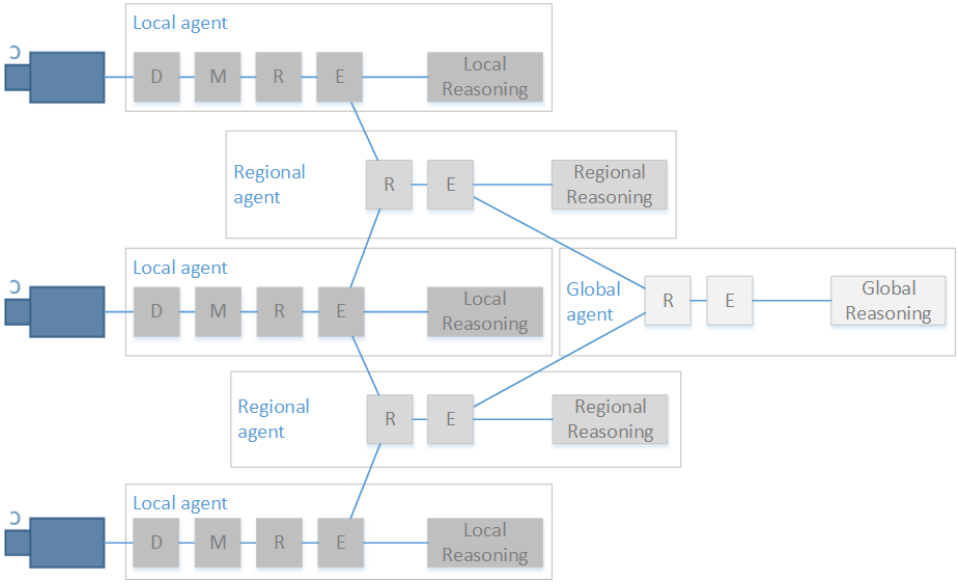

Figure 6.8: Agent roles and communication. Each pipeline has at least 5 components: detect, map, recognize, extract features and reasoning. Step 3 & 4 can be repreated to merge over tracks larger regions.

The processing pipeline for the automatic behaviour detection system runs through the local, regional and global agents. Starting with the agents connected to the cameras, each agent processes the available information and passes it on to the next process, creating an information flow. The local agents provide the local track data, which the regional agent subscribes to using the tunnelling mechanism. The updates received by the regional agents are stored in the agents memory and historical values are managed within the agent, based on the functionalities of the agent. New features extracted by the regional agents in turn are provided to the next layer of agents and finally end up in the global agent.
In our suspicious behaviour classification processing pipeline, agents perform five tasks: *detect*, *map*, *recognize*, *extract features* and *reason*. The relation between these agents are design with an hierarchical model, where each higher level gains a more global image of the car's behaviour, as shown in figure 6.8.
Reasoning is performed at multiple levels and distributed over different agents. Conceptually each agent has an area which it will have to guard. The larger the area, the more complex the behaviour of the cars will be. Local agents look at high resolution images with an high frequency, while global agents look at almost binary detections with low frequency over the subregions.
The hierarchical system uses one-way communication to form the distributed pipeline. It creates a distributed variation of the divide-and-conquer approach to classification [37]. The local agents are independent of each other and can run in parallel. The following layer has to wait for the input from the local agents, but once received, all the agents can execute the reasoning in parallel. The simultaneous executing of different programs on a data stream is called stream parallelism [30]. The hierarchical mapping is a powerful tool in parallel computing, but only applicable in specifically structured problems [30]. We designed the reasoning pipeline to ensure the hierarchical stream parallelism was possible, allowing the information processing to run in parallel.

Like every classification process, each pipeline contains a preprocessing, feature extraction and classification step. The distributed design splits up this process and combines the processes within different agents. Every type of agent has its own unique processing pipeline. In the following paragraphs, we will describe how these classification pipeline utilizes for the local, regional and global reasoning agents. Once this picture is clear, we will go into the technical details of how the features are extracted.

6.3.2. Local information processing

All the camera's are equipped with a local reasoning agent. The processing steps are shown in figure 6.9. The details of each DSP module will be discussed in the next section. For now, we only describe what each step does. It is interesting to note that all the DSP modules run in parallel. The event mechanism creates a sequential processing pipeline, but all the modules run on separate JADE behaviours. Each agent receives a series of images over time. The pipeline starts with detecting the cars in the image. Once the cars detections are known, the image coordinates have to be mapped to the environments coordinates. In real world environments, the mapping step requires the mapping of both the two dimensional image to three dimensional world and the relative location of the camera to the other cameras.

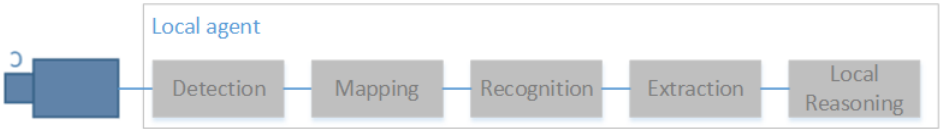

Figure 6.9: Local information processing

Once multiple detections are made, the detections must be combined to form a track. The recognition algorithm is applied for every new detection in the camera image. The recognition fuses the individual detections to a track. It uses the behaviour of each track to select the most likely track for new detections, including the option of creating a new track. This makes the tracking algorithm a classification process.
The final processing step extracts additional information about the track. The parameters calculated based on the track are used to classify the behaviour of the car. The local feature extraction calculates the following features from the track:

- Location x,y
- Heading
- Speed x,y
- Speed turn
- Acceleration x,y
- Duration

Almost all of these parameters can be extracted from the last three detections. The only exception here would be the duration. The duration is an iterative property and is incremented with each new detection.
The local information stream continues to the local reasoning process. The local reasoning process checks for locally detectable suspicious behaviour. Local reasoning components by design do not reason about implicit suspicious behaviour outside the designated region & only reason about behaviour detectable within the camera view. The local track and camera images hold only a small amount of information and the detectable behaviour will not be of high complexity.

6.3.3. Regional information processing

The regional agents analysis behaviour over a small region with multiple cameras. It combines local tracks, forms regional tracks and analysis the behaviour detectable within the scope of the cameras. Thanks to the cooperation between the guardian agents, the regional agent does not need to perform the same processing steps as the local agent. Several local agents go through the detection and tracking process described in the

previous paragraph. The same local track information that is used for local reasoning is also send to the regional guardian agent. The communication between the agents is done using a combination of a pipeline and subscription technique. The process is shown in figure 6.10.

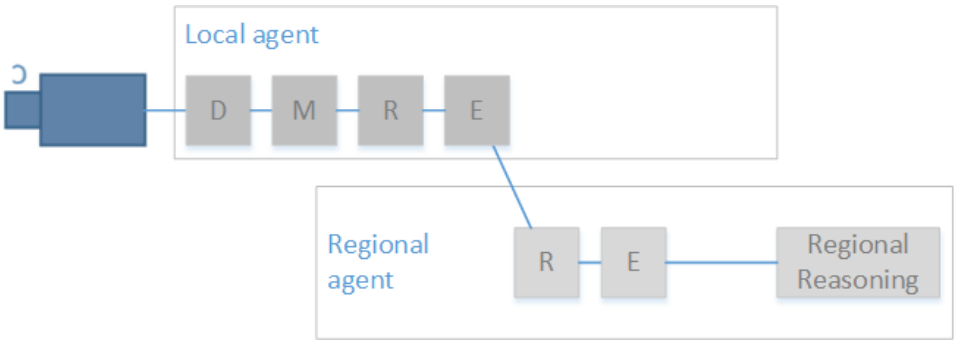

Figure 6.10: Regional information processing

The regional agent receives the local tracks from multiple local agents. As the car navigates through the environment it passes from one camera view to the neighbouring camera views. Regional agents receive the local tracks and merge them to form one long track. Combining the local tracks is a process similar to the recognition of a car from individual detections. In fact, linking the tracks together is based on the same principle that the car will continue on the same heading. However, we do not need the same information as the local tracking used; we only need the information around the edges. Everything in the middle of the camera is less informative and can be summarized in the features of the track. The extract feature process for regional agents perform exactly the same process as the local agent version and extracts the features of the regional track. This effectively reduces the data packages size with every hierarchical step. The packages reduction is shown in figure 6.11.

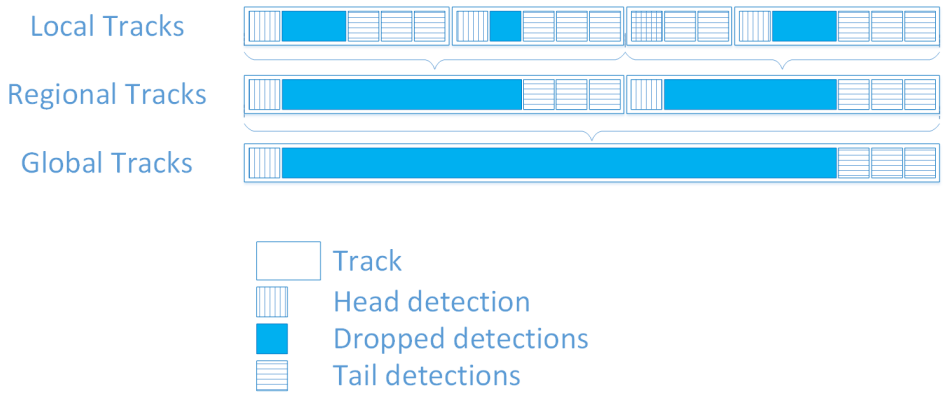

Figure 6.11: Track data visualization. The dropped detections are summarized with the features extracted from the track.

The extraction process for the local reasoning component and the communication are slightly different. The cooperation is optimised to maximize the available information within the system and minimize the data communicated between the agents. The track within the local agent contains all the detections and has the full resolution. But before the track information is communicated between the agents, the information is stripped from its irrelevant detections. Only the edge detections are kept and the detections in the centre of the track are dropped. This process is performed by an edge track module, which only sends the track information when the last detection is on the camera's edge. This optimizes the communication by both reducing the message content and the frequency.

6.3.4. Global information process

The global information process highly resembles the regional information process. It combines the track from the different regions to one track through the complete region. Every new step in this track combination process looses more detail and tracks are considered with a lower resolution. The decrease in resolution, also decreases the bandwidth of the data streams and avoids the bottleneck on the global agent's datastream. For this reason, the global information process connects regional tracks. The tracks are combined using a chance model, including knowledge of the environment. The process is shown in figure 6.12.

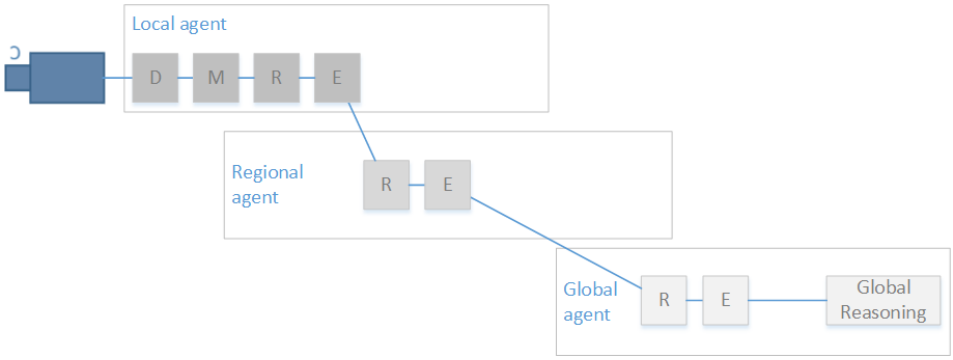

Figure 6.12: Global information processing

Global reasoning is only applied when the track covers more than one segment of the map. As a result, the global reasoner mainly focusses on the most complex behaviour, such as wandering cars. However, particularly the edges of the regions form a problem area for the local and regional reasoners, since the behaviour is only observed partially. For the model to optimally distribute the task load, the cameras and regions should be installed in such a way that the main activities occur away from the edges.
The system described until now forms the basis of the hierarchical reasoning model. This process of combining track information can be repeated both temporal and spatial. Over time, the tracks information is reduced by the extract feature step and the track will have a limited data size. Spatially, the track segments are fused to form larger tracks, but the data size does not increase. The combination of track segments can be performed over and over again for every intermediate step between the local agent and the global agent. With every step the agent is responsible for a larger region, has to detect more complex behaviour and will have to base its detection on lower detailed information. Every behaviour classification problem will have to be evaluated to determine at what level this detection has to be performed. The final choice depends heavily on the application.

6.3.5. Track reconstruction

Throughout the layers, the tracks lose their precision to avoid the bottleneck in the higher levels of reasoning agents. For the feedback to the guard, it is important that the track can be reconstructed on demand. The data structure is adjusted to support this reconstruction, without storing the track in full precision on any single node.
When combining the track segments, the precision of the original track segment is lost. The car gets a new track (segment) for each layer with its own unique identifier. For the higher levels of agents, the track data structure is expended with a list of children track segments. When the track segments are combined to form the higher level track segment, the most of the content of the track is lost, but the reference to the original track segment is stored. This process is shown in figure 6.13.
The track can be reconstructed using these children references. We created one track that will store all the detections found in them. Using a divide-and-conquer approach, we can collect the details of the track, starting with the global agent and recursively work our way to the local agents. Each layer queries the referred agent for the child track segment. When the track segment is received, the track is combined and the track is reconstructed in full detail.
Interestingly enough, the detections on the edges are less precise, since the car was partly outside the image. But since the cameras have a neighbouring non-overlapping configuration, when one camera detects a part

of a car, another camera is expected to detect the other part of the car (with the exception of the global edges). Using these two detections, the regional agents are able to reconstruct the actual location of the car. To combine the detections and link the track segments, the concept of stitches is introduced. Stitches are the parts of the track that contain the overlapping detections. They contain new detections that will replace the less precise detections from the camera and store the sink and source track by reference. These reference can be used to reconstruct the track.

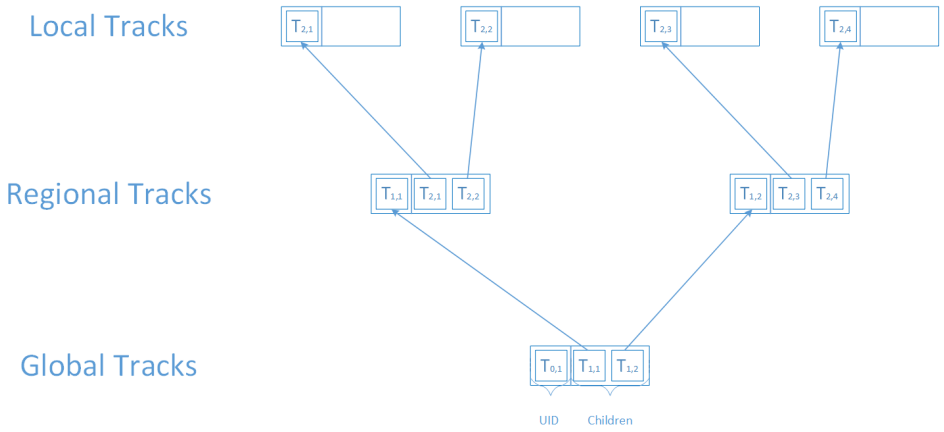

Figure 6.13: Track reference structure visualization

6.4. Digital signal processing modules

The images from the cameras tell the story of the events in the environment. But before the system can reason and classify the behaviour, information has to be extracted. The system knows at least four processing steps before the reasoning is applied: detection, mapping, recognition and feature extraction. This section describes the techniques applied for these calculations.

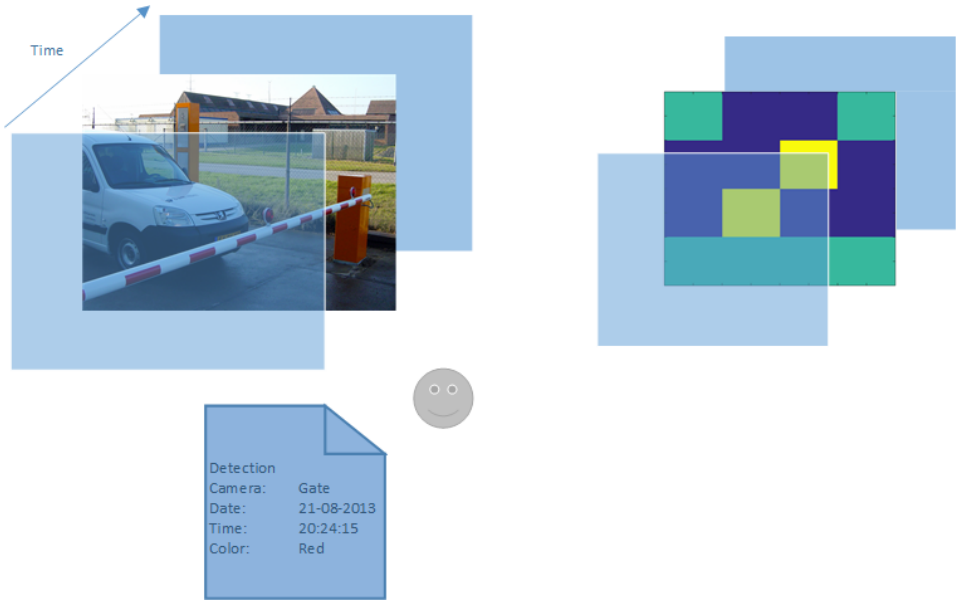

Figure 6.14: Local information processing

6.4.1. Detection

The detection step detects the regions in the image which contains the cars. For real cameras, this would be an object detection process, including noise filtering and object classification. These steps all have a chance of misclassification. For the current research, this process is out of the scope and we want to limit the misclassification as much as possible. The low error rate is ensured by simplifying the process of detecting with the setup of the cameras, the resolution of the cameras and the simplicity of the classification process. As we indicated earlier, the simulation environment allows us to configure the cameras such that they minimize the error introduced by the detection process. Figure 6.9 shows how the real world detection problem is translated in the simulation environment. The virtual cameras will not encounter any noise and will be placed straight above the environment.
The object classification will include two simple steps. A preprocessing step applies a well known technique of background subtraction to find the dynamic components of the image. The background is sampled over time and the average over a large set of samples is stored. The background than subtracted from the new images and changes in color become visible. The remainder is any dynamic object in the image.
Then the cars are detected through template matching. A template car is generated and used as a two dimensional filter. This cross correlation process will indicate where in the image a dynamic object most represents a car. The process is repeated for multiple angles, using the templates shown in figure 6.15. The location with the highest cross correlation is classified as a car, assuming the correlation is higher than the threshold value. Even though the technique will detect the car in the simplified image, there still is a 180° symmetry in the image, since it is unknown in what direction the car is driving. This information will have to be extracted from the track. Therefore, every detection has both a low-angle and a high-angle, respectively representing the angle less than 180° and more than 180°.

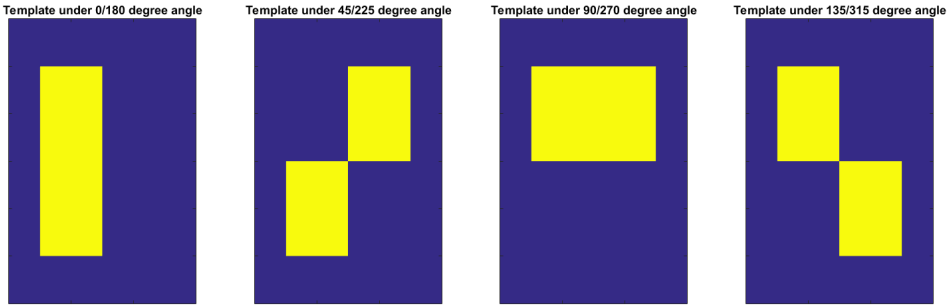

Figure 6.15: Car templates under different angles

6.4.2. Mapping

Mapping from the image coordinates to the environment coordinates requires the transformation function. The transformation function is a common problem in computer vision and can be done using the location and angle of the camera. For the current system, the cameras are virtual and the transformation is straightforward. All detections contain the coordinates within the image of the Centre of Gravity (CoG) of the car. The top right pixel in the image corresponse to the offset in the environmental map. Due to the specific angle and orientation of the virtual camera, the transformation function only moves the image coordinates with the same amount as the camera offset. The process is mainly included to simulate the real world situation, but is currently trivial. It does show that every local agent requires its relative offset on the map. The coordinates are provided in the initialisation phase by the delegator agent.

6.4.3. Recognition

The recognition process attempts to re-detect cars in new images. It uses the trajectory of the car, based on previous behaviour, to estimate the chance of the new detection is the same car. The process is visualized in figure 6.16. The model for recognizing local tracks based on detections and regional tracks based on track segments uses the exact same model. The only difference between the modules is the trigger and fusion of

the segments. We will first discuss the model and show the non-local adjustments afterwards.

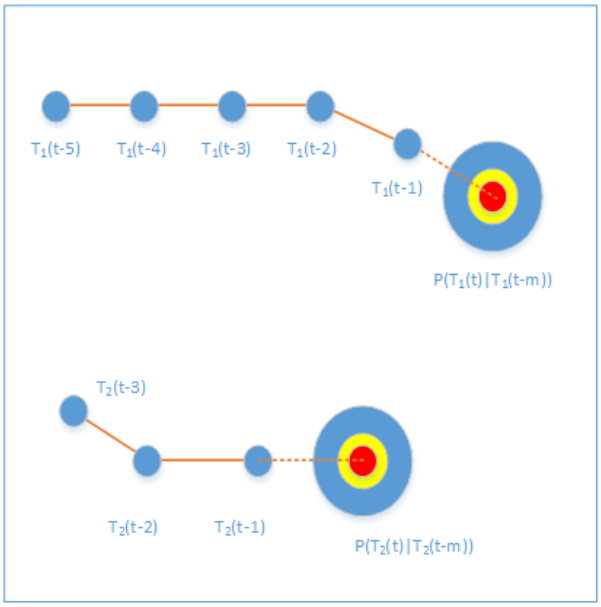

Figure 6.16: Regional information processing

Recognition model

The combining of individual detection or track segments is essentially the classification process of a detection to a track. Every new detection is added either to one of the existing tracks or creates a new track with just the single detection. This process is shown in figure 6.17.

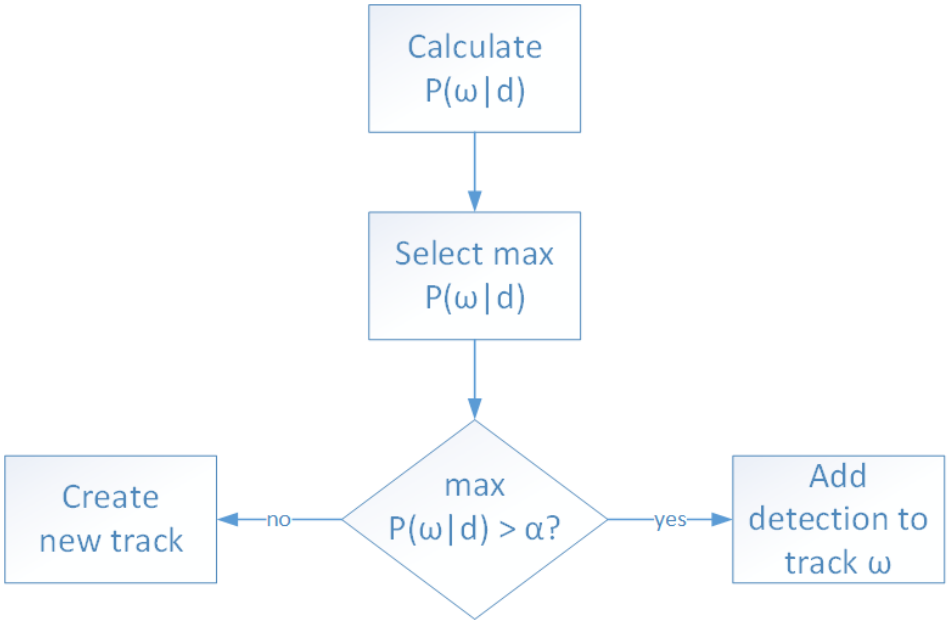

Figure 6.17: Program state diagram recognition process

A common applied technique uses the knowledge of the behaviour of the object (in our case the car) to evaluate if it is probable that the car would end up there. The classification process for the tracking calculates the chance that a new detection is part of an existing track ($P(\omega_i|\text{detection})$). Like in many classification

problems, this distribution is unknown and the Bayes rule is applied to compare the probability. We assume the prior probability of the detections and the car is equal in all situation, which allows us to consider the chance of the car proportional to the chance of detection, given the car. The track is assigned to the track that is most likely to have the detected features as the added component.

$$\begin{aligned} \Pr(\omega_i|\text{detection}) &= \frac{\Pr(\text{detection}|\omega_i)\Pr(\omega_i)}{\Pr(\text{detection})} \\ \Pr(\omega_i|\text{detection}) &\propto \Pr(\text{detection}|\omega_i) \end{aligned} \tag{6.1}$$

The likelihood of the detection is based on a model of the car behaviour. The model include the two dimensional location and the angle of the car, extracted from the tracks detected in the past. Using this information available in the track, the system predicts where it would expect the new detection. These predictions are used to estimate the likelihood of the new detections. We apply the following predictors:

$$x^*(n+1) = x(n) + v_x(n) \tag{6.2}$$
$$y^*(n+1) = y(n) + v_y(n) \tag{6.3}$$
$$\phi^*(n+1) \equiv (\phi(n) + v_\phi(n)) \mod 2\pi \tag{6.4}$$

x^*, y^*, ϕ^* are the values from the detection. We assume the new location X, Y and angle Φ are independent and all of them consist of a normal distribution around the predicted value. The angle of the car uses a Von Mises distribution, which is a normal distribution for circular data.

$$X \sim \mathcal{N}(x(n+1), \sigma_x^2) \tag{6.5}$$
$$Y \sim \mathcal{N}(y(n+1), \sigma_y^2) \tag{6.6}$$
$$\Phi \sim f(x|\phi(n+1), \sigma_\phi^{-2}) \tag{6.7}$$
$$= \frac{e^{\sigma_\phi^{-2}\cos(x-\phi(n+1))}}{2\pi I_0(\sigma_\phi^{-2})} \tag{6.8}$$

Due to lack of symbol, the Von Mises distribution will be represented as a normal distribution, unless it is unclear from the context. The variation of the distributions are filled in empirically. It is dynamic over time and the longer the track had no detections, the more the true value vary for the modelled value.

Non-local variations

The problem of combining track segments uses exactly the same principle as the combining of detections. Every time a car reaches the edge of a region (camera or guardian agent region), the car is expected to appear in the neighbouring region. In related work, we found this problem is often modelled as a source/sink problem, where the sink and source are connected. In our model, we know the coordinates of the environment map. The recognition proces looks at the first detection of the new track segment (also referred to as the head) and predicts the expected location of each car by the last detections in the track (also known as the tail). By storing the head and tail of the track, the car can be recognized on a regional level in the same way the local recognition is performed.

Regional recognition reasoning has multiple parallel processes and is a distributed system. Such systems tend to grow in complexity in terms of the number of states the process can be in. To show this aspect and gain insight in the possible states of the system, we analysed the system using the Linear Transaction State Analyser (LTSA). The tool allows developers to script the individual processes and deduct what the processes would do, once they they are runned in parallel. In designing a parallel process, it is important to analyse the possible states and identify potential error states with the purpose of eliminate them.

As discussed earlier, the tracks will only be send when the latest detection on the edge of the agent's region. The regional agent that is receiving these track segments is expecting both a sink and a source track. However,

as long as the source track is not yet recognized as the continuation of another track segment, the it is still unknown if it is a new track or a source track segment of an existing track. On top of that, there are two exceptions to these situations. The first exception is when the car drives of out of the surveillance area. In this cases there will be no source track (when leaving the area) or no sink track (when entering the area) available. The second exception is when the car is when the car is parking on the edge of a region. When the car parks on an edge, the stitch has to be extended, one detection at a time. These two exceptions can also occur together and we find these cases back in the states of the track. The regional track recognizing states are generated using the LTSA and given in figure 6.22 (see end of the chapter). Table 6.1 provides additional descriptions of the state transaction events.
What we can conclude from the state machine is that the information always becomes available between two time events. The system has to wait the sample frequency (in our case one second) until it can conclude what the track segment(s) represent. The recognition step (which is left out of the state diagram) recognizes an orphan track segment to be a source track. This happens for example the states sequences $0,10,11,3$ and $0,10,8,9,3$. Which path is chosen depends on the result of the recognition step. The setback of this method is that there is a second delay with each increasing hierarchy level, because the delay propagates through the levels. The good news is that it always is the same delay, no matter what state sequence is found. Higher level agents can apply the exact same recognition method, without having to consider the delay.

Table 6.1: Description of the regional track states

Event	Description
orphan	Track segment is received, but it is unknown to which parent track it belongs
sink_track	Track segment is received and recognized as the sink track of an existing parent track
source_track	Track segment is received and recognized as the source track of an existing parent track
onTimeEvent	Time-out event occurs and no more track segments will be received
new_track	Create a new parent track with the given source track
append_track	Add the new detections in the track segment to the parent track
append_stitch	Combine the latest detections from the track segments to add a new detection to the latest stitch of the parent track
append_detection	Add the latest detection from the only received track segment to the latest stitch of the parent track
stitch	Combine the last detection of the parent track and the source track to replace the latest stitch detection of the parent track
stitch_edge	Create a new stitch with a single track segment in the parent track

6.4.4. Feature extraction

Given the set of detection, the features can be calculated. The calculations of the features are straightforward and contain the measured features and first and second order derivative of the features. In all cases the forward Euler method is used to calculate the derivatives. Table 6.2 gives an overview of the features calculated from the set of detections.

Table 6.2: Summary of extracted features

Feature	Symbol	Calculation
Location	x and y	Known
Speed	$v_{x,y}$	$x[n]-x[n-1]$
Acceleration	$a_{x,y}$	$x[n]-2x[n-1]-x[n-2]$
Heading	ϕ	$\text{atan2}(v_y[n], v_x[n])$
Change in heading	v_ϕ	$\phi[n]-\phi[n-1] \mod 2\pi$

Dependency between recognition and feature extraction

The fact that the extracted features are used in the recognition step shows the dependency between the two steps. The required information is communicated using the historical database. As soon as the features are

extracted, the track is stored in the database. The recognition process collects the already analyzed tracks and uses them for the future detections or track segments.

Alternative recognition for small tracks

The calculation of the features depend on past detections. In the case of small tracks, these detection are unavailable. We consider three states of small detections.

New track

No historical detections are available and none of the calculations are available. The recognition is based on the ruling out of other tracks. The calculates features cannot be extracted and the detection provides two possible headings.

Existing track with one detection

Since only one detection is available, the calculated features are unavailable. For the coordinates, the variation is increased to counteract the missing information in the model. The heading at this point is one of two options, based on the previous detection. The change in heading is kept at 0 and the distribution is set on a mixture of the two Von Mises distributions as shown in equation 6.9. The feature extraction is unable to calculate the acceleration and change in heading.

Existing track with two detection

The acceleration and change in heading are unavailable. The recognition based on the coordinates can proceed as normal, but the change in heading is unknown. In this case, the recognition is performed as normal, but the change of heading is set to 0. Once the detection is recognized as part of the car's track, the detection forms the third detection in the track and the feature extraction can proceed as normal.

$$\begin{aligned} \Phi_{dl} &\sim \mathcal{N}(\phi_{dl}, \sigma_{dl}^2) \\ \Phi_{dh} &\sim \mathcal{N}(\phi_{dl} + \pi, \sigma_{dl}^2) \\ \Phi_{d} &= \frac{\Phi_{dl} + \Phi_{dh}}{2} \end{aligned} \tag{6.9}$$

6.4.5. Stitches, distances & data reduction

The stitches combine two track segments by stitching (mostly) the last and first detections of the track segments into the parent track. The stitches serve two purposes. By storing the origin of both child track segments, the stitches are used to retrace the steps and find where all the detailed information is still stored. Secondly, the stitches are used to store corrections made to the detection. Detections on the edge of the camera, covering only parts of the car in the image, have a chance of error in the angle and center of gravity. When two detections from different cameras are known to represent the same detection, the information from both can be used to correct the error, as is shown in figure 6.18. Once corrected, the travelled distance also has to be corrected, using the new findings. The correction is performed for the path leading up to the stitch and the path after the stitch. This is performed with the following simple equations:

$$\begin{aligned} &\text{distance}(t) = \text{distance}(T_1(t)) - \text{distance}(T_1(t-1), T_1(t)) + \text{distance}(T_1(t-1), T_S(t)) \\ &\text{distance}(t+1) = \text{distance}(t) + \text{distance}(T_2(t)) - \text{distance}(T_2(t), T_2(t+1)) + \text{distance}(T_S(t), T_2(t+1)) \end{aligned} \tag{6.10}$$

6.5. Reasoning methods

Reasoning methods are applied ones the detections and track (segments) are available. In section 4.4.1, we discussed the distribution of the cameras. Considered from the cameras, region size delegated to each agent increase with every level in the hierarchical model. The reasoning method and delegated logical reasoning for each agent is affected by level in the hierarchical model. We will briefly discuss how the reasoning methods are affected by the level. Afterwards we discuss the individual reasoning for the compiled scenarios and finally we discuss the tasks delegation for all agents.

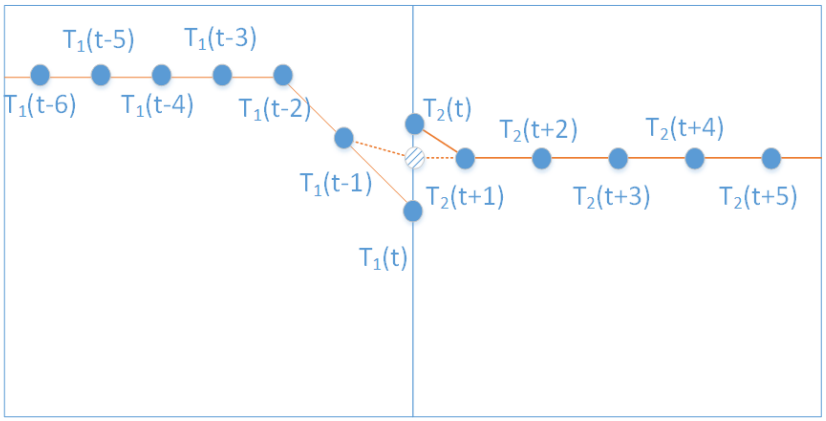

Figure 6.18: Correcting detection error and distance on stitches

6.5.1. Reasoning scope

The features used in logical reasoning will include the latest detections, the detections in the memory and the features extracted from the tracks. The further the reasoning includes samples from the memory, the larger the track will have to be in order to be able to perform the reasoning. The processing pipeline in the hierarchical design only flows in one direction, meaning the agents do not know the history of the track outside of the designated range. For example, cars that recently crossed from one camera to the next will only have a few samples in the latter camera. Only the regional agent will know the history of the track and even there it is possible only part of the track is known. The limitation of the track size force the logical reasoning methods to be delegated to different layers in the hierarchical model. This is shown in figure 6.19. The figure uses an example where the camera images are 8x8, the regional agents each have 2x2 cameras in its scope and the global agent has 2x2 regional agents in its scope. The example shows that the local agents become less capable to reason and the MCS depends heavily on the global agents. This in turn makes the MCS less scalable by creating a bottleneck.

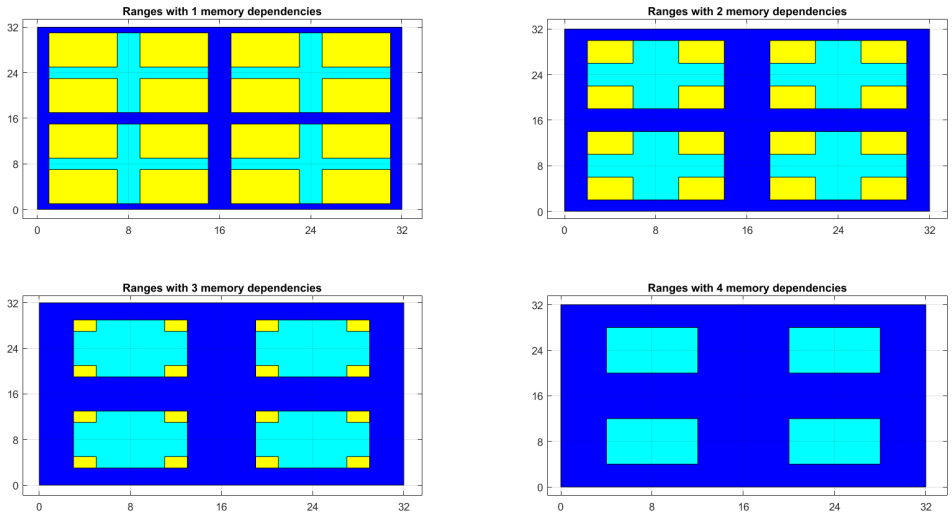

Figure 6.19: Memory dependency reasoning delegation

Another problem with neighbouring regions is the precision of the detections. Detections made on the edge of the edge of the camera are based on detections of parts of the car, in contract to the full contour of the car. Higher levels of reasoners are able to combine information to reconstruct the true location of the car. Therefor, some reasoning must be performed by the higher levels of agents, since they require the recon-

struction of the detections.

6.5.2. Scenarios

Each scenario will have a different reasoning method. The reasoning is performed with the CLIPS expert system, which is a rule based system. Since the reasoning is performed over multiple cameras, the classification not only consist of the reasoning of agents but also with the cooperation between the agents. The reasoning for each scenario is discussed in terms of the rules applied in the reasoning and the task delegation over the different cameras.

Unwanted entry

Unwanted entry merely looks at the location of a car and correspond it to the restriction map. The reasoning only applies to the current measurements and requires no further memory of the past detections. It is triggered by any detection on any camera.

if the car is on location l
and l is a restricted area
then there is an unwanted entry at location l

The reasoning is performed on any camera that can restriction locations in its sight. The reasoning scope only contains the local information and is performed by the local agents. No global or regional reasoning is required.

Unwanted parking

For this scenario, we assign certain locations on the map that do not allow you to park. It is however impossible in our simulation (and complex in a real life situation) to distinguish between a parked car from a car that is only standing still. We simplified the problem by defining the locations where it is in fact unwanted to stand still. The track information, received from the tracking process contains the speed of the car, based on the last two detections. The rules applied in the reasoning are the following:

if the car is on location l
and l is a no parking area
and the speed of the car is 0
then there is an unwanted parking behaviour.

Restricted direction

The one-way street is similar to the unwanted parking, since it also needs one more detection from the memory to be able to use the heading of the car.

if the car is on location l with heading $h1$
and l is a one way street in heading $h2$
and difference between $h2$ and $h1$ is π
then the car is driving in the wrong direction

Aimless driving

The aimless driving only appears on longer tracks and is only applied in global agents. The reasoning considers a maximum track length on all locations. Once exceeded, the reasoner gives an alarm.

if the car is on location l with track length of tl
and l has a maximum track length of mtl
and $tl \geq mtl$
then the car is driving aimlessly

6.6. Agent distribution

A key requirement of the scalability of the system is the equal distribution of processing demand over the intelligent cameras. For the local agents, the distribution is obvious, but the higher levels of the hierarchical

agents have not been assigned to any agent operating platform. In this section, we will discuss how the workload is spread over the levels of agents.
The local agents will each be assigned to one camera. The map created in section 4.4 is divided in size units of 2 metres. The total map includes 72 size units (or 144 metres) in the x-direction and 56 size units (or 112 metres) in the y-direction. Each camera has a range of 8x8 size units, each representing one pixel in the image. Each regional agent has 3 cameras in the x-direction and 3 cameras in the y-direction within its region, including a total of 9 cameras. The most southern regional agents only include 3 local agents, since the map does not include any more cameras in the y-direction. The global agents in turn communicates with all the 9 regional agents (3 in the x-direction and 3 in the y-direction).
The proof-of-concept application will run all of these agents on one JADE container, since the only difference between the real and simulated environment is the configuration of the containers. However, without a method to distribute the agents, the processing demand distribution will not be equal over the intelligent cameras. Therefore, we developed a distribution policy to equally distribute the processing power over the agent operating platforms.
In real environments, the delegator agent will run on a separate platform in the terminal. The delegator will create the other agents on the different cameras platforms. The delegator will keep track of what process is where. Although not necessary for our implementation, we suggest a policy for the agent distribution over the AOP. To validate of the distribution is optimal, we can check the maximum load balance. The maximum load balance is defined as $\lceil 1 + \frac{K}{N} \rceil$, where N is the number of cameras and K is the number of regional and global agents. When the regions are distributed based on heuristics, it is hard to create a general policy for the distribution of the agents and we advise to create the policy manually. However, when the hierarchical model represents a balanced tree, we could apply the policy given in equation 6.6.

$$AOP(k,l,N_c) = \begin{cases} k, & \text{if } l = 1 \\ kN_c^{l-1} - N_c^{l-2}, & \text{if } l > 1 \end{cases}$$

Here k represents the index of the agent in the layer, l represents the layer and N_c represents the number of children nodes for each parent node. Basically the maximum load balance in the balanced tree case is $\lceil 1 + \frac{\sum_{l=0}^{L-1} N_c^k}{N_c^L} \rceil = 2$. By applying this policy, the higher level agents will never be placed on the same AOP twice and the load balance will never be higher than 2.

6.7. Test environment

The system is tested using a custom made unit testing framework, named J2Unit or Java & JADE Unit testing. The framework is an extention to the JUnit testing framework, used in many software development projects. The framework is extended to fit the testing of multi-agent systems.

6.7.1. Challenges

Thorough testing is an art on itself and is essentially performing an experiment on every run. In our research, we want to test the functionalities of the multi camera system as described in the previous chapter. Unit testing can automate the experiments and provide insight in the flaws of the system. Although unit testing is a mature technique for sequential coding, testing parallel processes or multi-agent system is still a challenging topic.
Agent based systems assume the agents to perform autonomous. This greatly influences the testability of the agents, since they are designed as a black box system. The only method to test the functionalities of an agent is to communicate with it or analyse the result in the user interface or database. By design it is not allowed to interrupt the process for any external purposes, including testing. The JADE framework starts the agents by reference, meaning the JADE container is responsible for the lifecycle of the agent and initially only allows the agents to be started from the main process. The challenge in testing agents is to make the agents into testable processes, while at the same time honouring the autonomous property of the agents.

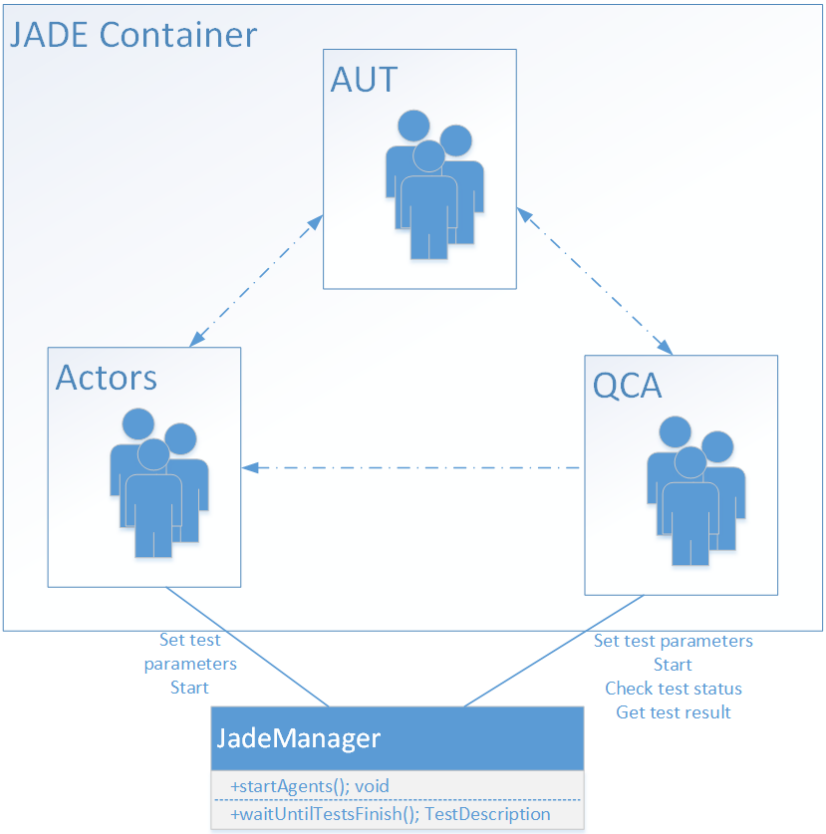

Figure 6.20: Agent types in JADE test environment

6.7.2. Mock agents, actors & quality control

We developed a method that allows multi-agent systems to be tested, with minimal invasive techniques. A commonly used technique in software testing is the use of mocking classes. The mocking classes perform the same actions and share the same interface as their real counterparts, however they to not perform any of the information processing that the real implementation would perform. They can be used for testing to validate the functionality of other software components. By calling functions the mocking classes check if the components under testing produce the expected results or can check if the components under testing call functions at the right time (and with the right parameters). The mocking classes themselves form no useful aspect of the final product, but can give insight in the functionalities of the components under testing.
Mock agents (MA) form a similar addition to the multi-agent system. Although they do not add anything to the functionalities of the multi-agent system, they communicate with the agents under testing (AUT) using the same methods the AUT use. The AUT have no method to check if the system is used for testing or for real environment. It processes the information as normal. The mock agents in turn can validate the information from the AUT and notify if the test has passed or failed.
Mock agents solve the problem of autonomy of the AUT, but the MA are agents themselves and by design are autonomous. We need a method to listen in to the MA to check their conclusions and possibly set parameters. JADE does provide a method to do so. It is possible get the reference to the live agents through the agent's controller, provided the agent has a registered agent to class interface. Any agent that is called from the unit test environment has to register and implement corresponding agent to class interface. This is strongly discouraged, because it both changes the functionalities of the agent and reduces the autonomy of the agent. For our mock agents however, this is no problem, since they are not the agents under testing and their functionality is not in question.
Our design of the multi-agent unit testing environment uses two types of mocking agents: *actors* and *quality control agents*. The actors only pretend to be agents that in the real environment would communicate with the AUT. They are used to trigger an event in the AUT by communicating the scripted messages. The Quality Control Agents (QCA) validate the result. Every QCA is responsible for performing one or more tests and val-

idate if the messages from the AUT contain the expected information. A multi-agent unit test succeeds when all the QCAs report a success in there tests. The UML Class diagram of the mock agent design is shown in figure 6.21. The TestCollector interface is used internally to store the test result.

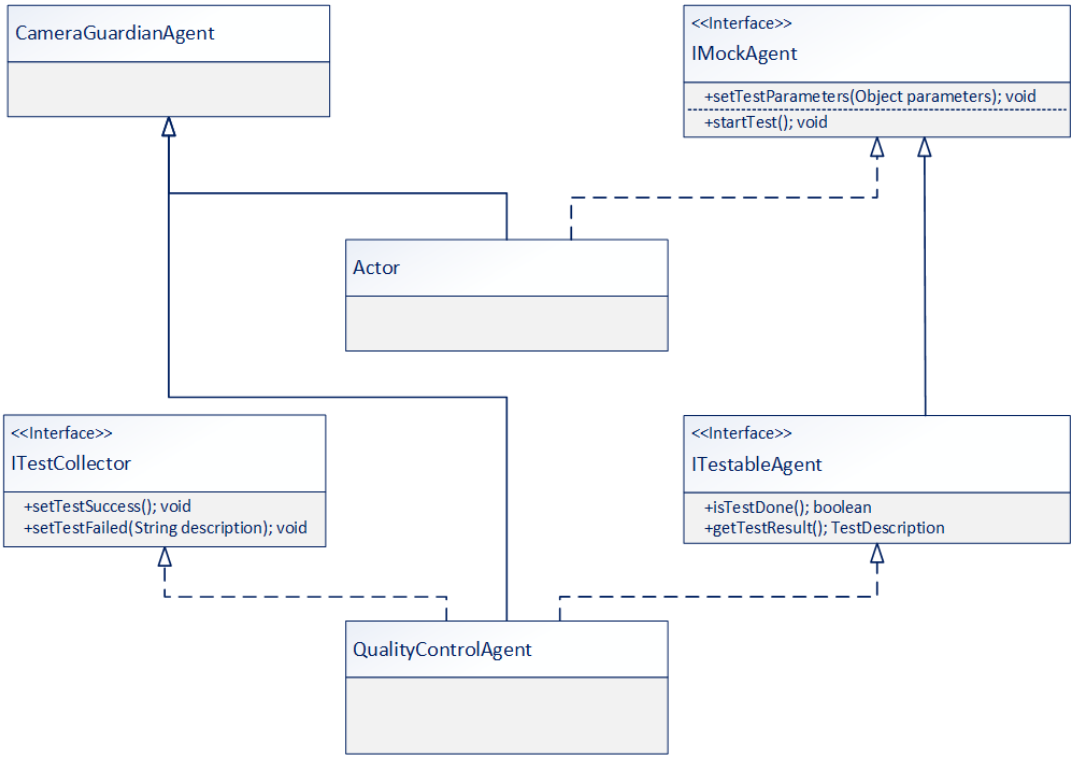

Figure 6.21: UML Class diagram mock agent design

6.7.3. Test components

With the mocking agents discussed, we have all the tools to define the test environment. Back in section 4.3.5, figure 4.6 we already gave a sketch of the components and their interaction. Every feature test discussed in section 5.2.1 will have its own unique composition of the following components:

Test data generation script

The car locations, car tracks and any test parameters used are generated for each test. This is automated to ensure the data is valid for the test and randomly chosen to reduce the probability of a hidden bug.

Test data

The test data is provided to all the simulation and test components, to allow the test scenario to be simulated and validated.

NetLogo simulation

The NetLogo simulation file is unique for each test, although it is similar in almost all situations. The simulation plays the car locations, similar to a video is being played. The main difference for each situation is the car location, which are given in the test data.

Quality Control Agents

Every test has one or more QCA. Each of these agents has a checklist of tests they have to go through in order to validate the outcome of the AUT. The test only succeeds when all the QCA give a success respond. The QCA interact with the AUT by means of the event log system. In the initiation phase, the QCA subscribes to all the process events on the AUT to be able to validate the outcome. The validation is also based on the test data given from initially by the test environment.

Actors

The pipeline designed in the hierarchical model processes the data in one direction. Testing regional and global agents would normally require all the predecesors to be functional before the agents functionalities can be tested. The actors allow us to test the regional and global agents separate from the local agents. The actors will pretend to be the local agents and send predefined information to the regional agents. The QCA in turn will listen to the results from the processing and continue the test as normal.

Agents Under Testing

Although isolated, the AUT must be configured the same way as it would in the fully functional system. The test is only valid when there is no difference between the real situation and the test simulation.

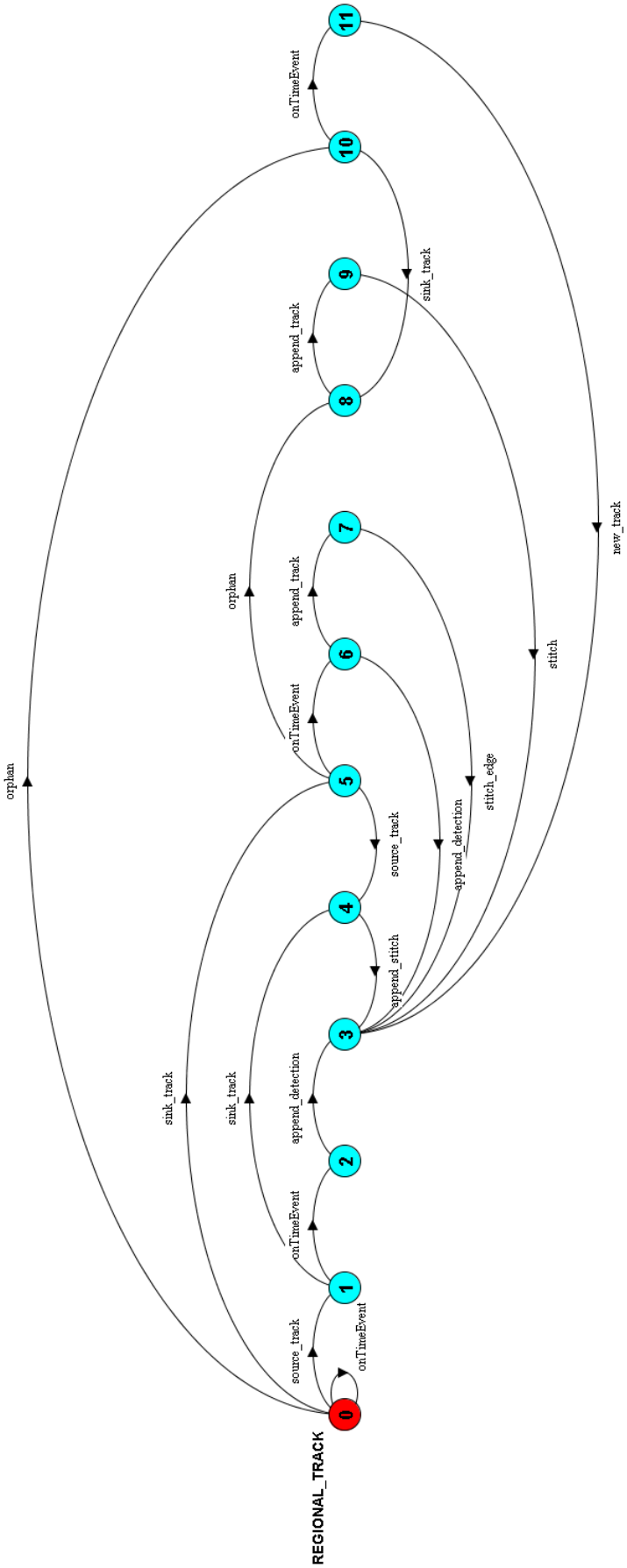

Figure 6.22: State representation regional track events

7 Implementation

The focus moves from the plan and design to the actual implementation of the MCS. The chapter implementation discusses how the design was realized, what problems were found during the implementation phase and how they were solved. Going through all the lines of code would defeat the purpose of this thesis. Instead, we will discuss key aspects of the implementation and show the complications (and solutions) found during the implementation.
A key element in the implementation is the way the agents communicate with each other. A major part of this research is focused on the task delegation. Only through well defined communication between the agents would it be possible to setup the required information flow and allow the agents to perform there tasks. We start the implementation description with the method of communication. Once established how the agents communicate with each other, we will continue by looking into the data processing and implementation of the pipeline components. The pipeline is the beating heart of the implementation, which runs through the local, regional and global agents and analysis the stream of data to reason about the perceived behaviour. We divided this topic into three parts: initiation, tracking and reasoning. What remains is the implementation of the alerting agent.

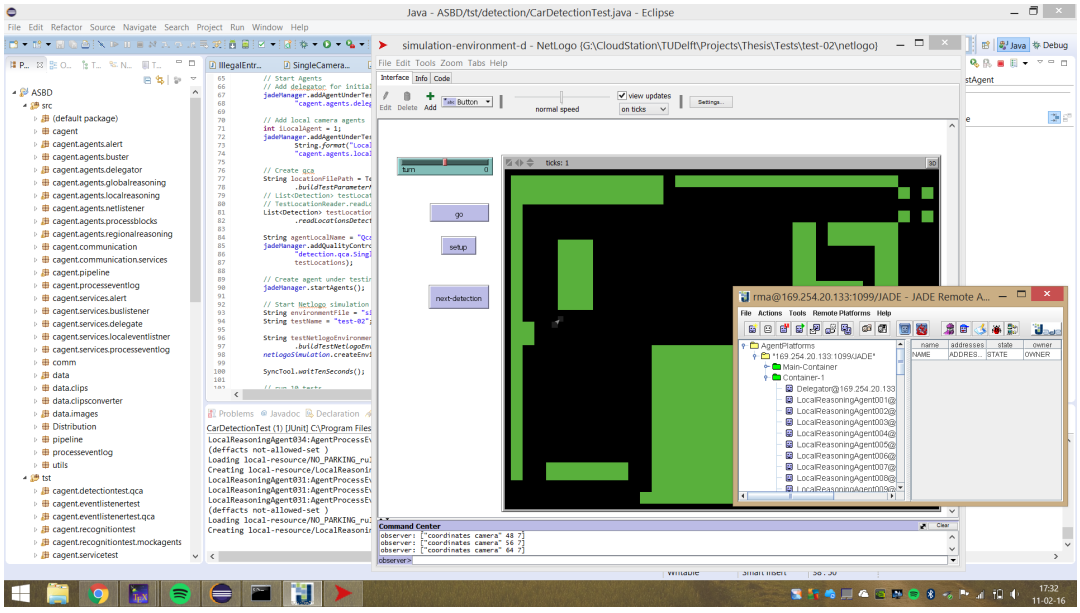

Figure 7.1: Classes remote function call

7.1. Guardian agent behaviour

The guardian agent design in section 6.2.1 shows the concept of a new generic agent, including the components used in the generic camera behaviour agents to communicate, process and log data & events. The new generic agents is created as an extension of the JADE agent and contains all the components required for the new features. Because it is the central component for all the processes that happen in the agent, it is designed as the central switchboard for internal communication between the components. Creating a new agent from the basis of the guardian agent instead of the JADE agent equips the new agent with all the features discussed in section 6.2.1. All the agents within the design of the MCS are developed as guardian agents. This relation is shown in the UML class diagram of figure 7.2.

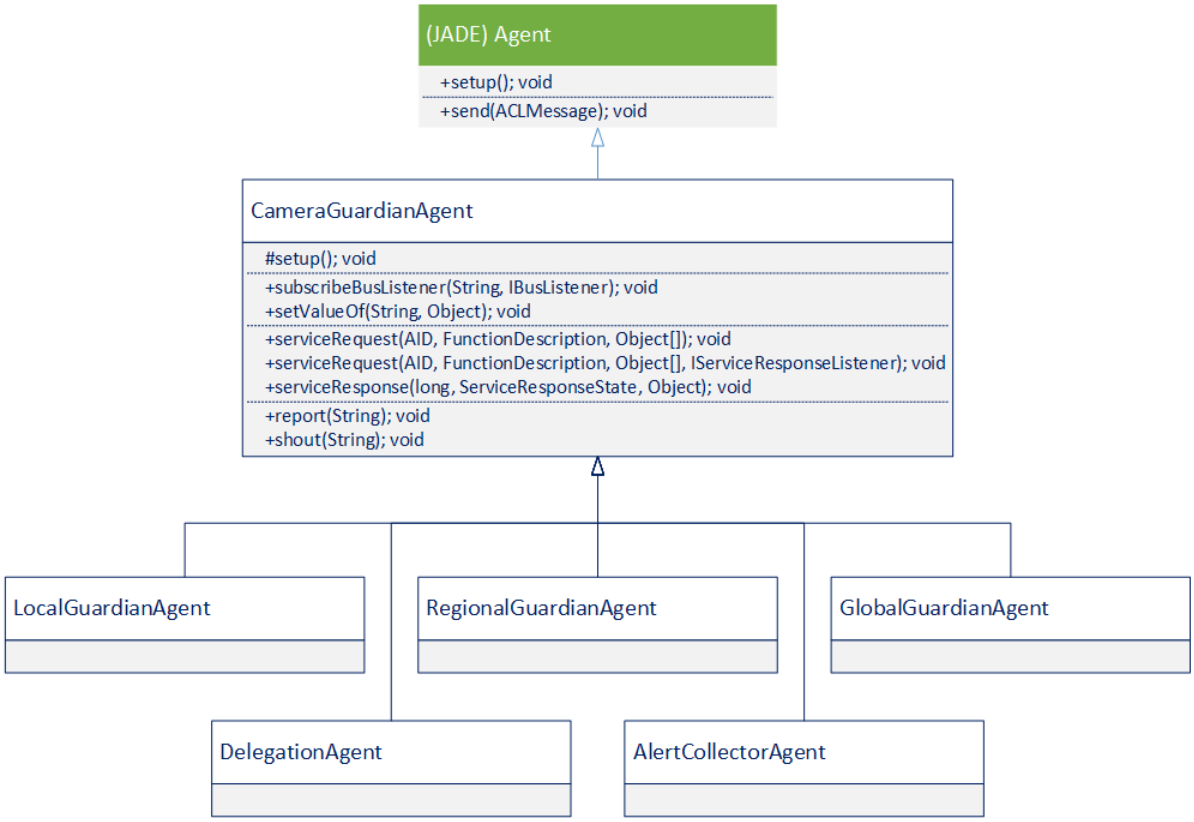

Figure 7.2: Guardian Agent uml

The following sections will discuss the generic components within the new generic agent. But before we continue with this description, we want to quickly refer to appendix A & B. These appendices describe two well-known techniques, which both are used intensively in the guardian agents. Appendix A describes the subscription model, which is used to create an event-driven environment throughout the system. The subscription model is used in all generic components to trigger (amongst others) processing blocks, event validation and service calls. Appendix B describes the chosen solution for generating unique identifiers in distributed databases. Although there are other methods to solve this problem, we feel this solution was both easy to implement and sufficiently solved the problem. The identifier generators are used for both the local databases within the agents, as well as the session identifiers used in the service calls.

7.2. Agent services

The agents are designed to support features and services. The services are implemented as Java classes with functions that are made available to all agents (including itself) and can be called remotely. The main differences between services and regular classes are the asynchronize and serializable properties. All the function calls of a service are encoded to a standard data structure and communicated using a three layered com-

munication protocol. We will describe the implementation of the services and communication using a top down approach. First we will discuss the procedure and classes required to build a service. Then we continue by describing how the service layer uses sessions to handle the logistics and process control. Finally we will shortly describe the transport layer and packaging method.

Layer principle

The communication is build up from three layers. The layer principle states that each layer supplies features for its parent layer and communicates only with its child layer. In our case, we define the application layer, the service layer and the transport layer. The three layer communication is implemented in every agent and every message from one agent is encoded top down through the layers and decoded bottom up on the receiving end. The layer principle implicates that the layers on the same level effectively communicate with each other, without the knowledge of how the lower layers encode and decode the messages. As a result, the transport layer communicates with the transport layer, the service layer communicates with the service layer and most important of all, the application layer effectively communicates with the receiving application layer. Essentially the layer principle allows two agents to communicate with each other, without knowing how the message ends up on the receiving end.

7.2.1. Building services

Any Java class can be converted to a service, but requires more components than just the class itself. Interfaces in Java can be used to separate the external functional description from the internal implementation of the features. They can even be used to dynamically change the functionalities of classes by instantiating different implementations of the interface, based on the state of the process. For the services we apply the interfaces to separate the functionalities from the actual process that will perform them. At the moment of the service call, the client process has no idea what process will perform the requested service. It only knows it will be handled. We will discuss how a regular class can be converted to a service in our system. The procedure unfortunately grows more complicated when functions have a return value and for clarity we will first discuss the procedure for functions without return value and later extend it.

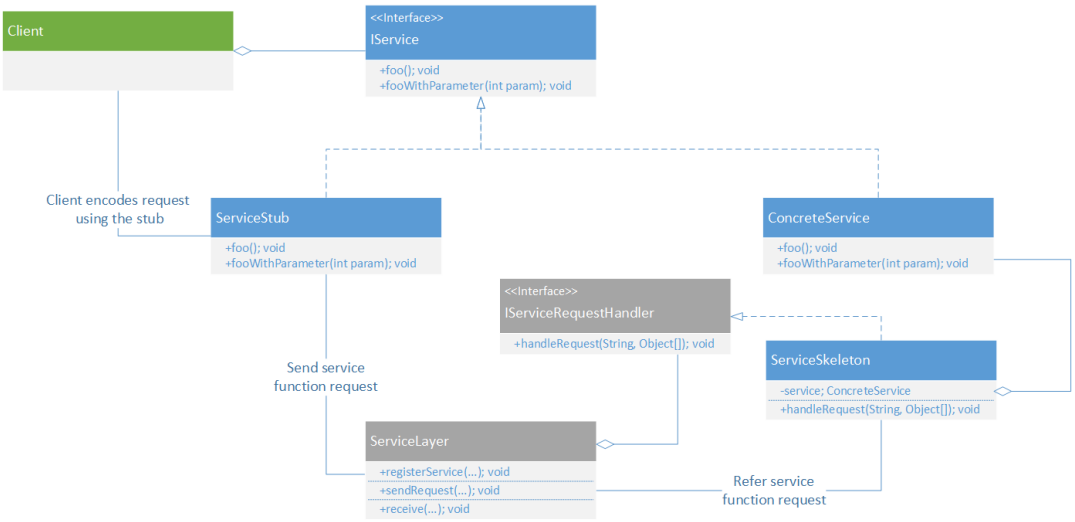

Figure 7.3: Template classes involved with agent communication

Building a service requires at least four new components, which are shown in blue in figure 7.3. The service layer is present in all agents and will be discussed in the next section. Building a service starts with the definition of the functions of the service. The service functionalities are defined using a service interface, which is known on both the server and client side. The service interface is a (Java) class interface which shows all the functions included in the service. Any service provider must implement these functions, which is also the second component. The service provider ultimately executes any of the functions when they are

requested. The service provider is registered with the service layer to indicate it can handle any requests for this service.
Before remote agents can call the service functions, the call has to be encoded to and decoded from a message of the type that can be send between the agents. The encoding at the client side is performed by the stub component. The stub component implements the functions of the service interface, but instead of performing the service, the stub encoded the request into a request message. The request message is send with the use of the service layer to the known agent that provides the service. The service layer is present at both the client and server agent. When building a service, the service layer is considered to be the gateway to the services on other agents and will send the messages on the client side and receive them on the server side. On the service providing side, the agent uses a skeleton to decode the message and call the appropriate functions on the service provider. The cooperation of these five components allow the call for functions of services on remote agents. However, when the functions have a return value the service layer becomes more complicated, which will be discussed shortly. First, we will look at how the messages are encoded and decoded within the stub and skeleton.

Encoding

The goal of the encoding is to communicate all the information required for the service provider to know which service must be called and with what parameters. The data structure used for communicating the request information is given in figure 7.4.

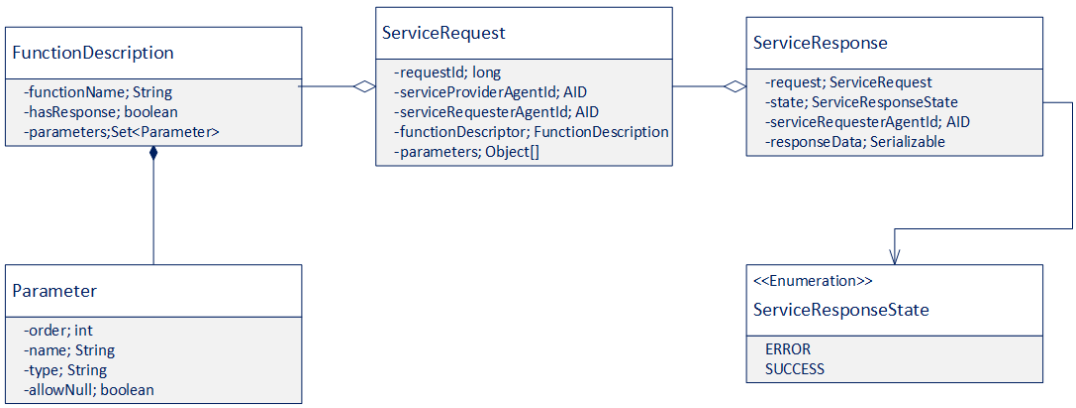

Figure 7.4: Data classes to represent the call and result

A service request contains the information of the function it was to call, the value of the parameters and the sending and receiving agent. The Agent IDentification (AID) is used as destination and origin address. The service, function and parameter description are currently only used as reference, but may be used in the future for validating the data and converting the values. The translation from Java classes to the data structure is shown in figure 7.5.

The approach for that functions with response value will be explained in the next paragraph, but for now it is enough to state that the response value from the function call is also wrapped into a message. The important elements in the response message is the response value and the indication of success of the call. If the service request was unsuccessful, the response state is set to "failed" and the response value remains empty. In order to trace back what the original request was, the request itself is added in the response, including the request identification. This identification number is generated by the service layer and tracks the request from original request up until delivery of the response to the client. The main objective of the stub and skeleton is to translate the original request and response into these data structures.

Service function response

We discussed previously that the complexity of the service request handling and building of the service increases when the function call include a response value. Now that we have laid the groundwork for building services, we will explain how the services also can support functions with response value.
We start with pointing out that all service calls are asynchronous. A consequence of this property is that the

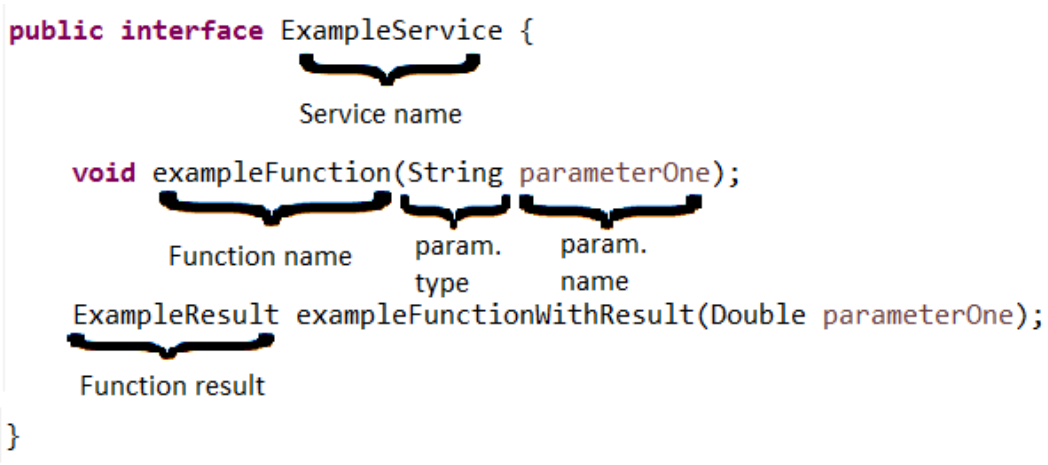

Figure 7.5: Information translated into function descriptions

function call by the client is non-blocking and the client process will continue, while the server handles the service call. In some cases it is possible to emulate the synchronized property by blocking the client thread, but this is not supported in our implementation. The client process will continue and service functions with response values will be received and handled after the call is processed and communicated. Meaning all functions with response value will be handled using event handlers.
Every function within a service that has a response value, will have an event that can be triggered when the response is received. For example, the example service given in figure 7.5 will have an event when the response value of exampleFunctionWithResult is received. This particular function actually could never be implemented this way, since the response will not be available directly after the function call. It is given in the example with the purpose of explaining the principle. If we wanted this particular function in our function, it would have to be defined with an event handler as shown in figure 7.6.

```
void exampleFunctionResult(Double parameterOne, ExampleServiceListener resultListener);
```

Figure 7.6: Remote function call with reponse event listener

The event handler is implemented differently on the client side and on the server side. On the client side, the client code is responsible for implementing the event listener interface on a class. The functionalities of this implementation is up to the client application, since there are no restrictions on what the client application wants to do with the data. As long as an entry point is provided by the client code. On the server side, the event listener is implemented with the task to encode the response value to a message that can be communicated and send back to the origin of the request. The response is send back with a reference to the original request, through the service layer. The UML diagram of the classes needed to build a new service with response value is given in figure 7.22.

Factory method

Building the components is vulnerable to configuration errors when the services are build on different places within the code. By using a factory method, the code ensures the service is always created and configured the same way. It has become good practice for building a service to add a factory method, that builds the service provider, stub and skeleton for each service.

7.2.2. Service layer

The services have different implementation for each service, but they all run on the service layer as service providers. The service layer is essentially a library of service providers, which manages the life cycle of the service calls. The registration of services uses a subscription method. The incoming messages feature as events, which are classified by service and send to the corresponding service provider. In the case of a response value,

a session is started and managed by the service layer.

Service behaviour

Sequential implementation of the service layer would hold any processing until the service function is processed. Other service requests cannot be processed for the duration of the function. The performance of the service layer is best kept independent of the performance of the service itself. Long durations, process holds or crashes of the processing of the service in a sequential implementation would cause bad performance of the service layer. In the implementation, this is avoided by applying the JADE behaviour system. The JADE behaviours create further parallelism within an agent. The service layer starts a "OneShotBehaviour" for each service request. The service processing is performed on a separate thread and the service layer can continue independently.
The services can be executed remotely, but because of it, all the functions are performed in a non synchronized manner. Functions with return values cannot be returned to the requesting agent directly and require an asynchronous approach. Returning data within an asynchronous approach apply event listeners to be called when the process is finished. These service listeners component are added to the service package. Any service that has functions with return values must have an interface defining the handler functions to be called in case of a service function request response. In the case a service function with a return value is called, the requesting party must provide a handler for the response value. On the server side, the returned value now has to be sent back to the agent that requested the function.

Sessions

Once the service layer is operational, many clients could make a service request simultaneously. When the response is available, the response has to be send to the client. For the purpose of bookkeeping, each service request start a session and the service layer generates a unique identication number to track the session. This unique identification number is generated at the initiation of the request on the client side. The session number is used in multiple agents and faces the same problems as in a distributed database. For this reason, we used the unique number generation technique, described in appendix B.

7.2.3. Transport layer

The transport layer is responsible for transporting the messages from one agent to the other, without any session management. JADE already provides a communication system that allows agents to send serializable messages to each other with the use of the AID as an address. We have build a transport layer on top of this communication system to work with our own messages. We had to decide how we were going to encode the messages and how the system would manage the received messages. We will discuss both items.

Encapsulating messages

JADE provides a method of sending serializable objects between agents. The layered communication method has different messages on each layer. Each layer uses a message structure of header and content seperation. The transport layer for example has the unique identification and the receiving address in the header. The content can be any serializable object. The session layer uses this to create session messages with different headers, including the information it needs to operate. Each inferior layer encapsulates the message and adds the information it needs in the header, creating a nesting doll effect. The result is shown in figure reffig:message-encaptulation.

Transport layer behaviour

The transport layer has the task of sending and receiving messages. This already straightforward task is made even more simple by JADE, since it allows serializable objects to be send between agents. The implementation of the receiving procedure is adjusted to the JADE environment by the use of JADE behaviour. This way the process of receiving message can be run in parallel with other processes. The transport layer does replace the existing methods of communication provided by JADE. In order to support the traditional method of communication, we implemented a subscription system, which calls all subscribed observers when the message is not recognized as a transport layer message.

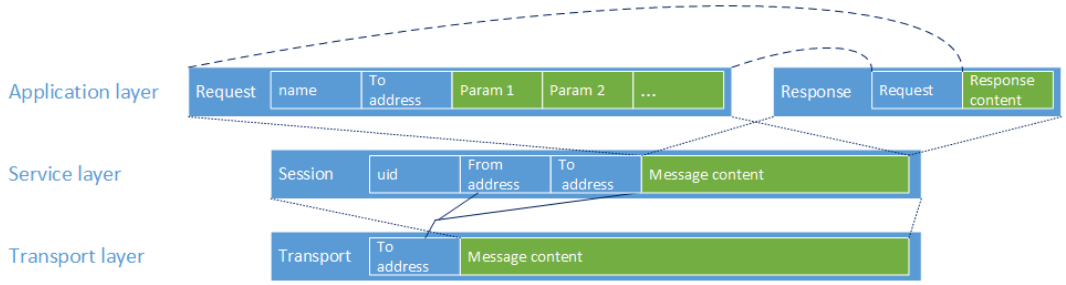

Figure 7.7: Nesting doll effect for messages

7.3. Agent event log

Initially the event log was a method of debugging the system, but it soon proved to be the best method to automatically test the MCS. We first build in a subscription system to connect different log event listeners, such as console output and file output. The local log event listeners have different levels of notification: *debug*, *info*, *warn* & *error*. The agent event log classifies the incoming events and only reports to the appropriate log level.

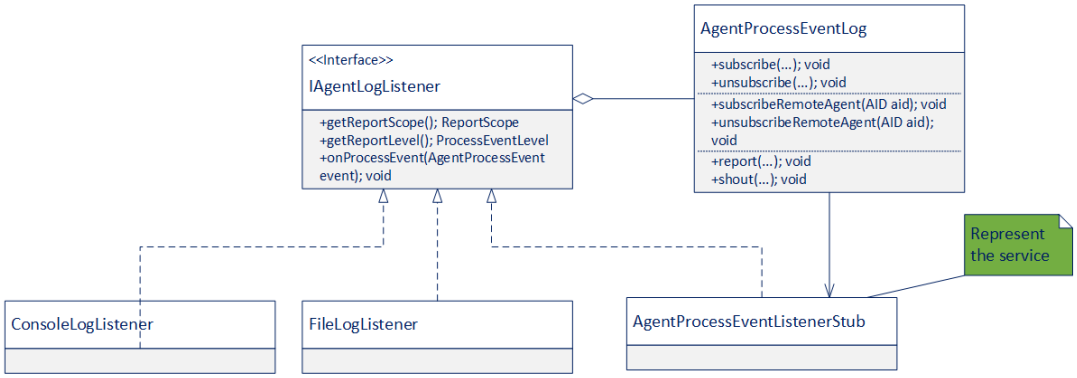

Figure 7.8: Event log UML class diagram

The event log is extended with the report scope. There are only two type of report scopes: *local* & *global*. The event listeners situated locally within the agents are notified of all events, whereas external listeners are only notified when the event has a global scope. External listeners are made possible through the services discussed numerous times already and the event listener service is implemented using the same technique. All the components involved in the agent event log are shown in figure 7.8. Finally, the separation between local en global events are represented using the report and shout function. The idea behind the shouting mechanism views the agents as a cooperative group. When an event wants to make a personal note, it creates a small report. When it wants everybody to know about the event, it shouts to everybody who is listening. The shouting mechanism is mainly used by the test environment, but can be used for any event that needs to be logged or notified to other agents. Our quality control agents subscribe to the global events of the agents under testing to validate the event with the expected test results.

7.4. Agent data processing

The processing structure described in section 6.2.2 contain a distributed processing pipeline. On this pipeline we find multiple processing steps, which starts with the images from the cameras and finally classify (suspicious) behaviour of the cars. The pipeline implementation is based on the same principles as OPC. All the values are stored on a databus and provided with a subscription algorithm. This section discusses the implementation of the pipeline, including the implementation of the distributed pipeline. But the pipeline is extended with data logging components, which allows not just the current data to be stored, but also the his-

torical data. The data logging and corresponding database structure will be discussed as well. And finally, CLIPS uses a different syntax to store the data (or known in CLIPS as facts). We developed a mechanism to convert the data objects used in our implementation to CLIPS fact syntax. This mechanism is discussed at the end of this section. The knowledge on the data processing will be used in section 7.6 to discuss the concrete implementation of all the processing blocks.

7.4.1. Pipeline

The pipeline uses a data bus as a memory storage. It is designed to standardize the data distribution and changes the system to an event-based processing system. Typical for the data bus is that every variable only stores the latest value of the variable. By default, the historical values are not available. The data bus components are given in the UML diagram in figure 7.9.

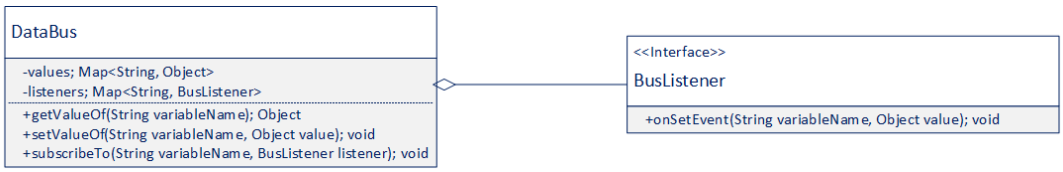

Figure 7.9: Data bus components

Immediately we recognize the subscription design discussed in appendix A and used already in the service model and event log model. Exactly the same principle applies: observers can subscribe to the events on the changes of a field value. Because the data bus only stores the last value, it is always a new value event. Bus event listeners are able to subscribe to state change events on individual variables. With the use of the data bus all components in the pipeline now become data driven calculators, which transform input values into output values. A component subscribes to all the input values, to get an update on each new value of the variables and the components themselves produce new values for the output variables. Using this implementation, we create a chain of calculators that form the pipeline. The data bus uses the JADE behaviours to call in the state change event listeners, similar to the technique used in the service layer. As a result, all components run in parallel, but are also running sequentially, because of the data driven triggers for each component. The technique is inspired by the streaming concept in massive parallel programming [23].
The data bus implementation supports local processing pipelines within an agent, but can also support distributed pipelines by implementing the bus listeners as a service. Shown in figure 7.23 is the UML class diagram used for the implementation of the bus listener as a service. Since the bus listener already is an interface and each pipeline component handles the data input differently, the skeleton does not use the concrete service, but only knows the bus listener interface of the pipeline component. The databus and service component form the basis for the distributed pipeline.
The only missing information is the name of the variable the listening agents wants to listen to. The delegator agent provides this information during the initiation phase of the agents. It is the only process that has to overview of all the agents and tasks, so all hierarchical agents will receive these variable names in the initiation phase.

7.4.2. Agent database & data logging

Every guardian agent is equipped with memory in the form of a database. The data bus only include the latest value for each variable bus and also has the historical data available, we need a form of memory. We decided to use a data logger system for the data bus to store the past variables. The database creates one table for each variable and has no prior knowledge about the data structures of the values themselves. But the table entries do need unique identifier or primary keys to separate between the entries. In this section, we while discuss the data log system, the identifier generation and the table query mechanism. The features are summarized in the UML class diagram given in figure 7.10.
The concept of the data logger is to store all the values that pass through the data bus. This is realised by creating a database that acts like a process block, meaning it subscribes to all variable store events. Every variable gets a seperate table in the database and is stored as a primary key and value tuple. All the values

will be of the type UidData, representing this tuple. Because of the map structure, values with new identification are added to the table, will values with existing identification will replace or update the values. The dependency of the UidData type affect the flexibility of the data logging system, since the data bus does not require the data to be anything other than a plain old java object (POJO). The flexibility returns by introducing an UidData wrapper. Any values received by the bus listener is checked if it is an instance of the UidData. If indeed it is, then the value is stored as described. If the value is no instance of the UidData, a wrapper class is created with a new identification. The value can be stored as normal, but since the identification always is a new number, the value is never replaced. In the current system, the detections are tracks of the type UidData and will be updated based on identification. All other variables will simply be stored as new values every time it stores a new value in the data bus.

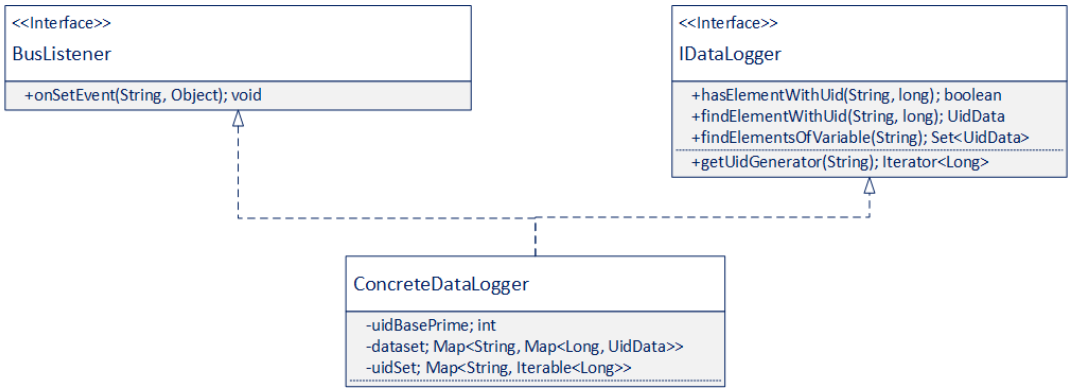

Figure 7.10: Datalogger UML class diagram

The unique identification generation uses the method discussed in appendix B. Every variable has its own identifier generator, which can be collected by the process blocks that create new values. Currently the queries only work locally, but the system is designed to be combined to create a distributed database. Each local database has part of the complete database and the identification will always be unique. Future versions will be able to query the databases using the distributed implementation. The local queries are shown in the IDataLogger interface in figure 7.10. These queries are used by the process blocks to gather the historic data. Through these queries, the database becomes the memory for all process blocks and avoids the necessity to have memory in the process blocks themselves.

7.4.3. Event-driven processing blocks

We mainly discussed the data driven events for the processing blocks, but we also mentioned the support of time-driven event. In fact, processing blocks can have multiple triggers configured. The UML class diagram in figure 7.11 shows how the processing blocks are implemented. The trigger holds all the information required for the agent to know want events the processing block should be subscribed to. Instead of subscribing manually, the guardian agent allows registration of processing block and based on the trigger information, the guardian agent subscribes the processing block to the different events. In the case of the periodic time-event trigger, the guardian agent adds a TickerBehaviour to the JADE agent behaviours, calling the "onTimeEvent" function for each periodic time event. The time-out event can be registered by the processing block itself at any time during the process.

7.4.4. CLIPS reasoning blocks

CLIPS environment has its own method to define variables and execute rules. We created a processing block that adapts the CLIPS procedures and data representation to the guardian agent model. We will discuss these steps and conversion in this section. For the encoding and decoding from Java code to CLIPS facts, we created a custom protocol to automatically convert the information. We will start with a short general introduction of the CLIPS concepts, continue with the adaption to the process blocks and finally discuss the conversion protocol.

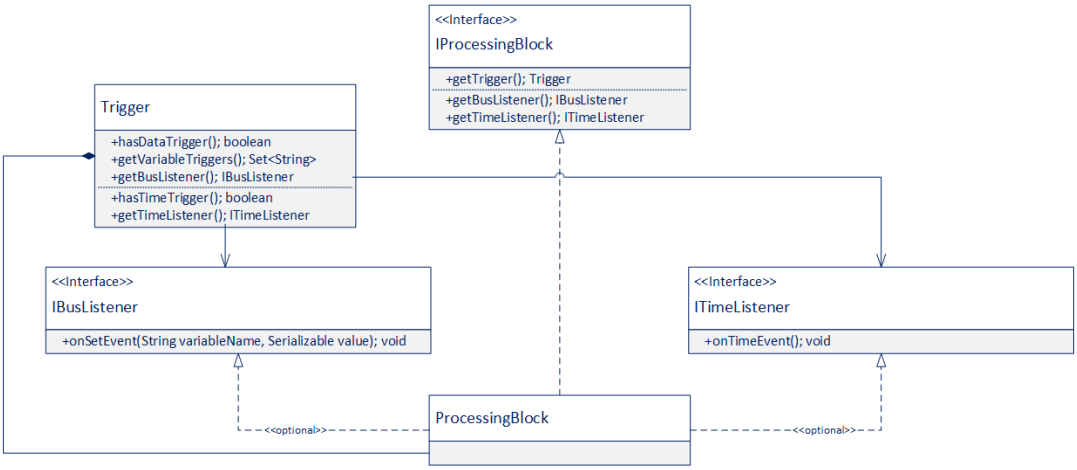

Figure 7.11: UML processing block template

CLIPS is an expert system, which uses rules to deduct information about the prior knowledge. Initially CLIPS has knowledge about the environment, such as the detection within one camera. When it starts to run the reasoning, CLIPS will go over all the rules that apply to the knowledge and update the knowledge base with the newly deducted information. The new information can again trigger other rules and CLIPS continues to run the epochs, generally until no more rules apply. The CLIPS API allows us to perform these steps, as long as we provide the knowledge.
The CLIPS processing blocks have an initiation phase and an execution phase. The latter is triggered by the set event from the data bus. In the initiation phase, the processing block defines the facts that are static in the environment. The feature 'deffact' allows us to define facts that are asserted in the knowledge base, every time the environment is reset. The rules are independent of the knowledge and will not be forgotten by the environment, unless they are specifically removed by using the clear or undefine command. What remains is the syntax of facts themselves.
Before we can exchange the JAVA Plain Old Java Objects (POJO) from Java to CLIPS, we need to ensure that the two environments understand each other. By defining the fact templates in CLIPS, we follow the same principle as the class definition in Java. Each POJO we want to include in the reasoning process needs the fact template before it can be exchanged with the CLIPS environment. The conversion of the object itself is a straightforward process of storing all the values on the designated position in the string. Our protocol uses a CLIPS converter for each POJO that is used in the CLIPS reasoning. The components involved in the conversion are shown in figure 7.12. The CLIPS manager provides a look-up function for the supported classes.

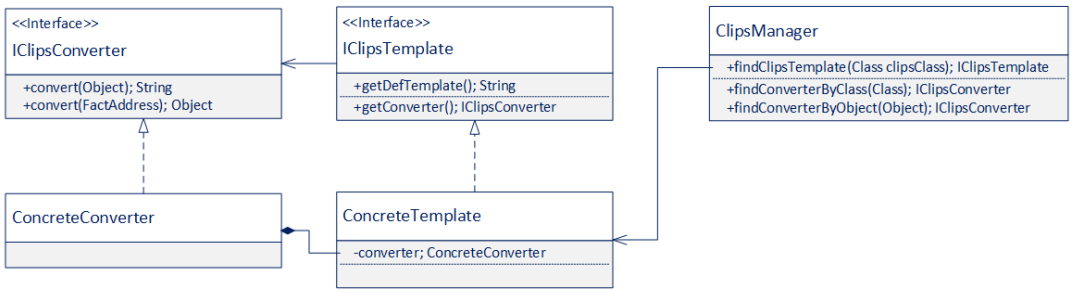

Figure 7.12: CLIPS conversion model

Using these components, we can initiate the environment to reason about every cycle. The rules and initial facts are defined, including the parameters received from the delegator to indicate the regions where actions are illegal. Once stored, the CLIPS can reset to this initial state and run the reasoning again. The facts that apply for the current cycle are converted in every new cycle and added to the fact base before the reasoning starts. Finally, when the CLIPS reasoning concluded there have been an illegal action, the alarming fact is collected from the CLIPS environment and converted back to the POJO. The Java wrapper uses these facts

to notify the alert agents of illegal facts. This summarizes the CLIPS conversion process. The specific implementation for each reasoning processing block is given in the description of the reasoning block themselves. The general process is shown in the UML sequence diagram, shown in figure 7.13.

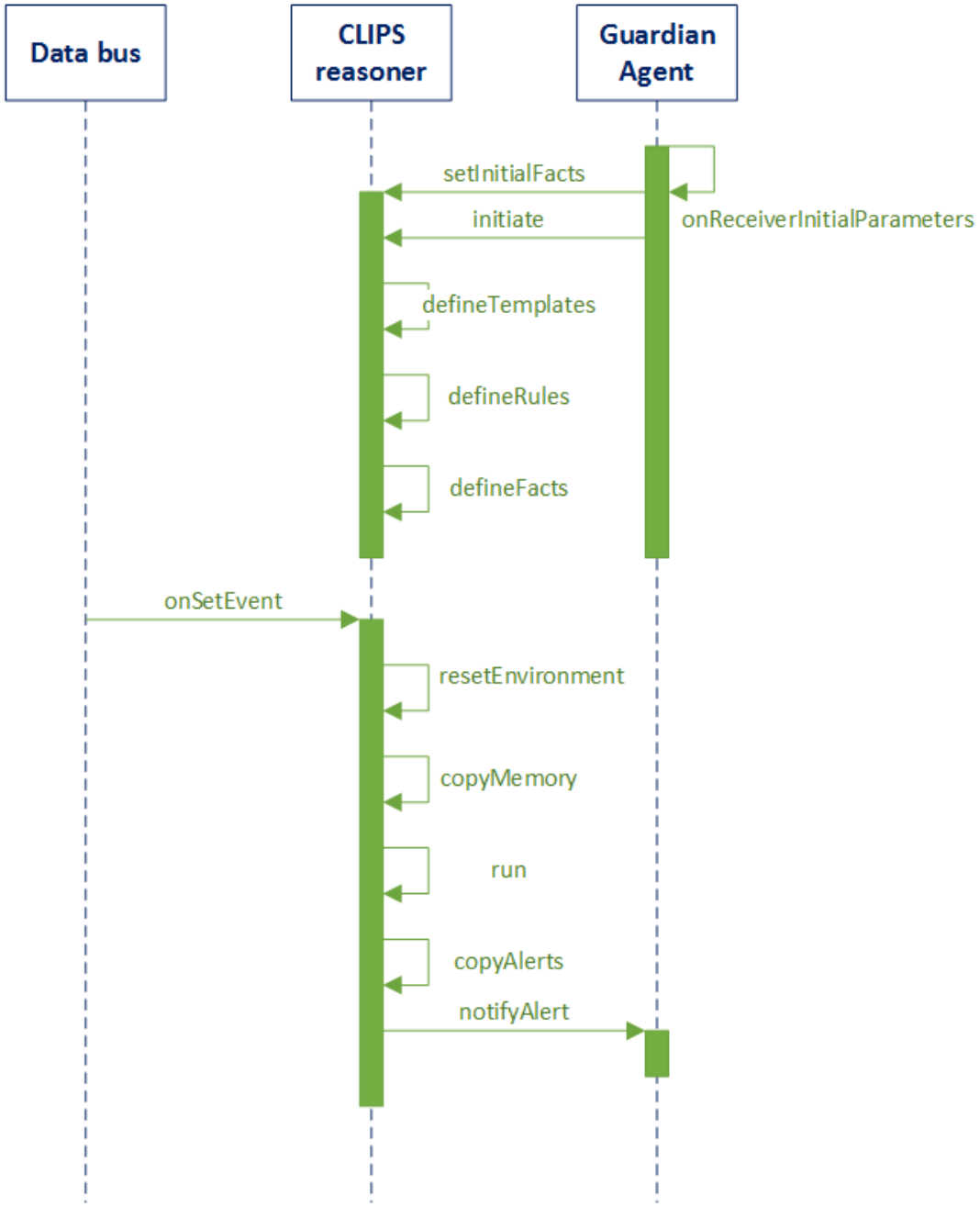

Figure 7.13: CLIPS conversion model

7.5. J2Unit test framework

The J2Unit test framework was invented during the implementation phase of the multi camera system. The name is derived from the well known JUnit framework and extends it with the management of the JADE agents. The point of the J2Unit framework was to create a test framework for JADE agents and use it to test the multi agent features. It is inspired by the framework proposed in [13] and adjusted to suit the needs in the multi camera surveillance system. The design is described in chapter 6, but here we show the implementa-

tion and problems we found during the implementation.

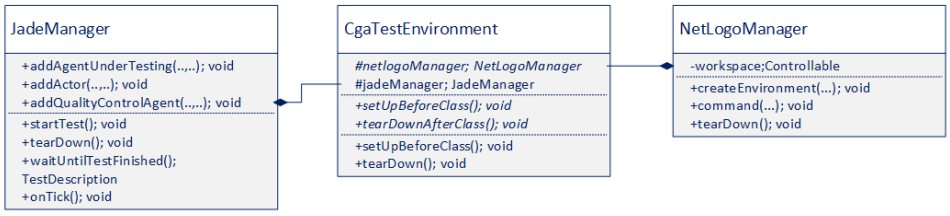

Figure 7.14: Bus listener as a service

Figure 7.14 shows the classes involved with the J2Unit framework. The CgaTestEnvironment is the JUnit test cases and every test case inherent the properties. The JADE and NetLogo environments are automatically created so they only contain the case specific properties.

7.5.1. JADE manager

The name of the JADE manager is fairly misleading, since technically it is the manager of the JADE agents. The closest representation of the real environment has a separate JADE instance running and our JADE manager would add the JADE agents on this platform. Particularly, the JADE instances would run on the agent operating platform. But an implementation with multiple JADE instances would be unnecessary complicated for our simulation and we only used one JADE instance. The JADE manager connects to this JADE instance and commands it to instantiate the agents.
The agents are managed by collecting the agents required for the test case and then start all the agents. The mock agents (which include the actors and the quality control agents) can be initiated using initiation parameters. After the creation of agents, the initiation parameters are passed through using the Agent to Object (A2O) interface provided by JADE. Since the mock agents are specifically designed for the test environment, we can use the dangerous method of sending it as common Java Objects and cast them within the agents. This would be bad practise for any other case, but since the mock agents only exist for test purposes, this does not create any problem.
Running the JADE agents unfortunately create some challenges in the JUnit test environment. Since the agents run as seperate threads, the junit test continuous the sequential code without considering the running agents. Once the test function is done, it simply stops the test and through the Java garbage collection, the agents also stop. In order to halt the junit test from finishing early, the JADE manager has a build-in feature to wait until the tests are finished. It also includes a fail safe to check if the test is taking too long to complete. But the halt feature waits until all the QCA reported the results of their tests. The test results are combined and returned as the final verdict of the test. By combining these features, the JADE manager wraps the JADE environment in the junit test framework, hence the name J2Unit test framework.

7.5.2. Mock agent structure

The mock agents are agents used in the test environment to control the states of the simulation or test the events in the AUT. There are two types of mock agents as shown in figure 7.15.
Normally, Java environment cannot control the JADE agents directly and only see the state of the agents through the controller. It is possible to use the A2O interface, but this is strongly advised not to use it, since it goes against the autonomy constraints of agents. But since the mock agents are create only for test purposes, the autonomous constraint does not apply. Applying these interfaces to the AUT must be avoided at all cost, that is why we decided to make services to listen remotely to the event logs of the AUT. We now have a framework that both meets the autonomous constraints and have enough insight to test the MAS using junit testing.
QCA often use the event log system to test the features of AUT. By subscribing the event log of the AUT, the QCA received all the events in the AUT. The QCA searches for the specific events it needs and checks the parameters. When testing the logical reasoners components, the QCA acted as Alert agents, which is essentially the same technique. Essentially, the QCA has a set of tests that have to succeed. All QCA call either the "TestSucceeded" function or "TestFailed" function and in both cases the QCA sets its state to done. This state is

detected by the JADE manager and the test result is collected as described earlier.

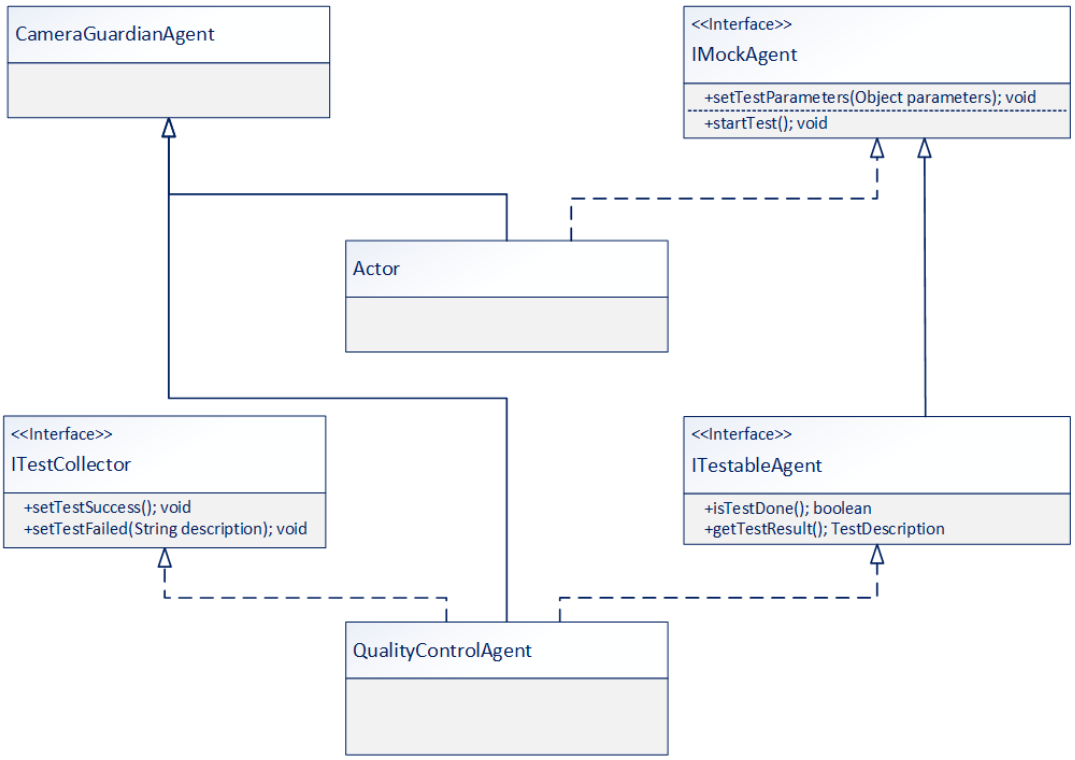

Figure 7.15: UML Class diagram mock agent design

7.5.3. NetLogo manager

After researching the topic, we found it was possible to start our simulation environment, written in NetLogo, from the test environment as well. However, the ease of which the NetLogo environment can be controlled from the test environment is far less than for instance the JADE environment. The NetLogo environment can start once per test, but is able to switch to difference scenarios by loading environment files. These commands are possible from the JUnit test environment, so instead of loading the NetLogo environment every time, we decided to load the NetLogo environment once and create different environment files for each test. Because every scenario has their own environment file, we decided to hard-code the simulated car location file reference in the environment file. Every test now has a NetLogo environment file, the car detection file (used by both NetLogo and the J2Unit test environment) and a test function in the JUnit Test Case. The NetLogo simulation now becomes an advantaged movie player that "plays" through the detections in the car detection file. We seperated the simulation step and the sending of the camera image step to increase the manageability for us. The "next-detection" command would update the simulation to the new car location and the "go" command would generate the images and send them to the individual cameras.

7.5.4. Environment state model

Combining the three components (JUnit, JADE and NetLogo) creates a sequence of events for each test. The steps in the J2Unit test framework are shown in figure 7.16. The steps have to ensure all the components and agents are ready at the same time and can perform the test.
The J2Unit test environment contains the NetLogo simulation, JADE agents and JUnit test case that would all run simultaneously. The timing of these test cases become a serious issue, since the detections are expected only when the simulation shows the car at that location. The processes are synchronized by introducing cyclic behaviour to the components. The cyclic behaviour in the test environment is different from the JADE cyclic behaviour. In the JADE environment, JADE schedules the cyclic behaviours of the agents. The test environment uses the cyclic behaviour to synchronize the actions between the three environments. The three

environments implement a on-tick function. The function is used to move the simulation or perform the next step in the mock agents. Again, the AUT are not influenced by these actions, since the synchronization step cannot be controlled from the JADE manager. But the synchronization step allows the control of the scheduling from the test environment to ensure that the results from the AUT can be validated.

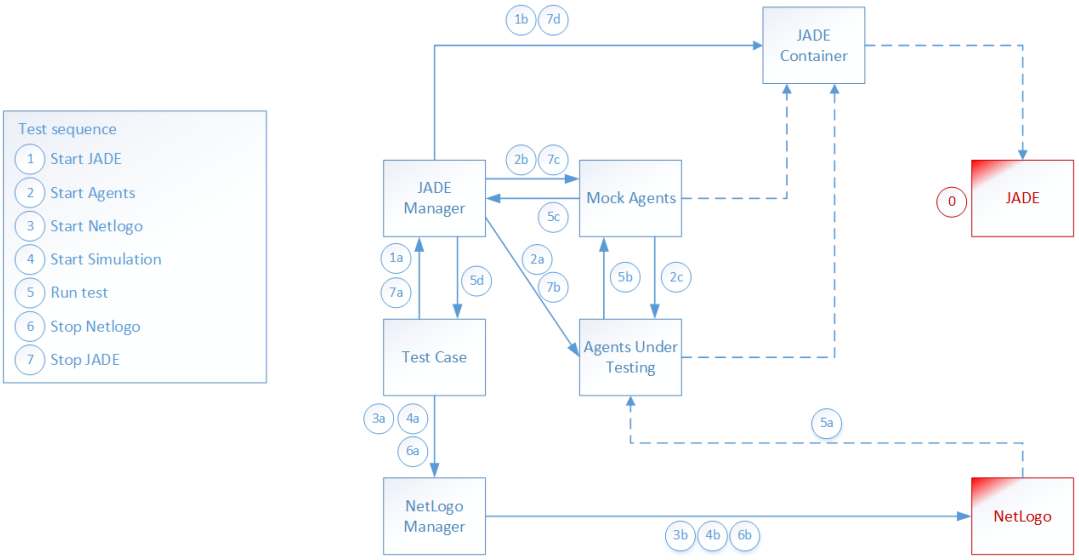

Figure 7.16: UML Class diagram mock agent design

7.6. Processing pipeline

Now that we have discussed the implementation of the supporting features, we can look at the processing pipeline and implementation of the specific processing blocks. To keep the blocks themselves simple and ordered, we sometimes broke up the DSP modules from the original design into multiple blocks. The track extraction has been replaced as a DSP module with properties of the track object or are calculated during the track building phase at the end of the recognition blocks. We will first discuss the distribution of the agent pipeline properties. Once established, we will go over each module one by one, explaining how they are implemented. Finally, we will discuss the reasoning applied by the CLIPS reasoning blocks.

7.6.1. Delegation and pipeline configuration

The cameras each see designated part of the environment. Although the cameras are implemented homogeneous, their location implicate different relative location, rules for approved car behaviour and regional agent they report to. These dynamic components that make each camera unique are combined in the camera parameters. The delegator agent is in charge of assigning each set of parameters to each camera, since the assigning determines the behaviour of the cameras. The initiation process for each camera consists of three phases: creation, collecting camera parameters and starting the pipeline. In the first phase, the agent is created. For this project, we created an extension of the regular JADE agent, which also starts the communication process. Once all the agents have completed the creation step, there is a stable communication possible between the agents. Since the agents are created in parallel and it never is certain when this process is finished, the agents are instructed to wait a while before going into phase two. Phase two collects the parameters. The parameters can be collected though a service running on the delegator agent. This is shown in figure 7.17. The IDelegator interface is given the service template, which means all the components needed for a service is also created. Finally, in the third step of the initiation, the pipeline components are initiated (based on the parameters) and started.

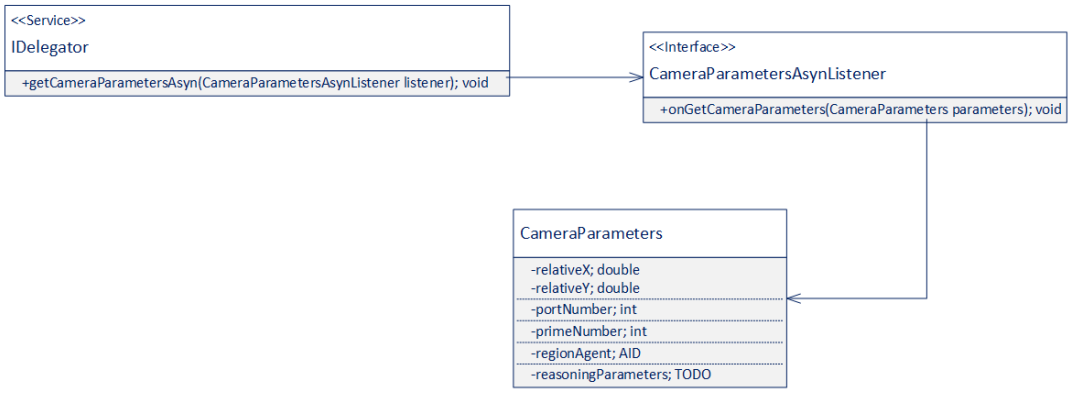

Figure 7.17: Delegator camera parameters UML class diagram

7.6.2. Detection and mapping

The detection and tracking components contain many decisions on the method of implementation. An overview of the processing pipeline is given in figure 7.18.

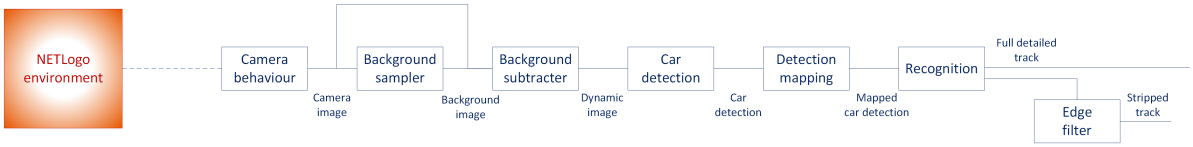

Figure 7.18: Detection pipeline components

The NetLogo environment is programmed to send matrices of images, representing the current state of the environment. For each camera a matrix is generated, encoded and send over a unique UDP socket. Any process listening on the port can pick up the messages. Each camera agent is provided a port number in the initiation process and a TickerBehaviour is started to periodically listen on the port. Once received, the message is decoded and converted into a specially designed DImage class. The timing of the camera images is unpredictable, due to the many processes that run simultaneously and the lack of guarantee of when the images will be send. A buffer is implemented to allow the pipeline to have a more stable sample frequency. The image receiving process stored the image in the buffer and a separate camera behaviour periodically samples the buffer to start the processing pipeline.
The first process to look into the images is the background sampler. The background sampler estimates the background by locating the static elements in the images. Taking the average over time will find the static elements in a picture and dynamic elements will be erased. An increasing method is applied to find the average of the images over time.

$$\mu_I[n] = \frac{n-1}{n}\mu_I[n-1] + \frac{1}{n}I[n] \tag{7.1}$$

where n is the last time index, $I[k]$ is the image at time instance k and $\mu_I[k]$ is the mean of the image at time instance k.
The next components on the pipeline is the background subtraction. The background found in the background sampler is subtracted from the original image. This straightforward process is complicated by a timing issue. The background subtraction requires both the original image and the background image. Since the background image is based on the original image, there is a slight delay before the background image is available. The process is unable to start without both images, however it receives an event from both of them individually. This problem is again solved by using a buffer. The event of the original image only stores the image in the buffer. Only on the event of the storage of the latest background image will start the subtraction process.

Car detection

The car is detected using a template matching. The matrix functions required for these calculations are implemented in an extended math class and include well-known operations, such as summations and cross multiplications. The process block calculates the most likely position for each orientation and then picks the most likely orientation. The configured threshold decides if the most likely is in fact a detection. This threshold is found empirically during the debugging process.
The detections are implemented using a specific data structure. The data structure is shown in figure 7.19. We seperated the Detection class and UidDetection class to be able to perform generic calculations on detections, such as distance, angle difference and similarity. The UidDetection is situated within an agent, containing an unique identification and origin. The origin is particulary interesting for the detections at the edge, since they may be replaced by detections on the stitches. The unique identification also helps us during the stitching processes to validate which detection is new and which was already received.

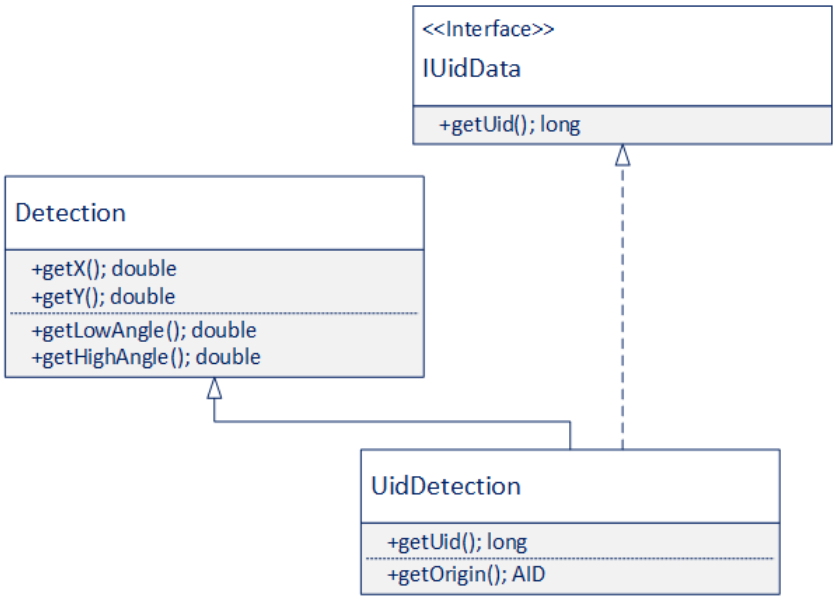

Figure 7.19: Detection data structure. Generic properties and situated detections are separated for processing purposes.

We found that the classification method was sufficient for the detections were the car was fully visible. The detections on the edge of the camera image were slightly more complicated, since multiple orientations would give the same likelyhood. After several attempts, the best method to detect these cars was to average the orientations that were likely enough. This gave us sufficient precision, but would have to be corrected by the higher layer agents.

Detection mapping

The implementation of the mapping is the simplest processing block of the collection. Once the camera offset is received from the delegator agent, the local agent is spatially aware and the mapping is nothing more than an addition of the image coordinates and the offset. This transforms the relative location to the global location.

Local recognition & meta-data extraction

The local recognition attempts to match the new detection with the already known tracks. The known tracks are gathered from the data logger with the use of the function "findElementsOfVariable(String)". For each track, the module creates a prediction model as shown in figure 7.20. The probability of a detection is calculated with the density function. The prediction model design allows us to create different prediction models in the future, but we found that the first order prediction model was sufficient for our purposes.

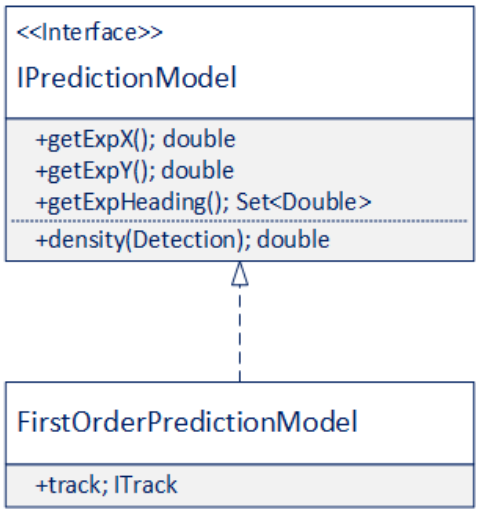

Figure 7.20: Track prediction model

The track data structure has changed the most during the implementation. This is not too much of a surprise, considering it is the data structure that is used the most over multiple components within the multi-camera surveillance system. The track component is used in almost all DSP modules, is communicated between the agents and is used in most of the CLIPS reasoning components. But the main reason the track data structure has changed so much during the implementation has been the regional recognition of the tracks. Before we get to the regional recognition, we will first discuss the final representation of the track, local recognition and meta-data extraction. Afterwards, we will look at the track distribution and regional recognition.
The final version of the track data structure is shown in UML class diagram of figure 7.21. The first thing we notice it that the track is of the type IUidData, which means every track will have a unique identification number and it will be created and updated with the set data logger mechanism. The ITrack interface start with numerous properties to trace the origin and reconstruct the track by its subtracks. For the local recognition, we will be using the leaf track, which means there are no subtracks and we will discuss this topic further with the regional recognition. Following the path down through the ITrack interface, the track shows properties that can be expected from a ordered collection of detections. All the known detections in the track can be collected and specific detections, such as the first and the last, have dedicated functions to collect them. We emphasize on the *known* aspect of the detections, because as the tracks are distributed to higher layer agents, specific detections are dropped to reduce the message size. Finally, we stumble upon the track properties such as sample size, duration and speed. Although the original design included the feature extraction, during the implementation we found that these features can be calculated as needed, as long as we stored the right detections. The speed and acceleration is calculated the moment is requested, instead of the result of the feature extraction module. The sample size, distance and duration are also calculated immediately when the recognition has taken place and the replacement track is being build. We found that this was the best time to perform these calculations, in contract to what we designed earlier.
During local recognition, we will always be using leaf tracks. Leaf tracks are not build out of subtracks from other agents and initially store all the detections that are added to the track. We developed a track builder to create the updated tracks, which we will discuss further with the regional recognition topic. Building a new leaf track simply adds one detection at a time, so the creation process is straightforward. The builder also ensures the sample size, duration and distances is adjusted accordingly. The local agents ends up with the track in full detail with the corresponding features, without the use of an explicite feature extraction processing block.

Regional recognition & track distribution

Regional recognition proved more complicated, even though is follows the same principle. After trying different policies for distributing the tracks, we decided to distribute the tracks, every time the last detection of the track was on the edge of the agent's scope. This had the following effects on states of the regional recognition processing block:

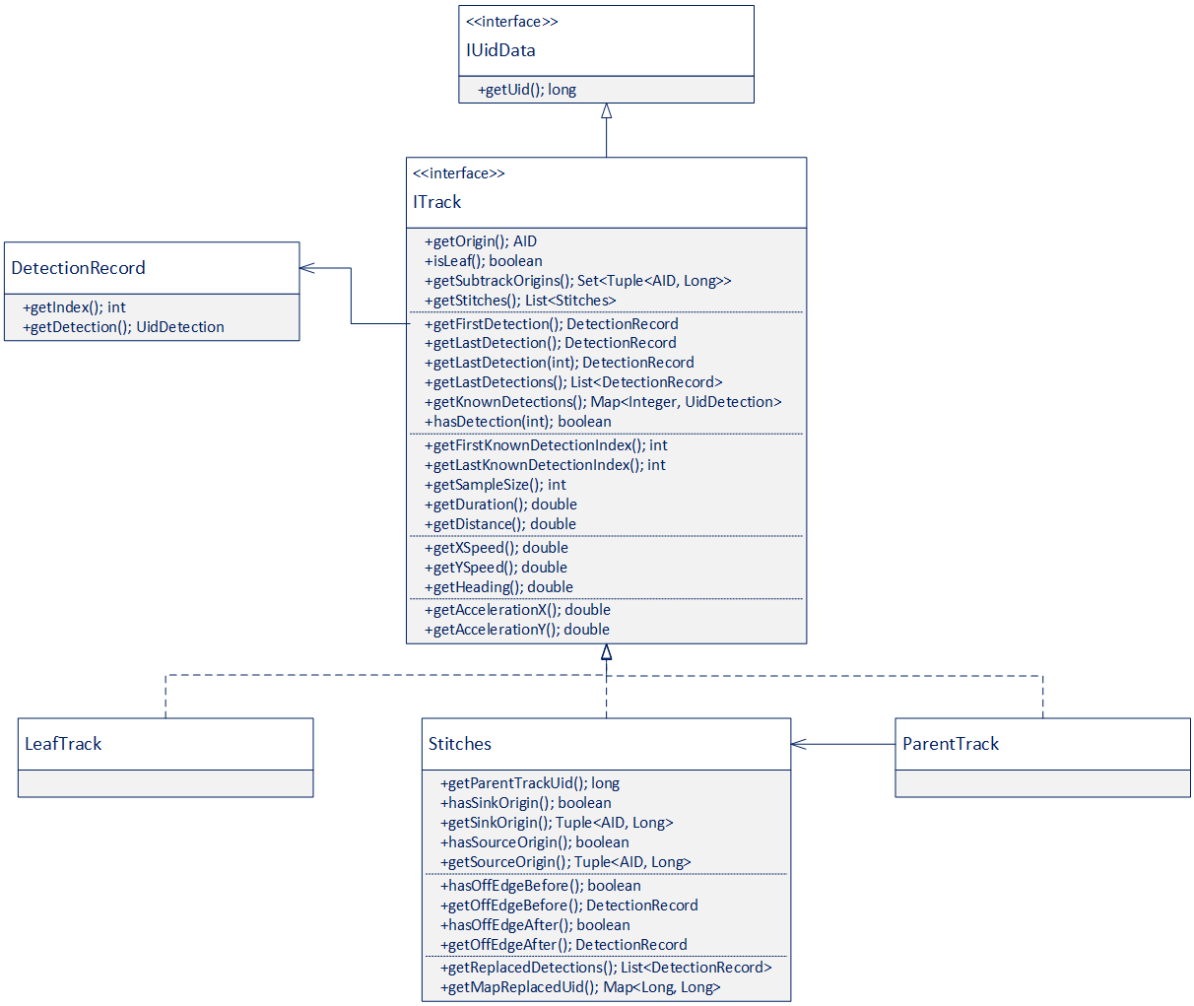

Figure 7.21: Track data structure with leaf and parent tracks. Stitches are also stored as tracks

- The first detection would immediately be distributed to all layers
- All detections on the edge of the lower layer agents and not on the edge of the higher layer agent would has two agents sending its subtrack. One from the sink track and one from the source track.
- All detections on the edge of both the lower layer agent and higher layer agents would have only one subtrack (the sinktrack). After processing on the higher layer agent, the updated track would in turn be propagated upward, because it is the edge of the agent's scope.
- Parking on the edge would mean the agents will send the track for every new detection. An advantage would be that the subtrack is already recognized as the source track and takes away the need for predicting over and over again.

The found effects on the states of the regional recognition processing block allowed us to analyse the states of the process with the LTSA. On the one hand, we used the LTSA to model the states of the regional recognizer, which we already saw in chapter 6, figure 6.22. We also used the LTSA as a tool to create the parent track builder. The state machine given in figure 7.25 shows the possible options during the building of the track. Based on this state machine, we created a track builder that would guarantee the track would be created as it should. The UML class diagram of this track builder is given in figure 7.24. The goal of the builder is to end up with a consistent track. Each state has specific actions that can or must be done to reach the consistent track. If an action is not possible in a stage, it will throw an exception. If an action is possible, the state of the builder often changes, so after the action is performed, the new state is returned. The ParentTrackBuilder acts as the central builder. It updates the pointer to the active builder (state) after each action.

The TrackModel is used to store the data so far and will be used to create the track, once the build action is called.
Parent tracks contain subtracks, which together from the full track. These subtracks are stored in stitches. The stitches contain all the information of the original tracks and the original detections on the edges. These original detections often have a some error in them, because the car was only partly visible in the image. If we recall the method of reconstructing the detection on the edge (figure 6.18), we find that we also need the last detection before the detection(s) on the edge and the first detection after, to be able to correct the travelled distance. This is why all of these detections are stored in the stitches. On top of that, the stitches also contain the origin and subtrack identification of the sink and source track. This effectively contains the reference needed for the track reconstruction.
Most of the techniques discussed so far, all keep as much of the known detections as possible. However, we want to minimize the data communicated between the agents. The minimization step is performed by stripping the track of the obsolete detections before the track is send. When stripping the track of the unnecessary detections, we need to know which edge is the outer edge of the current scope. The track always needs the first and last three detections. On top of that, we need the first detection after the edge and the last detection before the edge, in order to correct the distance. All the other detections (including the untouched stitches) can be dropped before the track is send to the other agent. The function stripTrack of the TrackBuilder performs this filtering. Ofcourse the sample size, distance and duration remain untouched. Every agent's pipeline includes an edge distribution processing block, which checks the last detection of new tracks to see if it is on the edge. If it is on the edge, the track is stripped of the obsolete detections and send to the superior agent. All of these components combined form the regional recognition process.

7.6.3. Distributed database query

Now that we know how the agents store the track and subtrack reference, the distributed database query mechanism can be explained. When a track has to be reconstructed, the client uses the global parent track identification to collect all the detections. Starting at the global agent, the parent track of that specific identification is queried from the agent's database. This track contains subtrack information, both the origin (in terms of the agent) and the identification number. The global agent uses this information to query the tracks from the regional agents. These regional agents in turn has the subtracks in their database, but has missing detections. But these subtracks have the original subtracks origin and identification available and query these agents to collect the remaining information. This process is continued recursively, until the process has reached a leaf track. The leaf track is returned without stripping the track of any detections. Once received by the regional agent, the track can be reconstructed and in turn be send to the superior agent(s). Finally, the track is complete reconstructed in the global agent and send back to the client. This shows how the track can be reconstructed completely, while at the same time minimizing the communication between the agents.

7.7. Reasoning components

The final component left for the processing pipeline is the CLIPS reasoning processing blocks. We designed the reasoning blocks to create there own CLIPS environment, to keep them as simple as possible. Each CLIPS environments needs a few things, including:

- Fact templates (class definitions).
- Rule definitions.
- Initial facts
- Current facts
- (Optional) support functions

We designed a converter mechanism to convert our Java data objects to the CLIPS syntax and back. We will start by discussing how we exchange the data between Java and CLIPS. Then follows the support functions used in some of the CLIPS reasoning environments. Finally, we will discuss the reasoning of the CLIPS environments themselves.

7.7.1. CLIPS conversion

The facts that are supported by CLIPS are Alarm, Detection, GridPoint and Track. The facts in CLIPS will have a subset of the properties on the Java variation, based on the needs in the reasoning. The data structures of the detection and track have already been discussed earlier. Listing 7.1 and 7.2 show the CLIPS template definition of these two classes.

Listing 7.1: Detection template

```
(deftemplate Detection (slot x (type FLOAT))
                         (slot y (type FLOAT))
                         (slot lowAngle (type FLOAT))
                         (slot highAngle (type FLOAT))
)
```

Listing 7.2: Track template

```
(deftemplate Track      (slot xSpeed (type FLOAT))
                        (slot ySpeed (type FLOAT))
                        (slot heading (type FLOAT) (default -1.0))
                        (slot duration (type FLOAT))
                        (slot distance (type FLOAT))
                        (slot sampleSize (type INTEGER))
                        (slot xStart (type FLOAT))
                        (slot yStart (type FLOAT))
                        (slot lowAngleStart (type FLOAT))
                        (slot highAngleStart (type FLOAT))
                        (slot xLastM1 (type FLOAT))
                        (slot yLastM1 (type FLOAT))
                        (slot lowAngleLastM1 (type FLOAT))
                        (slot highAngleLastM1 (type FLOAT))
                        (slot xLast (type FLOAT))
                        (slot yLast (type FLOAT))
                        (slot lowAngleLast (type FLOAT))
                        (slot highAngleLast (type FLOAT))
)
```

Alarm and gridpoint are data structures that are only used by the reasoning process and Alert agent. The alarm is used to indicate where the suspicious or unwanted behaviour is detected. The template of the alarm data is given in listing 7.3.

Listing 7.3: Alarm template

```
(deftemplate Alarm (slot centerX (type FLOAT))
                        (slot centerY (type FLOAT))
)
```

The gridpoint is used in the configuration of the reasoner. Every reasoning is situated in a different agent and has different points where specific behaviour is not allowed. The gridpoints are distributed by the delegator agent as part of the initial parameters. The represent where specific behaviour is not allowed and, in some cases, what heading is not allowed. The template is given in listing 7.4.

Listing 7.4: GridPoint template

```
(deftemplate GridPoint (slot centerX (type FLOAT))
                         (slot centerY (type FLOAT))
                         (slot width (type FLOAT)(default 1.0))
                         (slot height (type FLOAT) (default 1.0))
                         (slot heading (type FLOAT) (default -1.0))
)
```

The templates are collected as needed at the initiation of the reasoning blocks and defined before the first reasoning cycle. Every CLIPS environment has these templates available once the reasoning begins.

7.7.2. CLIPS support function

Next to the facts and rules, there are some commonly used calculations developed in CLIPS. We will quickly discuss the two developed for this project.

Listing 7.5: Calculate distance

```
(deffunction distance (?x1 ?y1 ?x2 ?y2)
        (bind ?dx (- ?x1 ?x2))
        (bind ?dy (- ?y1 ?y2))
        (sqrt (+ (* ?dx ?dx) (* ?dy ?dy)))
)
```

The distance between two points is a feature that was used multiple times during the reasoning. Of course we use Pythagorean theorem to calculate this distance. The implementation in CLIPS is given in listing 7.5. Another property that is used in the reasoning is the different between angles. Be aware that when calculating with angles, the value is always represented between 0 and 2π. In the case the difference is negative, we simply add 2π. This procedure is given in listing 7.6.

Listing 7.6: Difference in angle

```
(deffunction angle_distance (?a ?b)
        (bind ?d (- ?a ?b))
        (bind ?tp (* 2 (pi)))
        (min (abs (- ?d ?tp)) (abs ?d) (abs (+ ?d ?tp)))
)
```

We now have all the required components to reason about the car behaviour. We finish the implementation chapter with the rules applied within the CLIPS reasoning processing blocks.

7.7.3. Illegal entry

Illegal entry means the car is at a position where it should not be. The unwanted behaviour can be detected with just the one detection and does not need the track information. The reasoning attempts to match the car detection with the configured illegal entry locations and raises an alarm when they match. The rule is given in listing 7.7.

Listing 7.7: Illegal entry alarm

```
(defrule illegal-entry
        (Detection (x ?dx) (y ?dy))
        (GridPoint (centerX ?cx) (centerY ?cy))
        (test (<= (distance ?cx ?cy ?dx ?dy) 1))
        (not (Alarm (centerX ?dx) (centerY ?dy)))
=>
        (assert (Alarm (centerX ?dx) (centerY ?dy)))
)
```

7.7.4. Unwanted parking

Unwanted parking is similar to the illegal entry reasoning, however the speed of the car is needed to check if the car has stopped. The speed is only available in the track and so the reasoning requires the track to detect the unwanted parking behaviour. If the car is at the unwanted parking location and the speed is low enough, the alarm is raised. The rule is given in listing 7.8.

Listing 7.8: Illegal parking

```
(defrule parking-alert
        (Track (xSpeed ?mxs0) (ySpeed ?mys0) (xLast ?mx0) (yLast ?my0))
        (GridPoint (centerX ?cx) (centerY ?cy))
        (test (<= (distance ?cx ?cy ?mx0 ?my0) 1))
        (test (<= ?mxs0 0.001))
        (test (<= ?mys0 0.001))
        (not (Alarm (centerX ?mx0) (centerY ?my0)))
```

```
=>
        (assert (Alarm (centerX ?mx0) (centerY ?my0)))
)
```

7.7.5. Restricted direction

For the restricted direction, we have to know for each location, what is allowed in terms of heading. The initial configuration includes the illegal heading and the CLIPS environment include the rules to validate if there is a track that is driving in the illegal direction. The rule is given in listing 7.9.

Listing 7.9: Illegal direction alarm

```
(defrule illegal-heading
        (Track (heading ?mh0) (xLastM1 ?mx1) (yLastM1 ?my1) )
        (GridPoint (centerX ?cx) (centerY ?cy) (heading ?gpa) )
        (test (<= (distance ?mx1 ?my1 ?cx ?cy) 1) )
        (test (<= (angle_distance ?mh0 ?gpa) (/ (pi) 4)) )
        (not (Alarm (centerX ?mx1) (centerY ?my1)) )
=>
        (assert (Alarm (centerX ?mx1) (centerY ?my1)) )
)
```

7.7.6. Track length

Finally, the wandering behaviour is detected with the use of the travelled distance. If the car has covered too much distance, the system detects the behaviour as being suspicious. The CLIPS reasoning uses a straightforward comparison, but the distance is only validated in the global agents. The travelled distance in the regional and local agents only represent the travelled distance within the agent's scope. The rule is given in listing 7.10. Since this is complex behaviour and the behaviour is not technically illegal, the system sends a warning to the security guard.

Listing 7.10: Aimless driving warning

```
(defrule track-length
(Track (distance ?md0) (xLastM1 ?mx1) (yLastM1 ?my1) )
(GridPoint (centerX ?cx) (centerY ?cy) (distance ?gpa) )
(test (<= (distance ?mx1 ?my1 ?cx ?cy) 1) )
(test (<= ?gpa ?md0)
(not (Alarm (centerX ?mx1) (centerY ?my1)) )
=>
(assert (Alarm (centerX ?mx1) (centerY ?my1)) )
)
```

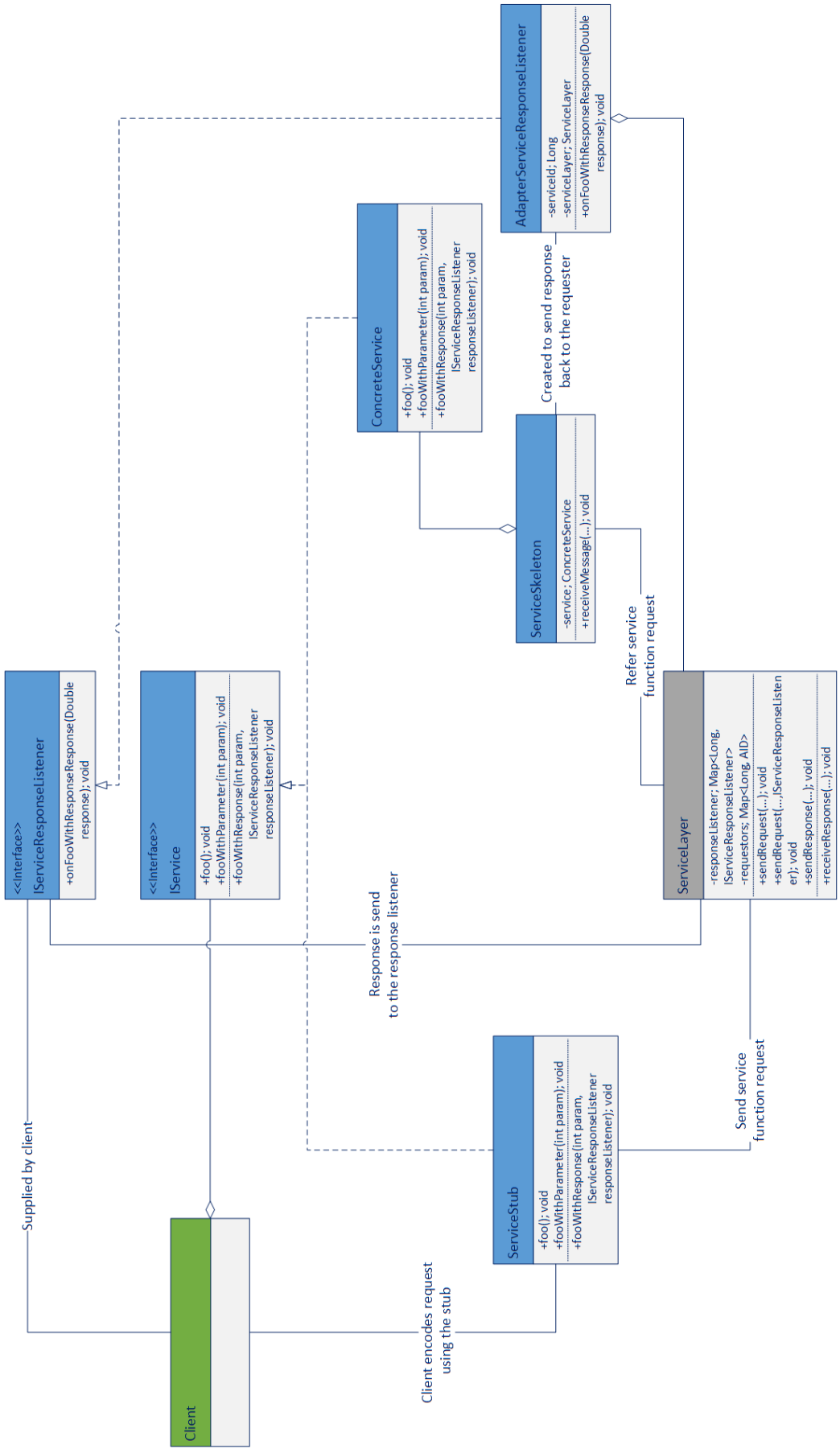

Figure 7.22: Classes remote function call with response

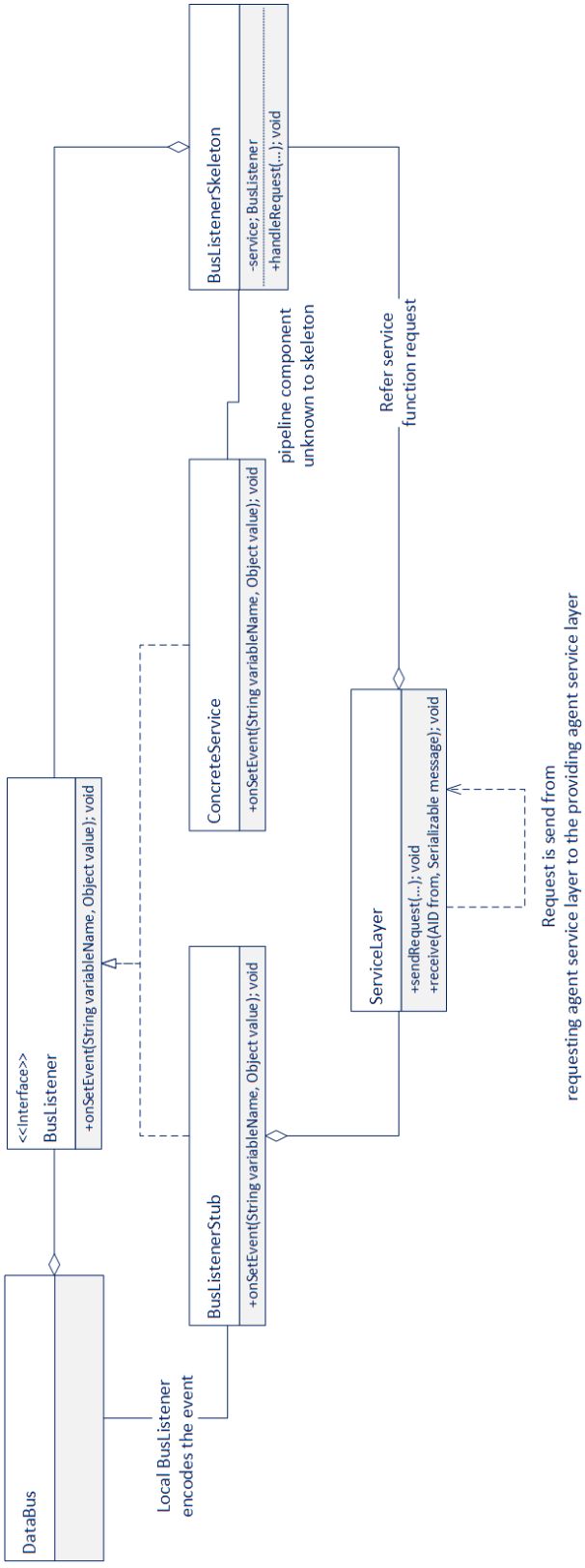

Figure 7.23: Bus listener as a service

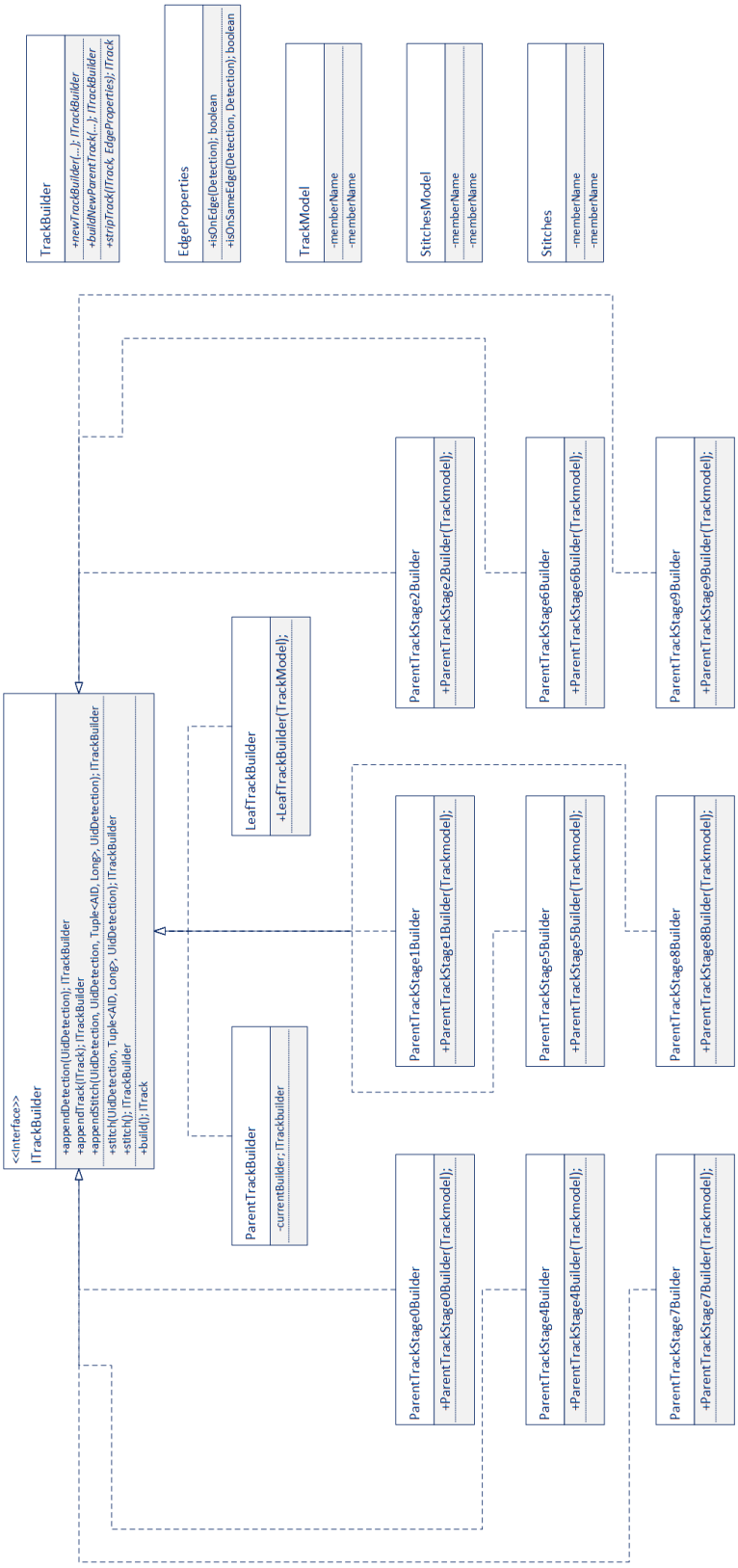

Figure 7.24: Track builder states data structure.

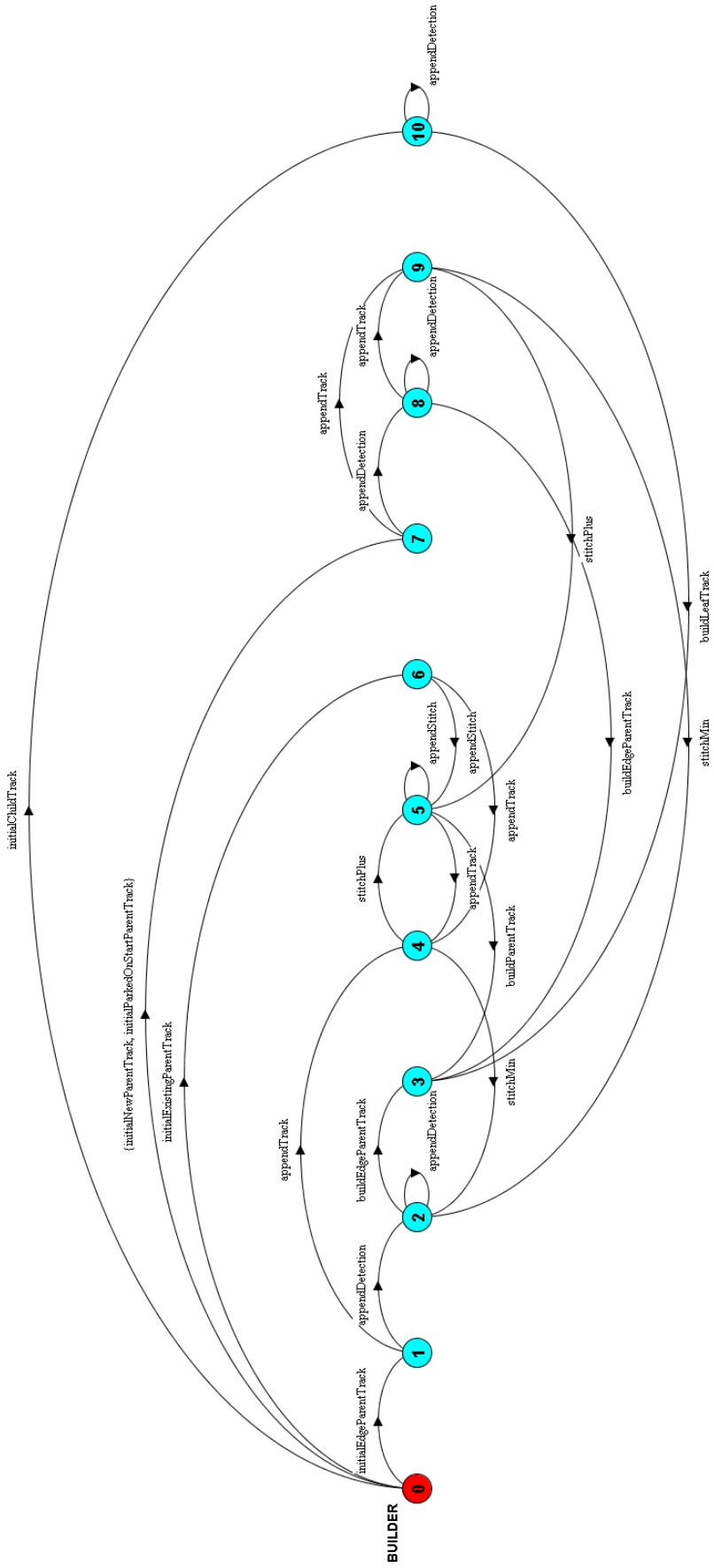

Figure 7.25: States during track building (analysed with the LTSA)

III

Tests & Conclusions

8

Experiments

The automatic suspicious behaviour detection system has been implemented and can be runned to detect particular unwanted behaviour. The performance of the system will validate with several tests. The tests include test both the individual features of the system and the reasoning capabilities. This chapter discusses the findings of the tests, based on the criteria discussed during the orientation.

Figure 8.1: Real-world equivalent of multi-camera surveillance

8.1. Guardian agent components tests

The guardian agent framework relies on a generic design for the agents, including components for communication, data storage, image processing and reasoning. During the orientation, we already saw that the autonomy properties of the JADE agents prohibited us from unit testing these components, while the were situated within the agents. We will shortly describe how we tested these components and what the results were.

8.1.1. Manual testing

Testing multi-agent system has been an important part of this thesis and as such, manual testing was considered a last resort. But as explained in section 2.4.2, testing software agents on the implementation level was not possible without voilation of the autonomy property. We attempted to build the generic components independently of the software agent. However, large portion of the components dependent on the event-driven system within the guardian agent. Particulary, the JADE behaviour implementation was hard to replace within a test environment. Replacing the JADE behaviour with custom test processes would practically mean creating a whole new agent platform and would create too much overhead. We ended up manually testing most of the components, including data storage, processing blocks and event log. Results of the tests would be displayed in the console and log files.

8.1.2. Invasive tests

In one particular case, we were able to automate the testing process by ignoring the autonomy property of the software agent. Testing service communication would create a bootstrap problem. The design test framework is dependent on the service communication to send the events from the Agent Under Testing (AUT) to the Quality Control Agent (QCA). Although the calling of a remote method would be possible, the communication of the success event relied on the correct implementation of the very same component. Ironically the more complicated feature of calling methods with return value could be tested non-invasively, since the QCA would be the agent receiving the response message.
The service communication protocol, present in all agents is essential for the functioning of all the agents. It is technically tested every time two agents communicate, but also the reason why it is important to verify that the service is functioning properly.

Table 8.1: Test parameters service communication test

	w/out input parameters	w/ input parameters
w/out return value	void foo()	void fooWithInput(double param1)
w/ return value	int fooWithResponse()	int fooWithResponseAndInput(double param1)

We created four tests to check the functionality of the service communication. The tests increase in complexity and show relative to each other, were potential bugs are present. The simplest test checks the calling of a mock function without return value or input parameters.
We ran both the invasive and non-invasive tests to see if the service communication component was working properly. The tests showed the service communication worked in all the scenarios. This gave us the confidence to build the remainder of the components based on this communication. Ofcourse, every test including communication between two agents would test the service communication again, but without these tests the potential bugs would be much harder to detect.
As predicted, we were not able to create unit tests without violating the autonomy property. This mend that the custom build test framework work with the much slower and much larger integration tests.

8.2. Integration tests

After all the components were manually tested, we could turn to the integration tests. The integration tests were knowingly sorted in dependency. Features that would present itself early in the processing pipeline were implemented and tested early in the process. This allowed us to iteratively build the system, knowing that the

dependent component would work on reliable data.

8.2.1. Car detection

The ability to detect cars was tested by generating random locations for the car to appear and QCA checked to see if the intelligent cameras were able to detect them at the right location. We tested the detection of cameras within one camera, throughout multiple cameras and on the edge of cameras. The locations were generated using the Matlab script, specifically created for this purpose. The sample size and scope were set for each test as given in table 8.2.

Table 8.2: Test parameters car detection test

	Single camera	Multi camera
Within camera view	10 random locations in single camera view	63 random locations in different camera views
Edge of camera view	10 random locations on the edge of single camer view	63 random locations on edges of different camera views

We conducted four tests on the subject of car detections, combining the amount of cameras and the location constrains. An event labelled "NEW_DETECTION" was created in the event logger to notify the QCA of the detection in the camera. The detections within the camera were expected to detect the exact location of the cars; the detections on the edge were expected to have a small amount of error in them, due to the missing part of the car that was out of sight of the camera. Particular for the test with multiple cameras and edge locations, the QCA had to be configured to anticipate multiple cameras detecting different parts of the same car. This created timing issues, due to the uncertainty which camera would report the detection first. It was sorted by building a sight tolerance for the order in which the test locations were validated.
The car detection tests showed the system was able to detect the cars in the location. The system was able to find the centre point of the car and the heading. As expected, the system was unable to find the front and the back of the car from one detection, so the heading always consist of two angles: the actual heading and the reversed heading. Since it is unknown, which is the actual heading, the system stores these headings as "low angle" and "high angle", referring to the lowest angle and the highest angle. The difference between these two angles is always π.

8.2.2. Detection mapping

Mapping the detections from the local image location to the global map is the first feature that required the agents to communicate with each other and use the shared knowledge in their reasoning. The tests performed for this feature where very similar to the tests on detection as seen in the test parameters of table 8.3.

Table 8.3: Test parameters car mapping test

	Single camera	Multi camera
Within camera view	10 random locations in single camera view	63 random locations in different camera views
Edge of camera view	10 random locations on the edge of single camer view	63 random locations on edges of different camera views

A new event labelled "NEW_MAPPING" was picked to broadcast to the QCA. The same margin of error in the detection on camera view edge was anticipated here, but due to the global location, the results are more precise. The test validation in the car detection tests checked if the locations were equal in the modulo, where as here, the tests where on the exact location.
The mapping of the detection from the local position to the global position shows the systems capability to collect the relative offset from the delegator agent and make the appropriate transformation. It represents the task for real cameras to transform the 2D images to a coordination system that is shared within all cameras.

Once the parameter initiation was working, the mapping tests succeeded effortless.

8.2.3. Local recognition

Local recognition is the ability to re-detect a car in a new image. No communication between cameras was necessary for this test, but in staid it needed cars to drive around within the view of one camera. We devised a Matlab script to generate random walker tracks. The script ensured the track would stay within the view of the camera and the path was a minimum of two detections. In the style of the previous tests, we first tested the tracks within one camera and later tested the tracks to appear in any camera view. This is shown in table 8.4.

Table 8.4: Test parameters local recognition test

	Single camera	Multi camera
Within camera view	3 random walker tracks in single camera view	10 random walker tracks within different camera views

The QCA know the detections in full detail, using the FullTrack representation of the track. The local camera agents used to event "EDIT_TRACK" on every change in the track. As a reminder for the next test, this is more often than the tracks are communicated between the agents, since these are only send when the car reaches the edge of the camera view. The QCA checked every state of the track with the test tracks given by the Matlab script.
Car driving around within one camera shows it is able to connect multiple detections. When the start position of the following track was too close to the previous track, the detection was often recognized as the previous car. This showed the system is able to recognize a car, but needs a different technique for multiple cars. The second test with tracks appearing in different cameras was no bigger challenge than the single camera test and gave the same results.

8.2.4. Regional recognition

The regional tracking proofed a bigger challenge than originally anticipated. The chronological order of the received track segments became critical to the systems ability to stitch them into one track. The tests showed the cameras were capable of combining the detections.
A comment should be made on the chance model used for combining the tracks. Although the system was able to chose between creating a new track and extending an existing track, the model is never thoroughly tested, since the simulation only contained one car. The results in that sense are not bad, but would require additional research in the case that it would be applied in practise. But for the single car track it got a perfect score.

8.2.5. Meta-data extraction

The implementation of the track feature extraction already showed that the extraction within a process block was deprecated. The features were extracted on the fly, when the get functions are called. The effect of this new approach is that the testing is no longer required. The features are always calculated the same. No test was designed for this feature and the extraction is considered to be functional.

8.2.6. Recursive track recognition

The key functionality of the hierarchical model was the combining of (local) track segments to from the overall track. The hierarchical approach requires a stitching method to combine track segments to form one complete track. In the design, we mentioned that the reasoning method to combine the tracks would be similar for all the higher levels in the hierarchical model. The experiments have proven that the stitching is indeed capable of combining tracks on different levels using the same reasoning for each level. The final test showed the smart cameras were able to track all the way from entering on one side of the environment to exiting on the other side of the environment.

We performed two types of test. One test singled out a regional agents and used actors to act as if the local reasoning agents would track a car within view. This allowed us to validate the stitching process of the regional reasoning agent. It proofed the agent is able to track the car over multiple camera views, including exceptional cases where the car would park temporarily on the edge of the camera(s). In all cases, the track was recognized correct, stitched and stored. The second test applied the full multi-camera system with the 63 cameras, 63 local reasoning agents, 9 regional reasoning agents and the single global agent. A car would drive a long track with the goal to drive through multiple local and regional reasoning agents' scopes. All the detection and recognition processes would have to work in order to stitch the track segments to one track in the global reasoning agent. The test showed the system is in fact able to track a car over multiple regional agents, proofing the complete recognition system within the multi-camera surveillance system works as intented.

Increasing delay issues

Even though the multi-camera surveillance system was able to track the car over the complete environment, the recursive track recognition design has a delay issue. Because the regional recognition requires one or two subtracts in order to combine them into one full track. Since the local and regional reasoning agents run as separate parallel processes, the order in which the track segments are received varies. In fact, when the car is located on the edge of the region, only one track segment is received. These properties force the process to wait for period of time, until it is certain all the track segments are received. (It also is the cause for the many possible states, but this has been discussed often in the design.) Particular the track segments that will be communicated to the next level will have the full delay. They will always be on the edge of a region and the last detection will always be in only one track segment within the region. This delay propates to the next level again and again. Since we are using one second delays, the highest level will have a build up delay equal to the number of layers minus 1. The good news is that it does not affect the regional recognition. All the track segments are delayed with the same amount of time. The recognition process is not aware of the delay build up by the previous layers and reasons as if it is happening right now. The bad news is that it affects the scalability of the process. When the system contains many cameras with many layers, the build up delay between the cameras and the global reasoning agent will be significant and may cause synchronization issues when the track is reconstructed.

8.3. Reasoning testing

The reasoning tests applied a different approach to the relation between the test agents and the agents under testing. The feature tests were designed to test specific features, which did not need the full system to work in order to be tested. The reasoning tests on the other hand focuses on the fully functioning system with all the agents working together to solve the classification problem. For the reasoning tests, we would start the complete environment, including the local, regional and global agents. The events we wanted to look for, were all present in the alert collection agent. Every time the behaviour is classified as suspicious or unwanted, the alert collection agent is notified through the alert service. We figured we may as well replace the existing alert agent with the QCA during the tests. All of the following reasoning tests are performed with all the agents in the design present, except for the alert agent, which is replaced by the QCA for the test.

8.3.1. Classify entry behaviour

The setup for the entry behaviour was to drive into a restricted area and validate if an alarm would be raised on the location. The simulation has several locations where the cars are able, but not allowed to drive. The best location would be the courtyard, used for gatherings and ceremonies. It is accessible by car and could be a shortcut to the sleeping quarters. In exceptional cases, cars are allowed to load and unload resources, but generally it is not allowed to enter the region. A perfect scenario for our test case. We manually created a track and stored it in a track file. The simulation ran the scenario and the QCA listened for any alerts when the car drove into the courtyard.
The test showed the camera was able to detect the illegal entry of the courtyard and raised the alarm on the first detection.

8.3.2. Classify parking behaviour

Our focus for the parking behaviour tests soon turned to the region around the armory. Parking in this area is the most threatening scenario in the case of the Den Helder training facility. Parking near the armory may indicate people attempting to steal weapons or ammunition. We manually created a track and stored it in a track file. The simulation ran the scenario and the QCA listened for any alerts when the car parked near the armory (in the restricted regions).
The test showed the camera was able to detect the illegal parking around the armory and raised the alarm on the first detection.

8.3.3. Classify heading behaviour

The original case of the Den Helder training facility had one case where the cars are only allowed to drive in one direction. The abstraction of the environment for the simulation moved the one way lane slightly to ease the simulation. But the essence is still the same and drivers can still be temped to drive against the driving direction as a short cut to get off the premisses faster. Again, we produced a track file with this hypothetical case and see if the MCS was able to detect the heading.
We soon found out the location was exactly on the edge of two cameras and two regions, meaning the detection would have to be performed on the global level. We decided to alter the simulation and increase the length of the one way street. This way the classification of the heading behaviour would be possible on both local and global level. The local level succeeded in detecting the heading violation. However, later test revealed the global reasoning agent is also able to detect the violation. The tests proofed the multi camera surveillance system is able to detect heading behaviour, including the exceptional cases.

8.3.4. Classify wandering behaviour

The wandering behaviour is detected by the travelled distance of the car. This is considered complex behaviour, since it required information about the complete track and will be detected in the global reasoning agent.
The test proofed the system is able to detect tracks with exceptionally long distance and send a warning.

8.4. Model evaluation

During the orientation phase, we developed a number of criteria on which to compare surveillance systems. We have evaluated the current work, based on these criteria. These are the findings.

Scalability

The proposed model is more scalable than the previous systems. The recursive design of the regional agents allows the model to be scaled with the addition of new agents or new layers. The bottleneck of the global agent is reduced with the frequency and data size of the packages. The limitation of the scalability is primarily based on the message delay between each layer, but will be significantly more than the previous models.The proof-of-concept contains 63 cameras (with local agents), 9 regional agents and 1 global agent. In total the system ran 73 reasoning agents, without performance or memory problems.

Modularity

The proposed model is very modular. The combination of common programming languages, virtualization of the agents and adjustment to the existing system allows the system to be easily installed as an extension of existing systems. Common cameras can quickly be converted into smart cameras and agents can be assigned to any smart camera, allowing equal distribution of processing demand.

Multimodal fusion

In terms of combining information from different sensory types, the current model has no particular mechanism to handle it. The generic agent components or pipeline allows future projects to quickly include this feature, since it will only require new processing blocks to be developed. The current model does combine track information from different cameras.

Centralised versus decentralised
The hierarchical approach combines the best of both worlds. All the agents can run in parallel and create a processing stream. The global agents is still a potential bottleneck, but is reduced significantly from the centralized approaches.

Vigilance
The current system is has real-time tracking and detection of unwanted behaviour. Most systems perform tracking of past information and are not able to detection complex behaviour in real-time.

Human model
The automatic suspicious behaviour detection is designed using the guardian angel approach. The agents will only assist the user. This can only be done if the reasoning model is based on the human reasoning model and in fact partly mimics this reasoning behaviour.

Implementation on current camera networks
The current system can be applied to any IP-based camera surveillance system and only requires small investment and time.

Local/global detection
The main focus of the model was to reason about both local and global behaviour. In the tests, we saw that the system is capable of both detection local unwanted behaviour and global unwanted behaviour.

Reasoning technologies
Only rule-based reasoning was applied in the current model. Some statistic methods were used to combine detection into tracks and track segments into larger tracks. However, the pipeline is ready for future implementation of more complex reasoning models. The modular approach increases the manageability and reduces the development time of the system. But the current implementation mostly relies on rule-based reasoning to detect the unwanted behaviour.

Experiments
The simulated environment and custom agent test framework allowed us to reduce the duration of the tests (as compared to real-life testing). This allowed us to test more scenarios and perform more thorough tests.

The findings are summarized in table 8.5, using the same notation as the evaluation of the previous models.

Table 8.5: Summarizing table

Criteria / Projects	Scalability	Modularity	Multi-modal fusion	(de-)Centralised	Vigilance	Human model	Implementation	Local/global	Reasoning	Experiments
Guardian agent model	+	++	+	dec	++	++	++	global	+/-	++

9

Conclusions

We started the thesis with a vision: "We envisioned cooperative intelligent cameras, working together to assist the guard to survey the area. Imagen a building where the cameras communicate with each other and the security guard is only alerted when countermeasures are needed. Intelligent cameras could get rid of the need for centralized servers and proof to be more scalable than traditional systems." At the end of the thesis we want to gather our thoughts and see how much of the vision has been realised throughout the work. In the conclusions we want to summarize the work and see how far we have gotten in achieving our vision.

Figure 9.1

9.1. Summary

In “Automatic Suspicious Behaviour Detection, a distributed approach to multi camera car surveillance” we research how to create a decision support system for multi-camera surveillance systems, using cooperative intelligent cameras. Commissioned by the Royal Dutch Navy, we set out to design and implement a distributed reasoning model for analysing car behaviour within the officer trainings facility.

9.1.1. Orientation

During the orientation phase, we found that visual surveillance systems have evolved from single low resolution analogue cameras to networks for digital high resolution intelligent cameras with automatic event detections and sophisticated database models. The focus of designing visual surveillance systems has shifted over time from improving single camera images to the design of data distribution and processing load balance. Once the camera images where digitalized, it became possible to automatically analyse the images and detect objects within them. As the capacity of the intelligent cameras increased, they became able to analyse behaviour and automatically detection abnormalities. But the processing power demand of such systems were rarely scalable and stayed within fixed sizes of the system.
The reasoning model designed for the multi-camera surveillance system follows the guardian angel concept, which means the goal was to assist the human operator, rather than replacing him. In the context description, we used the cognitive models of the human mind to design the automatic reasoning model. By including the human reasoning model, we were able to design the automatic reasoning model such that it would both optimally assist the human operator and make a distributed system inspired by human reasoning. Using Rasmusses levels of cognitive reasoning, we adjusted the design such that automatic hierarchical model would process simple instinctive information at the local levels and perform complex analysis at the global level. The human operator would be assisted by receiving alarms on small local incidence, while at the same time, presenting the car’s track at real time. Using this cooperation between man and machine, the intelligent cameras would stay vigilant for the human operator when he may have trouble doing so and help the operator to quickly gain insight in the car behaviour when usually this would require a heavy mental workload, keeping track of all the cars.
Further orientation in the available software & libraries, we decided to separate the project into fout aspects: behaviour analysis, world simulation, the designed system and the functional tests. Since the research is part of a general research in automatic behaviour analysis, this research only focusses on the distributed reasoning of the automatic visual surveillance and does not discuss the practical problems associated with real-life camera environments. The proof-of-concept is build on top of a simulated environment, which is simplified version of the actual case scenario. The advantage of the simulation it that it allows us to study the cooperation between the intelligent cameras. The disadvantage is that the simulated environment does not provide any insight in the typical behaviour. The expert knowledge about the typical behaviour in the environment is programmed into the simulation, rather than that the behaviour naturally occurred. Based on the expert knowledge, we devised certain heuristics about the car behaviour and ran simulations in order to adjust the parameters. Because the proof-of-concept is a balance between the scientific research and the commissioned case scenario, we applied unit testing to show exactly what the final system is capable of. Although unit testing is a mature developers method for sequential processes, unit testing of multi-agent systems is still being developed. Based on the mock agents testing framework proposed in [13], we decided to create an unit testing framework for the multi-camera surveillance system.
The experimental design closes the orientation phase of the thesis by defining the use cases and test method of the multi-camera surveillance system. The multi-camera surveillance system will support five main use cases: classify entry behaviour, classify parking behaviour, classify heading behaviour and classify complex behaviour. The use functionalities will only be possible of the supporting features are fully functional. The supporting features follow directly from the orientation in the related work and include: agent service communication, background separation, car detection, world mapping, local car recognition, track information extraction and recursive car recognition. All of these features would also be present in the implementation of visual surveillance systems in real environments, however may be easier in the simulated environment. The tests will show the performance of each feature and functionality.

9.1.2. Building the system

We started by building an event based pipeline structure acress the agents. A new generic agents was introduced (named the guardian agent) which unified communication, data processing, memory and event logging. This is realized by using the subscription method in combination with the service method of communication.
The hierarchical agent model creates a chain of agents, sharing a processing pipeline until the logical reasoning components. Local information processing detects, maps, recognizes cars and extracts behaviour information from the tracks. The behaviour information is used to reason about the events that occur locally. The regional information process reuses the extracted behaviour information to combine track segments into larger tracks. Here the car is recognized again over different track segments and the behaviour information from this combined track is extracted. This information is then used to reason about the regional events, which tent to be more complex, but less dependent on details. The process of reusing extracted information, combining track segments and extracting new information is recursively applied in the hierarchical model. Each step reasons about more complex tracks, while bottlenecks are avoided by reducing the resolution of the tracks.

9.1.3. Experimental results

Test results showed the cooperative reasoning model of intelligent cameras was capable of detecting different types of behaviour. The local agents were able to detection the cars in the images, map the locations to the world coordination structure, recognize the car over multiple images and extract information from the tracks. Local behaviour classification proved able to detect entry behaviour, parking behaviour and heading behaviour.
Also the agents were able to cooperate with success, the recognition process introduced a slight delay in the stitching process. The delay propagates through the system and each level will have to wait longer then the previous level. This stops the system from being truly scalable, since every level requires a definition of all the exceptional behaviour that could occur during the delay.

9.2. Addressed problems

At the beginning of the research, we set the scope for the project. We will discussed the problems individually and evaluated the findings for each problem. The main problem for the research was to design of a distributed model for reasoning about car behaviour, using multi-camera surveillance systems. The topic is divided over the following subproblems:

- Design of a model for automatic suspicious behaviour detection for multiple camera system as a decision support system for an human operator.
- Implementation of the model using JADE for multiple agent system and CLIPS for reasoning.
- Simulate the environment for the implemented scenario using NETLOGO.
- Application for specific environment of the defence academy at Den Helder.
- Design an automated agent test framework to validate the performance of the proof-of-concept in the different scenarios.
- Testing different scenarios of one car travelling over the military area using different routes and show different behaviour.

9.2.1. Distributed reasoning model

The first problem was to design the distributed reasoning model. The model would later be tested in the proof-of-concept system, but the intention of the reasoning model was to set a standard for reasoning over multiple cameras. Using the cognitive model of human reasoning, we were able to design an hierarchical model that was both inspired by and adjusted to assist the human operator. The model would simultaneously assist the human operator in keeping attention during tasks that required heavy mental and help the

operator stay vigilance when small detailed events occurred that would easily be missed. The model would support real-time tracking of cars to give quick insight in the complex behaviour and detect minor mishaps that would otherwise be lost in the sea of information. Following the guardian angel principe, the reasoning model helps the human operator to perform better, in contrast to attempting to replace the guard altogether. The reasoning model included a unique approach of separating many simple classification tasks and few complex classification tasks by solving the tasks in the scope on which they occurred. By distributing the tasks over multiple agents, the processing load balance was kept close together over all the intelligent cameras. It provides a new approach for designing scalable multi-camera surveillance sustems.

9.2.2. Implementation of the proof-of-concept

The next problem including building a proof-of-concept in a multi-agent system and show the reasoning model could be applied to solve multi-camera surveillance tasks. The implementation is done in a simulated environment. We found that is was possible for the multi-agent system to implement the simple classification tasks and reason about local behaviour. The functional tests showed the local agents were able to detection the cars in the images, map the locations to the world coordination structure, recognize the car over multiple images and extract information from the tracks. Local behaviour classification proved able to detect entry behaviour, parking behaviour and heading behaviour.
Collecting the track information from a distributed database was designed in theory, however were where unsuccessful in creating an automated test scenario for this case.

9.2.3. Simulation & real application

Using the expert's knowledge, we were able to create a simulated environment, including key scenarios of unwanted behaviour. We were able to design a test configuration which had agents receive images in a similar fasion as one would expect in the real scenarios.
The representation influences the conceptual model and reasoning on behaviour. The simulation allowed us to catch the essence of the reasoning, but the exact same reasoning could never be applied in real environments, since the raw data and conceptual model would be different and the behaviour would present itself different. The distributed reasoning model, like the cognitive model of the human mind can only serve as an inspiration for implementing the real multi-camera surveillance system. In terms of the pipeline, however the simulation represents the same information processing as was commonly found in the related work. Eventhough the calculations themselves will most likely be different in real environments, the hierarchical task delegation proved capable of performing the same reasoning steps as required in real environments.

9.2.4. Testing scenarios

The tests validated the developed multi-camera surveillance system was able to detect, recognize and reason about the behaviour of the car. The local reasoning agent was able to reason about the behaviour within the camera view and detect simple behaviour, such as unwanted entry or illegal parking. The regional agent was able to link the track segments into one track and use the information to reason about behaviour on the edge of the camera view, taking the first steps in the multi camera surveillance systems. Finally, the combined effort of all the agents were able to track the car over the complete environment. The complete track allowed the multi-camera surveillance system to reason about complex behaviour, such as wandering behaviour.

9.3. Conclusions

We have arrived at the conclusions of the thesis. The conclusions are drains based on the scientific challenges and research questions we composed at the start of the research.

9.3.1. Scientific challenges

We will start with an evaluation of the progress in the scientific challenges we defined for the thesis. We will go over each challenge and discuss how far we have met the challenge.

Probabilistic reasoning framework for detection and recognition of cars.

The proposed model showed how we could use the car behaviour in a probabilistic reasoning framework to detect and recognize the car from the raw images. The proof-of-concept in turn proved the model would work. However, only one car was used and the tests were performed in a simulated environment. Introducing more cars will most likely also introduce detection errors. The problem with the misclassification is that the remainder of the system is not flexible enough to correct the misclassifications. Any error in the detection process will propagate through the pipeline and increase. Even though the proof-of-concept was functioning and the probabilistic reasoning framework works, more research is needed to cope with the inevitable detection error.

Share information and data fusion.

The hierarchical reasoning model is based on cooperative agents, working together to detect suspicious behaviour of different complexity levels. It is fundamental for this model to share and combine track segments. The proof-of-concept was able to track a car over the complete environment, including the scope of multiple local and regional agents. This proves the system is able to share information and combine them to form new information.

Distributing tasks over multiple cameras.

We choose to apply a hierarchical reasoning model with simple behaviour reasoning at the lower levels and complex behaviour reasoning at the higher levels. During the orientation we found that the model has parallels to the method used during human reasoning, where simple behaviour is performed instinctively and complex behaviour uses cognitive reasoning. We decided to separate the tasks based on the amount of information needed to make a valid analysis. Simple behaviour needed one or a few detections and could be performed locally. But before each local reasoning agent would be able to reason about the behaviour, it had to become spatially aware and understand what behaviour is allowed per location. The delegator agent supplied this information. Complex behaviour analysis needs an overview of all the detections. This analysis is assigned to the global reasoning agents, which had all the required information. The cooperation between the local, regional and global agent with the assistance of the delegator agent allowed the tasks to be properly distributed across multiple cameras.

Design a scalable reasoning system.

The proposed hierarchical reasoning model is highly distributed and all components work in parallel. The result was a system that is far more scalable than the traditional centralized approach used in suspicious behaviour detection systems. However, the implementation of the model showed a delay in the distribution of the tracks. The delay is not problematic and won't be a problem until many layered are applied, but it is a known limitation.

Communicate findings with security guards.

During the orientation we found that the most valuable features was to track all the cars to be able to give a quick overview and warn or alarm the guard about unwanted behaviour. The goal was to assist the guard, not to replace his or her task with automated systems, which was performed best by keeping the guard vigilance and quickly presenting the track of each car. The proof-of-concept was able to track the car. Through the use of an alarm agent, all the suspicious and alarming behaviour that was detected by the reasoning agents are collected, enabling the system to warn and alarm the security guard.

Automate agent testing without violating the autonomy property of the software agent.

The custom build agent test framework was able to perform automated integration tests. It was not possible to test the software components within the agents without violating the autonomy property. These components were tested both manually and using invasive techniques. The test framework did proof able to perform the integration tests and allowed us to iteratively develop the processing pipeline. Although these tests were slower and had a larger scope, the custom test framework allowed us to reduce the development time for the processing components and create reliable code.

9.3.2. Research questions

Finally, we will evaluate the results based on the research questions. We will discuss them in order as given in the introduction.

Can a system detect cars from raw images and recognize the track within and between cameras?

The design of the system is based on a multi-camera surveillance system and although it focused on a simulated environment, it can be portable to real-life environments and the pipeline in the simulation forms the basis for the pipeline in real scenarios. The proof-of-concept started with two dimensional images containing structure and representations of the car. The system was able to separate the dynamic components from the image and detect the car within the image. Using the past behaviour of the tracks, the system was able to recognize the individual detections as part of a track. This proves a system is able to detects cars from raw images and recognize the track within and between cameras.

Is it possible to design a reasoning model in a hierachical way starting from multiple simple agents with limited view of the world at the bottom up to complex reasoning with global overview at the top?

The designed cooperative agent model includes separating the simple and complex behaviour reasoning. We found that the historical dependency directly influenced the level in which the reasoning would have to be performed. The further the reasoning dependent on the past samples, the higher the level of reasoning must be applied. This is directly related to the complexity of the reasoning.
The clustering of cameras is still a manual process. The reasoning currently only happens when the cars cross borders of regions, which may not always be enough and the designed model has timing issues, preventing it from having a quick reaction to new tracks. But what it showed is that it is possible to design a reasoning model in a hierarchical way with the separation of simple and complex behaviour.

Is it possible to design an automated surveillance system for multi camera surveillance system?

We consider the reasoning model capable of performing surveillance tasks automatically in a multi camera surveillance system. The simulation showed the reasoning model would be able to perform the same reasoning steps as were found in the real multi camera surveillance systems. The proof-of-concept implementation showed that the agent were able to detect cars, map to a world coordinate system, recognize the car over multiple images and extract information from the series of car detections. On a local level the agents were able to reason about unwanted behaviour and alarm the guard in the case of an unwanted event.

Is the developed model scalable?

In terms of processing load, the model is scalable. The model can be optimised by finding the processing load of each step and distribute the processing load as much as possible. This is dependent on the specific case and must be investigated for each new application. The processing delay introduced in the recognition step is the main limitation to the scalability. More research is required to solve te delay found in the communication of the track segments.

Is it possible to apply the developed model to real-life specific environments?

Using the expert knowledge, we were able to create a simulated environment, including key aspects of the unwanted behaviour in the rea-life environment. The model proved capable of reasoning about the environment. The simulated environment represented the objects in the environment differently than the real environment and the camera topology eased the recognition process. Yet, the model seemed capable of performing the same processing steps as is expected in the real environment and was able to detection local behaviour. Although the implementation of the processing blocks would have to be adjusted to fit the situation of the cameras, the method of reasoning would be applicable for real-life specific environments.

Is it possible to automatically test the software agent's reasoning capabilities without violating the autonomy property?

We proved there are definitely options for automatic software agent testing, without violating the autonomy property. However, there is still room for improvement. The test environment could be configured to be able to test generic components on a separate test bench. By applying dependency injection, the processing blocks could be tested individually, which would lose the dependency of the other agent's components. We decided to leave that, because of the dependency of the JADE behaviour mechanism. Replacing this with mocking objects would practically mean rebuilding the JADE agent principle. We leave the development of such a testbench to future work. We were able to automate the integration tests. By using an iterative experimental design, we were able to, in each step, build on reliable code.

9.3.3. Model comparison

We conclude this chapter by evaluating the model proposed in this thesis with the previous work of the research group. The comparison is made in table 9.1. We can see that the proposed model have made progress on almost all criteria for multi-camera surveillance systems.

Table 9.1: Summarizing table

Criteria / Projects	Scalability	Modularity	Multi-modal fusion	(de-)Centralised	Vigilance	Human model	Implementation	Local/global	Reasoning	Experiments
Single multi-modal camera	+/-	+/-	+	dec	+	+	+/-	local	+/-	+/-
Network of distributed cameras	+	+	+	dec	+	+	+	loc/glob	+	+
Network of distributed cameras of different type	+	+	-	dec - cen	+	+/-	+	loc/glob	+	++
A surveillance system of a military harbour using AIS	+/-	+/-	-	cen	+	+	+/-	global	+/-	+
Intelligent multi-camera video surveillance	+/-	+/-	-	cen	+/-	+/-	+/-	global	+/-	+
Guardian agent model	+	++	+	dec	++	++	++	global	+/-	++

10

Future work

After all the work that has been done, we like to take some time to look at future research. We asks ourselves, where do we go from here and how can we use the findings of this thesis in future projects. In this chapter, we summed up recommendations for future work. We divided the recommendations in two parts. The internal improvements discusses the actions that can be done to improve the automatic surveillance system, such that it would better fit the request of the Royal Dutch Navy. In the application scope, we ask the question what else can we do with the findings in this thesis?

10.1. Internal improvements

The internal improvements discusses the actions that can be done to improve the automatic surveillance system. The system is original request to design an decision support system for surveillance systems was intended for the specific scenario in Den Helder. But building the whole system is too much to build for one humble thesis project and contained too many scientific challenges. Throughout the thesis, we only focused on a selection of the scientific challenges, but we never let go of the idea that the result of the thesis would help in the design of the surveillance system for the Den Helder training facility. In the spirit of assisting the Royal Dutch Navy, we listed the key aspects that would be the best next step in the development of the automatic surveillance system.

10.1.1. Multiple cars

Currently, the automatic behaviour detection system is tested with only one car. Although the data model supports multiple cars, the detection of cars is expected to fail when multiple cars are in each others proximity. The problem with the template matching technique is that it is unable to detect the exact location and orientation of two neighbouring cars. The current approach would suffer detection errors, which would only increase with every step in the processing pipeline.
Related research in behaviour of ships near the harbor was able to track large numbers of ships and did not suffer this problem [68]. The advantage in this research over the multi-camera surveillance system is the Automatic Identification System (AIS) on board every vessel. Each vessel kept track of its own location, speed, heading and size. Information had the chance of not reaching the destination, but had no trouble with confusing ships within an image. Recognizing ships over different samples was easy, because of the unique identification.
For the VSS to support multiple cars in the view, it should at least improve the car detection. One might consider equipping each car with unique identification. Cars already have license plates mounted on them, however such features require license plate recognition. The Dutch government is considering wireless license plates that can be detected, like the AIS system, however there is no guarantee these systems will be applied within a reasonable time frame [1].

10.1.2. View edge track distribution triggers

The delay problem found during the testing phase greatly affect the scalability of the cooperative camera system. For the success of the approach it is important to find an approach that does not suffer the delay.
An alternative approach would be to send the track information when the car is near the edge of the region. The current approach uses the beginning and end of a track as the trigger to send the track segment to the next level. By changing it to every detection on the edge of the region, the higher level would no longer be sure the track is ended on one camera and would be more of an continuous flow track segments. The default case would still be just the start and and of the track, but it is possible for cars to stay on the edge longer. The worst case from the communication load point of view would be if cars drive over the edge for long periods of time. However, the delay problem would be reduced, since the track segment is send in the same cycle as the detection is made on the edge of the image. The region still has to wait one cycle to collect all the track segments from the local agents. This delay is reduced to one cycle for each layer.

10.1.3. Camera topology

The simulation allowed us to pick the camera configuration ourselves, which gave us the option to choose the camera topology that suited our needs. Real camera environment usually do not have the luxury of such configuration. Camera views tend to overlap of leaf unseen corners. We recommend further research to be done in reasoning about the overlapping and non-neighbouring camera views. Different topology of cameras (overlapping, non-neighbouring) also affect the hierarchy and reasoning on combining track segments. For example, the heuristic used in the current system of tracks appearing in one camera followed by the disappearing in the other camera is no longer applies to the situation. Future research should investigate how the camera topology affects the track combining process and find the best suited hierarchy for this topology.

10.1.4. Dynamic workload distribution

In the current research, we only assumed the hierarchy was defined at design time. The choice is justified, since the camera configuration does not ìncrease suddenly. Extensions tend to be scheduled investments that would allow the developers of the automatic surveillance system to re-evaluate the hierarchy and task distribution over agents. However, the rearranging of the tasks over the agents is a time consuming job and automatic dynamic workload distribution is generally a NP-complete problem [30]. It can only be optimally solved for small problems and only has known faster methods in certain special cases, such as balanced trees. Future work could investigate the automatic management on the hierarchical agents. This would allow the system to adjust to the addition and removal of cameras in the network, making it more scalable.

10.1.5. Bayesian networks suspicious behaviour detection

The reasoning methods applied in this thesis are still based on expert systems, a technology dated from the 70s. Since then, new artificial intelligence techniques have been introduced and many of them were also already applied in the visual surveillance domain [58]. The analysis of behaviour could be greatly improved by younger artificial intelligence techniques, such as Bayesian networks. Bayesian networks is already succesfully applied in the analysis of ship behaviour by other researches within the research group [68]. We decided to apply the logical reasoning method, due to the already unpredictable behaviour of cooperating agents. We were afraid that introducing statistical reasoning methods, the manageability of the system would be negatively affected, due to the many new possible outcomes of the agents reasoning. But we agree that the use of Bayesian networks would allow the system to cope with more complex behaviour patterns.

10.1.6. Online learning behaviour parameters

It is possible to learn the behaviour patterns online in contract to relying on expert knowledge to set the threshold parameters on design time. For car recognition, the system could be implemented using a Kalman filter, which would gradually find the best relation between the cars previous location and the expected new location [33]. The behaviour analysis could be extended with atypical behaviour detector. The system could learn the typical behaviour patterns by monitoring the cars during the learning period. As the system gain more knowledge about the typical behaviour, atypical behaviour could be detected, based on the distributions. The simulation environment stopped us from creating any online learning patterns (with the exception of background separation). But implementation in real environments would be able to apply these online learning methods for finding the behaviour parameters.

10.1.7. Build processing block test bench

For the current system, we decided to test the processing blocks, using integration tests. All the agents (and their components) are required to be fully operational during each of these tests. The design of the processing blocks allow for them to be tested separately, without the need for software agent in which it is situated, to be running. This would require a processing block test bench that would simulate the events otherwise generated by the agents. We decided not to include the test bench in the current work, due to the dependency on the JADE behaviour. However, we feel such a test bench would greatly improve the manageability of the software and reduce the development time. Future work could be done to develop such a test bench.

10.2. Application scope

We believe that the hierarchical model of cooperating agents in surveillance system has wider potential then the training facility in Den Helder. We feel that with some adjustments, the model would be a starting point in designing other surveillance systems. In this section, we broader our horizon and see what other research could be done after learning of the results from this thesis. We give our recommendations for future research in environments.

10.2.1. New behaviour scenarios

The most obvious recommendation is to apply the hierarchical reasoning model to other scenarios with different behaviour. The can be adjusted to fit highway patrol, parking garages management or (flood) evacuation management. For these research topics, the logical reasoning model would have to be adjusted accordingly. The hierarchical reasoning model would search as a guideline for the design, but the research focus would be mainly on the programming the expert knowledge in these cameras.

10.2.2. Advanced cameras

Cameras themselves become more and more versatile and the cameras mounted in a fixed position these days are only one variation of the many options. We define two types of cameras that we feel would greatly affect the reasoning model: Active panning, tilting & zooming (PTZ) cameras and mobile cameras.
PTZ cameras have the ability to rotate around the mounted point with two degrees of freedom and are able to zoom in. The cooperation between cameras would change significantly due to this ability. First, the camera view range would constantly change, changing the relative locations between two cameras. Neighbouring camera view could become overlapping or non-neighbouring. World mapping becomes dependent of the orientation of the camera and recursive recognition would have to incorporate the dynamic relative locations between the cameras in the reasoning for combining the tracks. These situations would complicate the processing blocks, but the hierarchical model would still be sufficient in coordinating these detections. When the cameras cooperate to zoom in on suspicious behaviour, the one-directional processing pipeline may no longer be sufficient. The coordination may require the cameras to communicate directly to each other to smoothly follow the suspicious movement. We would recommend to research the effects of PTZ cameras on the cooperative reasoning of the cameras and evaluate whether it would be better to allow direct communication or apply the same hierarchical reasoning model.
Mobile cameras have similar effects on the hierarchical reasoning model, since the camera view is constantly changing. But in addition to the viewpoint, the hierarchical reasoning model is based on the assumption that clusters of cameras are located within one region. Locations of mobile cameras are dynamic and could end up in different regions altogether. It would be interesting to study the effects of mobile cameras in hierarchical reasoning environments.

10.2.3. Multi-sensor environment

Cameras come equipped with many new features. Suspicious behaviour detection can be done based on many different measurements, for example shouting, radio-frequency identification (RFID) or even heat sensors. The main challenge for surveillance systems is to combine all of this knowledge in multi-modal data structure. We recommend a study in the effects on multi-modal data in a distributed reasoning system. The focus would mainly be in fusion the raw data types and event types within one environment.

10.2.4. Multi-object environment

Finally, we advise to study the effects of incorporating different type of objects in the system. Pedestrians have different behaviour compared to cars or cyclists. Obviously, the detection would be different, but more interesting, how would the system reasoning about the behaviour of these different objects. And what could we say about the interactions between the two? For the future work, we would advise to look into the mixture of different types of objects, which have unique behaviour patterns and interactions between them.
Include other entities, such as pedestrians or cyclists.

IV

Appendix

A

Publish–subscribe pattern

The guardian agent framework is designed as an event-driven system, where data event and time events trigger processes within and between agents. The multi-agent system is a collection of independent processes that should not be affected by the functionalities of other agents or other processes within the agent. We needed a method that would separate the functionalities of certain processes from the consequences of that event. Since each guardian agent is build out of numerous components that would have to be designed this way, we discuss the applied method once here. The problem is solved using the publish–subscribe pattern.
In this appendix, we will briefly discuss the publish–subscribe pattern and the application within the guardian agent framework. We will start with the design goal of the model, followed by the method of implementation. Finally, we will discuss the applications within the guardian agent framework and the different variations used.

A.1. Design goal

The goal of the publish–subscribe pattern is to decouple the publisher and subscriber. The functionalities of the publisher should not be affected by the processing of the subscriber. By generalizing the trigger event and standardizing the subscription method, the publisher has no further knowledge of the actions triggered by its event. There could be any possible actions triggered by the event, ranging from none to many actions. As a result, the process is more scalable and manageable. The decoupling separates the functionality of the components and allowing them to run on any thread or even agent. Publishers and subscribers can be tested individually and independently. A disadvantage could be that the components must be designed as independent processes, no longer allowing communication between the components. [28][14]

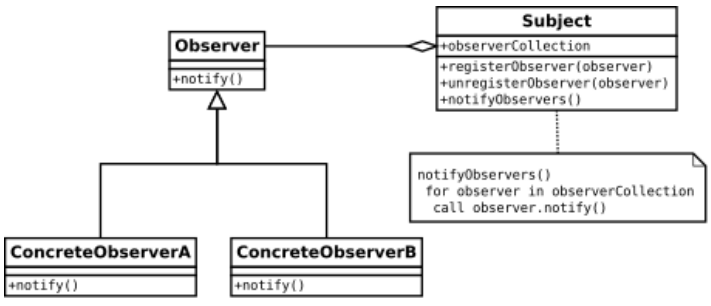

Figure A.1: Conceptual UML class diagram of the publish–subscribe pattern

A.2. Method

The design goal is achieved by introducing the concepts of the subject and the observer. The subject is any process that has events that we wish to publish to the observers. In order for the subject to notify the ob-

servers, it will have to manage the list of observers. Observers can subscribe to the subject, ensuring a place in the list of observers. The moment the subject has the event in question, the subject goes through the list of observers to notify them. The subject is decoupled from the observers, since how to notify the observer of the event. What actions the observer will perform based on this event is hidden behind the generalized interface. The UML class diagram of this model is shown in figure A.1.
Specifics of the notify function is left up to the system architect. The approach is either topic-based, content-based or an hybrid from the two approaches. Topic-based systems will publish all message based on the specific topic. The subscriber knows based on the topic what exactly has happend. Content-based sets specific constrains to event it wants notification from, but as a result will have to classify the message.

A.3. Applications within the guardian agent framework

The model is used often in different software components throughout the guardian agent framework. An overview of the applications is given in table A.1.

Table A.1: Application within the guardian agent framework

Publisher	Classified by publisher	Classified by subscriber
Event Log	Report scope	Event
Pipeline	Variable name	Record ID
Transport layer	Message type	
Service layer	Service	Function

The event logger uses the publish–subscribe pattern to separate local and global report of the event. Subscribers can be any type, from console output to automated testing objects. Some events can be handled within the agent itself, such as debug output. But for automatic testing purposes, we designed a service that distributed specific events to quality control agents. the publisher only knows the report scope of the message in order to decided to send it locally or globally to any subscriber. The subscriber will have to classify the content of the event message in order to validate the test.
The processing pipeline is the core of the car behaviour reasoning model. Processing blocks are linked together to form a processing chain, each triggered by its predecessor. The processing block will only be triggered by the predecessor and not by any other variable data change event. The publisher uses the variable name in order to notify the corresponding subscribers of the event. For the subscribers, the record identification allows them to identify what instance received an event.
Finally, two layers in the communication component use the publish-subscribe pattern. The guardian agent framework is intended to be an extension of the original JADE agent(s). We wanted to work alongside JADE and avoid removing features from it. JADE allows for different type of messages to be send between agents. Since our original design replaced the method of communication completely, we decided to implement the publish-subscribe pattern and have the transport layer identify the message type. Any messages supported by the guardian agent framework is send to the service layer and all other to the remaining observers. The service layer in turn, uses the publish-subscribe pattern to the corresponding service handler. Services are registered with the service handler in the same way the observers subscribe to the data events. Services are collections of functionalities and it is up to the service handler to identify the right function call.

B

Distributed key management

Distributed key management creates new challenges for information architects to manage data created over multi-processes systems. Decentralised assignment using for example incremental or randomly generated keys have the possibility of assigning the same identifier multiple times in different agents. We wanted a method to generate unique identifiers, without centralized approval for every new identifier. In this appendix, we would like to go over the chosen method. We will start with the design goal of the model, followed by the method of implementation. Finally, we will discuss the applications within the guardian agent framework.

B.1. Design goal

Data stored in rational database require field(s) that form an unique identification for each element, known as the primary key. The constrains set to the primary key are such that the unique identifier always represents one record. The generation of the unique identifiers has to ensure that two instances of the same primary keys are never generated. A simple procedure for generating unique identifiers is increasing the unique identifier for each new record. For single process databases with a few thousand records the mechanism works fine, but with it creates a problem with distributed databases.
We consider distributed databases to be multiple storages of data with the same structure, which combined form the full dataset. For example, a database could contain tables with track information or detections. Each database follows the same data structure and may contain a selection of all the records. If there are different clients that are allowed to create new entries at different, each of the individual databases could create new records and the generation of unique identifiers. When all the individual databases would follow the same routine for creating this database, we will most likely end up with non-unique identifiers.
Alternatively, the system could work with reservations of identifiers. Every new identifier is communicated with the other processes and reserved. This centralized method introduces a bottleneck in the communication and could slow down the process. Optimally, we would have a mechanism that requires as little communication as possible, while ensuring generation of unique identifier.

B.2. Method

The problem is solved by generating a number that is always unique. The identification generation process combines two numbers to end up with a unique identification. We will start by simply writing out this desired property. We are attempting to find the mechanism to combine the two fields (origin n and local identifier m) that creates the unique identifier in one field $f(n,m)$. The fundamental theorem of arithmetic shows us that any number can be decomposed into a unique multiplication of prime numbers. Whatever is the result of $f(n,m)$, we know is will follow the fundamental theorem of arithmetic. Because the numbers are unique, the multiplication is also unique. We are able to split the multiplication in two set, for example in low prime numbers and high prime numbers. As long as we pick a non-zero value for the two numbers, we have found a method of generating the unique identifiers, as is show here in equation B.1.

$$
\begin{aligned}
f(n,m) &= \prod_{k=1}^{L} p_k^{w_k} \\
&= \prod_{k=1}^{L_1} p_k^{w_k} * \prod_{k=L_1+1}^{L} p_k^{w_k} \\
&= n * m
\end{aligned} \tag{B.1}
$$

We find that if we pick a set of L prime numbers, sort them and split them into two groups, we have a simple mechanism to generate unique identifiers. We assign an unique number to each process and pick an unique number from the other group for each new record. The result is always a unique number.
Similar result can also be achieved by including the origin in the primary key. Some database allow multiple fields to be used to form the primary key of a record. As long as the identifier generation creates unique fields within one process, the combination of the two fields is always unique within the complete database. Although the two fields allow the system to generate unique identifiers with no communication, there are some setbacks to the use of multi field primary keys. Multi field primary keys increase the data size of each record and generally have slow queries. The search for a records based on two columns have no speed-up and search will have to be performed individually.
Although the proposed method does not require any communication for new identification numbers, the origin numbers still need to be assigned in a centralized manner. The main constrain for the origin field is that is it unique for each origin and the only method to achieve this seem to be validation from a centralized point. For this reason, every new database has to go through an initialisation process, which includes reserving the origin number in a single process.

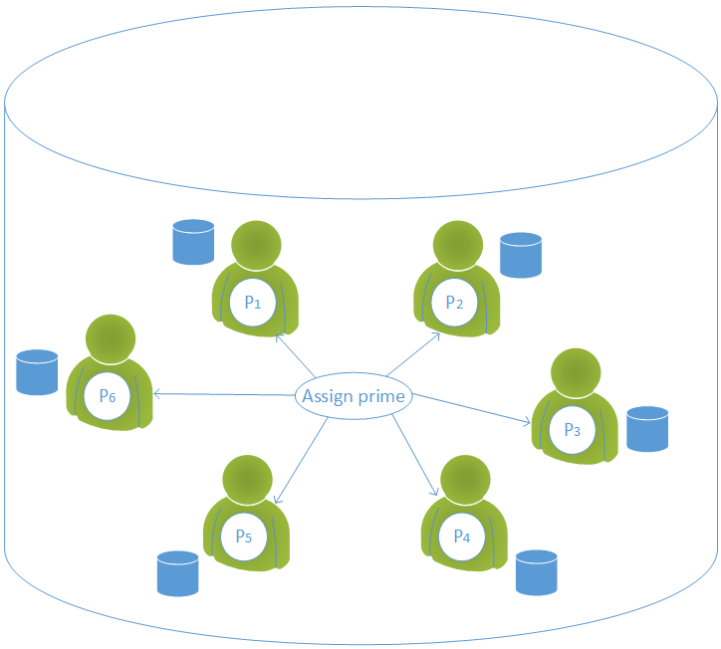

Figure B.1: Agent based distributed database

B.3. Implementation within guardian agent framework

The mechanism for generating unique identifier within a distributed agent network is applied for each situation in which the agents communicate information. For ease of programming, we decided to simplify the mechanism even further by only using prime numbers in the set of unique numbers. Every agent is assigned their own prime number and knows all the prime numbers between 1000 and 10000. The UML class diagram for the involved components in the generation of the unique identification numbers is given in figure B.2.

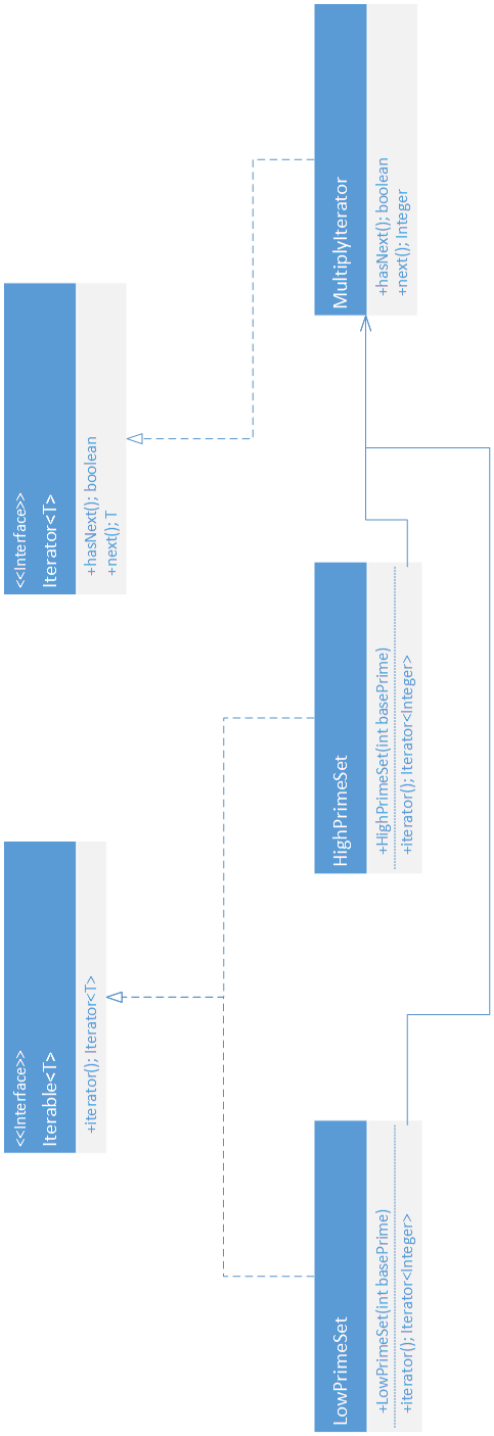

Figure B.2: Unique identifier generator

The distribution of the prime numbers is performed through the delegation agents. Once created, the agents will request the initial parameters from the delegator agent, including the agent's base prime.
The unique identification generation is used for creating new database entry, session identification in the service layer and the individual messages in the transport layer. The database entry start a new table for each variable. The session identification is initiated by the requesting party and is used until the the response is send to the corresponding requesting process. The transport layer currently only gives the messages a unique identification for debug purposes.

C

A Smart Surveillance System of Distributed Smart Multi Cameras modelled as Agents

Smart Cities Symposium Prague 2016

A Smart Surveillance System of Distributed Smart Multi Cameras modelled as Agents

D. Eigenraam, L.J.M. Rothkrantz

***Abstract*—A smart city is supposed to be a safe city. At this moment we observe multiple cameras in around public buildings and along streets to detect suspicious behavior. Samples of the video footage is monitored 24/7 by operators in control rooms. Currently the recorded videos are visual inspected after a suspicious event has happened. There is a need for real time monitoring with smart cameras which are able to detect, track and analyze suspicious objects over place and time. The increasing number of cameras requires a huge amount of monitoring operators Last year's we developed several prototypes of such a surveillance system using a distributed wireless network of smart cameras modelled as agents. The network has a hierarchical structure composed of different agents of increasing complexity, monitoring a street, crossing or even a global area of the city. In this paper we present the final version of our system, including design and test results.**

***Index Terms*—Agents, Smart Cameras, Surveillance systems, Rule Based Reasoning, Hierarchical Agent Network.**

I. Introduction

WORLD's leading information technology and advisory company Gartner defines a smart city as an urbanized area where multiple sectors cooperate to achieve sustainable outcomes through the analysis of contextual, real-time information shared among sector-specific information and operational technology systems. The question that needs to be answered is can smart cities be safer places for people to live and work? And what role does the advancement of technology and Internet of Things have to play in delivering public safety [1,2,3].

In this paper we consider automated detection of suspicious behavior of car drivers on the roads in a city. Every day many accidents happen in a city ranging from violations of traffic rules, car crashes up to aggressive behaviour involving cars. Nowadays many cameras are installed along the roads to survey the traffic streams. The video streams are monitored by operators in control rooms. Given the increasing amount of cameras it is evident that real time monitoring by human operators is no longer an option. There is a need for smart cameras able to detect, track and analyze the behavior of car drivers violating the rules or showing suspicious behavior.

D. Eigenraam is with the Department of Intelligent Interaction, Delft University of Technology (e-mail: G.D.Eigenraam@student.tudelft.nl)

L.J.M. Rothkrantz is with the Department Intelligent Interaction, Delft University of Technology and with the Faculty of Transport, Czech Technical University in Prague (e-mail: L.J.M.Rothkrantz@tudelft.nl)

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

Fig. 1. Surveillance camera of traffic streams (Axis).

Security surveillance means looking for the exception. The unwanted behavior can occur at any time and any place and there is a need for surveillance 24/7. In many cases incidents are localized but also many times an incident is distributed over a huge area and cooperation between individual cameras is needed. Take for example the tracking of a suspicious car-driver fleeing after an incident. Individual cameras are ordered in an hierarchical way and allow reasoning on different levels in the hierarchy.

In this paper we model every smart camera as an agent and consider a network of many intelligent agents (MAS). Over the past years, there has been done a lot of research on agent based technology. Agents within smart cameras would hold certain spatial awareness, which allow them to coordinate and equally distribute the surveillance task(s). It has mayor advantages compared to the traditional single process design The high level of parallelism allows the agent based system to solve more complex problems. Since every agent runs independently on a different processing unit, the total number of calculations can be much more and with the right design, the multi agent system has the potential to be a scalable system. Every agents is equipped with a knowledge based system and able to reason about observed features.

In [8] the concept of a surveillance agent was introduced to assist in potential dangerous environments. The human centered design applies software agents to assist operators and

improve the performance overall. Particular when the human operator is inclined to make mistakes due to low vigilance or heavy (mental) workload. The main problem for this research is the design of a distributed model for reasoning about car behavior, using multi-camera surveillance systems. The model will be designed as a Decision Support System (DSS), assisting security guards by extracting tracking information and classify specific car behavior. The topic is divided over the following sub problems:

- Design a model of cooperative smart cameras for automatic detection of suspicious behavior by hierarchical reasoning.
- Implement the model using JADE for multiple agent system and CLIPS for reasoning.
- Test different scenarios of cars travelling over a special test area showing different kinds of suspicious behavior.

The outline of this paper is as follows. First we present in section 2 related work, followed by tools. Then we display the architecture of our surveillance system in section 4 followed by a section on reasoning. Next we report about some implementation items in section 6. In section 7 we present our test framework and finally in section 8 we report about the outcomes of some tests.

II. Related Work

The Internal Security Program of the Finnish Ministry of the Interior identifies safety and security as key factors affecting people's wellbeing and movement and behavior in public places. It was studied in the AATU project [1], which was funded by Tekes, the Finnish Funding Agency for Technology and Innovation. Research partners in the project included VTT, University of Helsinki and Aalto-University. The Cities of Helsinki, Espoo and Vantaa provided interesting case study areas.

In [2] Clarke discusses trends driving Smart City growth, the vision for the Smart City of the future, and the maturity cycle that cities can progress through to become a Smart City. A vision of the future city, a city with a pervasive overlay of information and communication technology (ICT) connecting things, organizations, and people. For example, sensors connect cars to transportation management centers that analyze day-to-day traffic flow data and provide what-if scenarios in case of events or accidents.

In [3] a relevant aspect has been discussed when evaluating the city smartness related to the innovative approach to urban traffic management. A system called city kernel has been presented, designed to handle several subsystems, each addressing a specific sensor network and we describe an infrastructure for wide traffic control via a vision sensor network. This infrastructure consists of a network of smart cameras operating over an outdoor public lighting thanks to power line communication technology and equipped with a vehicle counting and classification algorithm.

In [4] researchers from the Smart Lab started a project on the design, implementation and deployment of innovative Smart Camera based intelligent systems for urban surveillance.

In [6] a comprehensive overview of existing state-of-the-art technologies developed for wireless video surveillance has been presented. Specifically, the physical network infrastructure for video transmission over wireless channel is analyzed. For video compression and transmission over the wireless networks, the ultimate goal is to maximize the received video quality under the resource limitation. In our paper we focus on local distributed processing of the video recordings.

In [8, 9, 10,11] prototypes of video surveillance systems are presented, able to detect and identify abnormal and alarming situations by analyzing object movement. The systems are designed to minimize video processing and transmission, thus allowing a large number of cameras to be deployed on the system, and therefore making it suitable for its usage as an integrated safety and security solution in Smart Cities.

III. Tools

Fig. 2. Layout of the used tools.

In this section we discuss the tools used to build our automatic surveillance system as displayed in Fig 2. MATLAB is used in this research to gather the knowledge about the common paths in a simulation environment. Due to the simulation environment, it was not possible to gather the common paths for the training phase. In short, we used a scripted system to calculate the characteristics of approved and unwanted behavior. The knowledge is used to define the rules of the automatic detection of unwanted behavior in the system.

The smart cameras, agents and software are all developed in a Java environment [3]. Eclipse provides IDEs and platforms nearly for every language and architecture. It is known for the Java IDE, C/C++, JavaScript and PHP IDEs built on extensible platforms for creating desktop, Web and cloud IDEs. These platforms deliver an extensive collection of add-on tools available for software developers [2]. Eclipse allowed us to setup a runtime environment for the smart cameras. It is used to program and test the intended automatic suspicious behavior detection and covers all that makes cameras smart.

The behavior analysis is performed using the expert rule based system CLIPS. The system CLIPS is an expert system platform, designed execution rule based reasoning. CLIPS provides methods of initiating facts and rules and provides the

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

process of reasoning. CLIPS is available as a Java Native Interface (JNI) and the reasoning engine can be managed from the Java code.

The design for the smart cameras implements an agent-based approach. JADE (Java Agent Development Framework) provides the software framework for the management of the agent's lifecycle and communication. It simplifies the implementation of multi-agent systems through a middle-ware that complies with the FIPA specifications and through a set graphical tools that support the debugging and deployment phases. A JADE-based system can be distributed across machines (which not even need to share the same OS) and the configuration can be controlled via a remote GUI. The configuration can be even changed at run-time by moving agents from one machine to another, as and when required. JADE is completely implemented in Java language.

In [7] we published a car license plate recognition system. This tool is needed to identify, recognize and track the cars. A model for an image vision system is presented that is capable of recognizing car license plates independent from plate location, size, dimension, color and character style. The system contains an image-processor, a segment-processor and five combined Neocognitron network classifiers which act as a character recognizer. The presented model of the system depends neither on specific license plate image features nor on license plate character style and size. The character recognition achieves over 98% accuracy.

IV. System &Software Design

This section focusses on the system design choices. It describes how the system components will work together. Our multi camera surveillance system is composed of three component groups: cameras, reasoning system and the terminal. The reasoning system consists of the collection of intelligent cameras that together form the automatic detection system. The reasoning system is intended to be an extension of the multi camera surveillance system and within the setup, the original and added functionalities of the VSS are separated. Multi camera surveillance systems have many cameras streaming images to the control room for further analysis. Our system is composed of intelligent cameras which have reasoning and communication capabilities. Since all the agents process information and communicate in a similar fashion, the generic Guardian Agent base class was developed to supply each of them with the framework to perform these actions. The added features include the pipeline structure, data and event logging and the services support system.

We decided to use services to communicate between agents. instead of the JADE communication protocol. Guardian Agent services are asynchronous procedure calls that are made available to other agents. A layered encoding system build on top of the existing communication method provided by JADE allows procedures to be called remotely. Each agent can support different services. The services are registered with the agent's service system, which will manage the triggering of call events and sessions during runtime.

Due to this extension, we now have a system on which every platform can run any agent and any agent can support its own set of services. There are five different types of agents: Alerting Agent, Delegation Agent, Global Reasoning Agent, Local Reasoning Agent and Regional Reasoning Agent.

The processing pipeline for the automatic behavior detection system runs through the local, regional and global agents. Starting with the agents connected to the cameras, each agent processes the available information and passes it on to the next process, creating an information flow. The local agents provide the local track data, which the regional agent subscribes to using the tunneling mechanism.

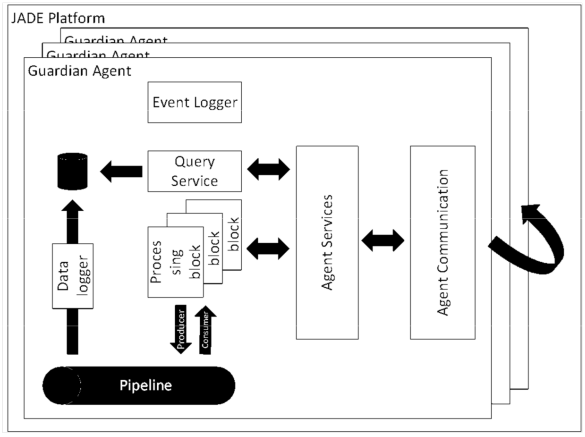

Fig. 3. Agent components.

The updates received by the regional agents are stored in the agents memory and historical values are managed within the agent, based on the functionalities of the agent. New features extracted by the regional agents in turn are provided to the next layer of agents and finally end up in the global agent.

In our suspicious behavior classification processing pipeline, agents perform five tasks: detect, map, recognize, extract features and reason. The relation between these agents has been designed as a hierarchical model, where each higher level gains a more global image of the car's behavior, as shown in Fig. 4. Reasoning is performed at multiple levels and distributed over different agents. Conceptually each agent has an area which it will have to guard. The larger the area, the more complex the behavior of the cars will be. This will create a bottleneck at the agents responsible for large areas. The bottleneck is counteracted by reducing the complexity of the information, for example with samples of lower frequency and resolution of the input. Local agents look at high resolution images with a high frequency, while global agents look at almost binary detections with low frequency over the sub-regions.

Each agent receives a series of images over time. The pipeline starts with detecting the cars in the image. The detection uses our license plate recognition system. Once the cars detections are known, the image coordinates have to be mapped to the environments coordinates. In real world environments, the mapping step requires the mapping of both the two dimensional image to three dimensional world and the relative location of the camera to the other cameras.

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

Once multiple detections are made, the detections must be combined to form a track. The recognition algorithm is applied for every new detection in the camera image. It uses the behavior of each track to select the most likely track for new detections, including the option of creating a new track. The final processing step extracts additional information about the track. The parameters calculated based on the track are used to classify the behavior of the car. The local feature extraction calculates the following features from the track: location x,y, heading, speed x,y, speed turn, acceleration x,y, duration.

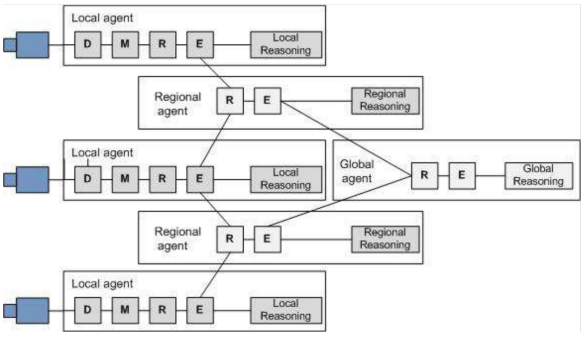

Fig. 4. Agent roles and communication.

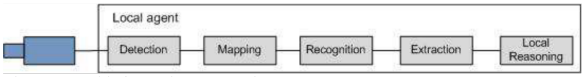

Fig. 5. Local information processing.

The regional agent analyses behavior over a small region with multiple cameras. It combines local tracks, forms regional tracks and analysis the behavior detectable within the scope of the cameras. Thanks to the cooperation between the guardian agents, the regional agent does not need to perform the same processing steps as the local agent. Several local agents go through the detection and tracking process as described before. The same local track information that is used for local reasoning is also send to the regional guardian agent. The communication between the agents is done using a combination of a pipeline and subscription technique.

The global information process highly resembles the regional information process. It combines the track from the different regions to one track through the complete region. Every new step in this track combination process loses more detail and tracks are considered with a lower resolution. The decrease in resolution, also decreases the bandwidth of the

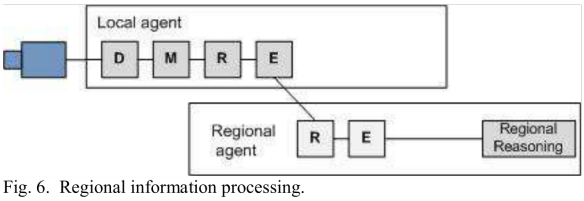

Fig. 6. Regional information processing.

data streams and avoids the bottleneck on the global agent's data stream. For this reason, the global information process connects region tracks. The tracks are combined using a probability model, including knowledge of the environment. The process is shown in Fig.7.

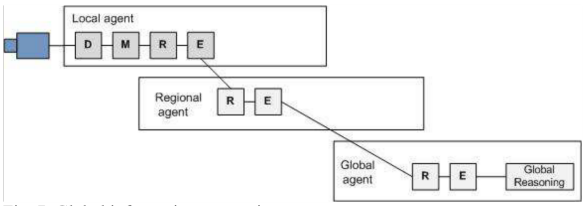

Fig. 7. Global information processing.

V. Reasoning

Each scenario will have a different reasoning method. The reasoning is performed with the rule based expert system CLIPS. Since the reasoning is performed over multiple cameras, the classification not only consist of the reasoning of agents but also with the cooperation between the agents. The reasoning for each scenario is discussed in terms of the rules applied in the reasoning and the task delegation over the different cameras. We discuss some examples:

Unwanted entry. In this scenario the location of a car will be considered and the correspondence to the restriction map. The reasoning only applies to the current measurements, no further memory of the past detections is required. It is triggered by any detection on any camera. No global or regional reasoning is required, only performance of the local agent.

if the car is on location l and l is a restricted area
then there is an unwanted entry at location l

Car tracking. This scenario starts with the license plate recognition by a local agent. At a crossing a car turns to the left or right, keeps in lane or parks somewhere which will be detected by local agents. The regional agent supervising the crossing takes a decision and messages the global agent. Successive observations of regional agents is considered by the global agents to make compose the car track. In case of misrecognition or missing observations the regional agents generalizes the observation. The global agent is also able to detect if a car circles around objects. In case of a fleeing car-driver the global agent is able to track the car and to generate a prediction of possible destinations.

VI. Implementation

The guardian agent design in section IV shows the concept of a new generic agent, including the components used in the generic camera behaviour agents to communicate, process and log-data and events. The new generic agent is created as an extension of the JADE agent and contains all the components required for the new features. Because it is the central component for all the processes that happen in the agent, it is designed as the central switchboard for internal communication between the components.

The agents are designed to support features and services. The services are implemented as Java classes with functions that are made available to all agents (including itself) and can

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

be called remotely. The main differences between services and regular classes are the synchronize and serializable properties. All the function calls of a service are encoded to a standard data structure and communicated using a three layered communication protocol. We will describe the implementation of the services and communication using a top down approach.

Any Java class can be converted to a service, but requires more components than just the class itself. Interfaces in Java can be used to separate the external functional description from the internal implementation of the features. They can even be used to dynamically change the functionalities of classes by instantiating different implementations of the interface, based on the state of the process. For the services we apply the interfaces to separate the functionalities from the actual process that will perform them. At the moment of the service call, the client process has no idea what process will perform the requested service. It only knows it will be handled. We will discuss how a regular class can be converted to a service in our system. The procedure unfortunately gets more complicated when functions have a return value and for clarity we will first discuss the procedure for functions without return value and later extend it.

Building a service requires at least four new components. The service layer is present in all agents. Building a service starts with the definition of the functions of the service. The service functionalities are defined using a service interface, which is known on both the server and client side. The service interface is a (Java) class interface which shows all the functions included in the service. Any service provider must implement these functions, which is also the second component. The service provider ultimately executes any of the functions when they are requested. The service provider is registered with the service layer to indicate it can handle any requests for this service.

Before remote agents can call the service functions, the call has to be encoded to and decoded from a message of the type that can be send between the agents. The encoding at the client side is performed by the stub component. The stub component implements the functions of the service interface, but in stead of performing the service, the stub encoded the request into a request message. The request message is send with the use of the service layer to the known agent that provides the service. The service layer is present at both the client and server agent. When building a service, the service layer is considered to be the gateway to the services on other agents and will send the messages on the client side and receive them on the server side. On the service providing side, the agent uses a skeleton to decode the message and call the appropriate functions on the service provider. The cooperation of these five components allow the call for functions of services on remote agents. However, when the functions have a return value the service layer becomes more complicated, which will be discussed shortly. First, we will look at how the messages are encoded and decoded within the stub and skeleton.

The services have a different implementation for each service, but they all run on the service layer as service providers. The service layer is essentially a library of service providers, which manages the life cycle of the service calls. The registration of services uses a subscription method. The incoming messages feature as events, which are classified by service and send to the corresponding service provider. In the case of a response value, a session is started and managed by the service layer.

The processing structure contains a distributed processing pipeline. On this pipeline we find multiple processing steps, which starts with the images from the cameras and finally classify (suspicious) behavior of the cars. The pipeline implementation is based on the same principles as OPC. All the values are stored on a data bus and provided with a subscription algorithm. The pipeline is extended with data logging components, which allows not just the current data to be stored, but also the historical data. And finally, CLIPS uses a different syntax to store the data (or known in CLIPS as facts). We developed a mechanism to convert the data objects used in our implementation to CLIPS facts syntax.

Every guardian agent is equipped with memory in the form of a database. The data bus only includes the latest value for each variable bus and to also has the historical data available. To design a form of memory. We decided to use a data logger system for the data bus to store the past values of variables. The database creates one table for each variable and has no prior knowledge about the data structures of the values themselves. But the table entries need unique identifiers or primary keys to separate between the entries. In this section, we will discuss the data log system, the identifier generation and the table query mechanism.

VII. Test Framework

The system is tested using a custom made unit testing framework. The framework is an extension to the JUnit Testing framework. Although unit testing is a mature technique for sequential coding, testing parallel processes or multi-agent system is still a challenging topic. Agent based systems assume the agents to perform autonomous. This greatly influences the testability of the agents, since they are designed as a black box system. The only method to test the functionalities of an agent is to communicate with it or analyze the result in the user interface or database. By design it is not allowed to interrupt the process for any external purposes, including testing.

A commonly used technique in software testing is the use of mocking classes. The mocking classes perform the same actions as the real implementation would perform. By calling functions, the mocking classes check if the components under testing produce the expected results or can check if the components under testing call functions at the right time (and with the right parameters). The mocking classes themselves form no useful aspect of the final product, but can give insight in the functionalities of the components under testing.

Mock agents (MA) form a similar addition to the multi-agent system. Although they do not add anything to the functionalities of the multi-agent system, they communicate with the agents under testing (AUT) using the same methods the AUT use. The AUT have no method to check if the system is used for testing or for real environment. It processes the information as normal. The mock agents in turn can validate the information from the AUT and notify if the test has passed

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

or failed.

Mock agents solve the problem of autonomously of the AUT, but the MA are agents themselves and by design are autonomous. We need a method to listen to the MA to check their conclusions and possibly set parameters. JADE does provide a method to do so. It is possible to get the reference to the life agents through the agent's controller, provided the agent has a registered agent to class interface. Any agent that is called from the unit test environment has to register and implement corresponding agent to class interface. This is strongly discouraged, because it both changes the functionalities of the agent and reduces the autonomy of the agent. For our mock agents however, this is no problem, since they are not the agents under testing and their functionality is not in question.

Our design of the multi-agent unit testing environment uses two types of mocking agents: actors and quality control agents. The actors only pretend to be agents that in the real environment would communicate with the AUT. They are used to trigger an event in the AUT by communicating the scripted messages. The Quality Control Agents (QCA) validate the result. Every QCA is responsible for performing one or more tests and validate if the messages from the AUT contain the expected information. A multi-agent unit test succeeds when all the QCAs report a success in there tests. The UML Class diagram of the mock agent design is shown in Fig. 8.

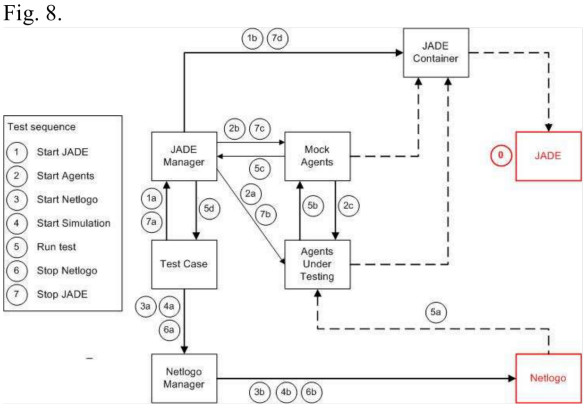

Fig. 8. UML Class diagram mock agent design.

VIII. Test Environment

At this moment commercial versions of smart camera surveillance systems are available. Such systems are able to detect violations of traffic rules or and traffic incidents in an automated way (Figure 1). But to detect a specific car crossing the city or to detect a car exploring specific areas of a city needs further research and development from most companies. We will report our results of experiments.

It is complicated or even not allowed to perform some experiments to collect data to design or to train a system. Suspicious car drivers preparing terroristic attacks, bank robbery are fortunately rare events and difficult to observe in real life. Similar remarks about ghost drivers, street racing etc. There is a need for a test environment to perform special experiments. As test area the Campus of the Netherlands Military Academy was selected. This area has a street network surveilled by a dense network of cameras connected to a control room where the video data is monitored by operators. At the campus many students are available to take part in experiments on suspicious driving behavior.

Our system was tested successfully using the scenarios, illegal parking, passing speed limit, entering one way streets and other scenarios on violations of traffic rules. But we were also able to track cars after an incident and trying to escape. From the observed track so far conclusions could be taken about possible destinations such as the exit of the campus on their way to the exit. We were also able to detect cars circling around the campus without specific goal and labeled as suspicious. We got our good results under lighting conditions (daytime, streetlights) and good weather. Next future we will research the performance of our system under bad weather conditions and bad lighting conditions.

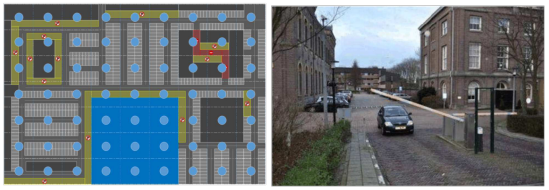

Fig. 9. Camera distribution within the test environment covering all streets, squares and parking places and entrance of the military campus.

References

[1] J. Keränen, M. Vaattovaara, M. Kortteinen, R. Ratvio, H. Koski, T. Rantala. 2013. "Safe living environments. Structural backgrounds, residents' experiences and everyday safety solutions". VTT Technology Espoo. pp.251. Available: http://vtt.fi/inf/pdf/technology/2013/T88.pdf.

[2] R. Yesner Clarke, "Smart Cities and the Internet of Everything: The Foundation for Delivering Next-Generation". Citizen Services, 2013, IDC, Government Insights.

[3] L. Calderoni, D. Maio, S. Rovis, "Deploying a network of Smart cameras for Traffic Monitoring on a city Kernel" Expert Systems with Applications, vol. 41 pp. 502-507, 2014.

[4] G. Cardone, A. Cirri, A. Corradi, L. Foschini. "Activity recognition for Smart City scenarios: Google Play Services vs. MoST facilities", in Computers and Communication (ISCC), 2014, pp 1-6.

[5] Yun Ye, Song Ci, K. Aggelos, K. Katsaggelos, YanWei Liu, Yi Qian "Wireless Video Surveillance: A Survey", IEEE Access, 2013.

[6] L. Calavia, C. Baladrón, J. M. Aguiar, B. Carro, A. Sánchez-Esguevillas. "A Semantic Autonomous Video Surveillance System for Dense Camera Networks in Smart Cities", Sensors, vol. 12 (8), pp. 10407-10429, 2012.

[7] L. Rothkrantz, "Surveillance angels," Neural network world 24, 1-25

[8] P. Hameete, S Leysen, T. van der Laan, I Lefter, L. Rothkrantz, "Intelligent multi-camera video surveillance," International Journal on Information Technologies & Security 4 (4), 2012

[9] I. Lefter, L. J. M. Rothkrantz, M. Somhorst. "Automated safety control by video cameras," in Proceedings of the 13th International Conference on Computer Systems and Technologies, Rousse, 298-305

[10] L. Rothkrantz. Dynamic routing using the network of car drivers, in Proceedings of the Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship, ACM, Prague, 2009.

[11] L. Rothkrantz. "Smart Surveillance Systems," Network Topology in Command and Control: Organization, Operation, and Evolution: Organization, Operation, and Evolution, IGI, 2014, 270-280.

978-1-5090-1116-2/16/$31.00 ©2016 IEEE

Bibliography

[1] Karin Alberts, 2015. URL http://nos.nl/artikel/2041098-chip-moet-einde-maken-aan-fraude-met-kentekens.html, 21-06-2016 (Accessed: 21-06-2016).

[2] Xinfeng Bao, Solmaz Javanbakhti, and Svitlana Zinger. Moving ship detection based on context modeling and motion analysis. 2014.

[3] Arslan Basharat, Alexei Gritai, and Mubarak Shah. Learning object motion patterns for anomaly detection and improved object detection. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008. ISBN 1424422426.

[4] José Luis Bermúdez. *Cognitive science: An introduction to the science of the mind*. Cambridge University Press, 2014. ISBN 978-0-521-70837-1.

[5] Alina Bialkowski, Simon Denman, Sridha Sridharan, Clinton Fookes, and Patrick Lucey. A database for person re-identification in multi-camera surveillance networks. In *Digital Image Computing Techniques and Applications (DICTA), 2012 International Conference on*, pages 1–8. IEEE, 2012. ISBN 146732180X.

[6] James Black, Dimitrios Makris, and Tim Ellis. Hierarchical database for a multi-camera surveillance system. *Pattern Analysis and Applications*, 7(4):430–446, 2004. ISSN 1433-7541.

[7] Michael Bramberger, Bernhard Rinner, and Helmut Schwabach. A method for dynamic allocation of tasks in clusters of embedded smart cameras. In *2005 IEEE International Conference on Systems, Man and Cybernetics*, volume 3, pages 2595–2600. IEEE, 2005. ISBN 0780392981.

[8] Michael Bramberger, Andreas Doblander, Arnold Maier, Bernhard Rinner, and Helmut Schwabach. Distributed embedded smart cameras for surveillance applications. *Computer*, 39(2):68–75, 2006. ISSN 0018-9162.

[9] Chad Brooks, 2014. URL http://www.businessnewsdaily.com/6897-video-surveillance-systems.html, 21-06-2016 (Accessed: 21-06-2016).

[10] Federico Castanedo, Miguel A. Patricio, J. Garcia, and José M. Molina. Extending surveillance systems capabilities using bdi cooperative sensor agents. In *Proceedings of the 4th ACM international workshop on Video surveillance and sensor networks*, pages 131–138. ACM, 2006. ISBN 1595934960.

[11] Nicolas Chleq and Monique Thonnat. Realtime image sequence interpretation for video-surveillance applications. In *Image Processing, 1996. Proceedings., International Conference on*, volume 1, pages 801–804. IEEE, 1996. ISBN 0780332598.

[12] C. Choy, M. Chung, G. Harahap, M.R. Natadarma, and S.L. Wu. *Surveillance system using abandoned luggage detection*. Thesis, TU Delft, Delft University of Technology, 2007.

[13] Roberta Coelho, Uirá Kulesza, Arndt von Staa, and Carlos Lucena. Unit testing in multi-agent systems using mock agents and aspects. In *Proceedings of the 2006 international workshop on Software engineering for large-scale multi-agent systems*, pages 83–90. ACM, 2006. ISBN 1595933956.

[14] James W. Cooper. *C# design patterns: a tutorial*. Addison-Wesley Professional, 2002. ISBN 0-201-84453-2.

[15] Dragos Datcu, Zhenke Yang, and Léon J.M. Rothkrantz. Multimodal workbench for automatic surveillance applications. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–2. IEEE, 2007. ISBN 1424411793.

[16] C. Lakshmi Devasena, R. Revathí, and M. Hemalatha. Video surveillance systems–a survey. *International Journal of Computer Science (IJCSI)*, 8(4), 2011.

[17] CLIPS development team, 2016. URL http://clipsrules.sourceforge.net/, 21-06-2016 (Accessed: 21-06-2016).

[18] Eclipse development team, 2016. URL https://www.eclipse.org/home/index.php, 21-06-2016 (Accessed: 21-06-2016).

[19] Jade development team, 2016. URL http://jade.tilab.com/, 21-06-2016 (Accessed: 21-06-2016).

[20] Java development team, 2016. URL https://www.java.com/en/about/, 21-06-2016 (Accessed: 21-06-2016).

[21] Java development team, 2016. URL https://docs.oracle.com/javase/8/docs/technotes/guides/serialization/index.html, 21-06-2016 (Accessed: 21-06-2016).

[22] JUnit development team, 2016. URL http://junit.org/junit4/, 21-06-2016 (Accessed: 21-06-2016).

[23] NVidia CUDA development team, 2015. URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/#streams, 21-06-2016 (Accessed: 21-06-2016).

[24] OPC Foundation Team development team, 2015. URL https://opcfoundation.org/, 21-06-2016 (Accessed: 21-06-2016).

[25] Qognify development team, 2016. URL http://www.qognify.com/, 21-06-2016 (Accessed: 21-06-2016).

[26] Raspberry Pi development team, 2016. URL https://www.raspberrypi.org/products/model-b-plus/, 21-06-2016 (Accessed: 21-06-2016).

[27] M Bernardine Dias, Robert Zlot, Nidhi Kalra, and Anthony Stentz. Market-based multirobot coordination: A survey and analysis. *Proceedings of the IEEE*, 94(7):1257–1270, 2006. ISSN 0018-9219.

[28] Erich Gamma. *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995. ISBN 8131700070.

[29] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Upper Saddle River, New Jersey 07458, Upper Saddle River, New Jersey 07458, 2008. ISBN 978-0-135-05267-9.

[30] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. *Introduction to Parallel Computing*. The Benjamin/Cummings Publishing Company, Inc., 2 edition, 2003. ISBN 978-0-201-64-64865-2.

[31] P. Hameete, S. Leysen, T. van der Laan, Iulia Lefter, and Léon J.M. Rothkrantz. Intelligent multi-camera video surveillance. *International Journal on Information Technologies and Security*, 4(4), 2012. ISSN 1313-8251.

[32] Robert Horlings, Dragos Datcu, and Léon J.M. Rothkrantz. Emotion recognition using brain activity. In *Proceedings of the 9th international conference on computer systems and technologies and workshop for PhD students in computing*, page 6. ACM, 2008. ISBN 954964152X.

[33] Weiming Hu, Tieniu Tan, Liang Wang, and Steve Maybank. A survey on visual surveillance of object motion and behaviors. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 34(3):334–352, 2004. ISSN 1094-6977.

[34] Raymond Huisman. *Scheduling the refuelling activities of multiple heterogeneous autonomous mobile robots*. Thesis, TU Delft, Delft University of Technology, 2014.

[35] Omar Javed, Zeeshan Rasheed, Orkun Alatas, and Mubarak Shah. Knight™: a real time surveillance system for multiple and non-overlapping cameras. In *Multimedia and Expo, 2003. ICME'03. Proceedings. 2003 International Conference on*, volume 1, pages I–649–52 vol. 1. IEEE, 2003. ISBN 0780379659.

[36] Christine Kern, 2014. URL http://www.bsminfo.com/doc/ip-video-growth-predicted-to-surpass-other-se 21-06-2016 (Accessed: 21-06-2016).

[37] Jon Kleinberg and Éva Tardos. *Algorithm Design*. Pearson Education, Inc., 2006. ISBN 978-0-321-29535-8.

[38] Alper Kemal Koç. *Body posture analysis in the context of shopping*. Thesis, Philips Research, 2012.

[39] Daniel Kuettel, Michael D. Breitenstein, Luc Van Gool, and Vittorio Ferrari. What's going on? discovering spatio-temporal dependencies in dynamic scenes. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1951–1958. IEEE, 2010. ISBN 1424469848.

[40] Quoc V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013. ISBN 1520-6149.

[41] Iulia Lefter, Léon J.M. Rothkrantz, and Gertjan J. Burghouts. A comparative study on automatic audio–visual fusion for aggression detection using meta-information. *Pattern Recognition Letters*, 34(15): 1953–1963, 2013. ISSN 0167-8655.

[42] Iulia Lefter, Gertjan J. Burghouts, and Léon J.M. Rothkrantz. An audio-visual dataset of human–human interactions in stressful situations. *Journal on Multimodal User Interfaces*, 8(1):29–41, 2014. ISSN 1783-7677.

[43] Iulia Lefter, Gertjan J. Burghouts, and Léon J.M. Rothkrantz. Recognizing stress using semantics and modulation of speech and gestures. *IEEE Transactions on Affective Computing*, 7(2):162–175, 2016. ISSN 1949-3045.

[44] Jingwen Li, Lei Huang, and Changping Liu. Robust people counting in video surveillance: Dataset and system. In *Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on*, pages 54–59. IEEE, 2011. ISBN 1457708442.

[45] Nan Lu, Jihong Wang, Q.H. Wu, and Li Yang. An improved motion detection method for real-time surveillance. *IAENG International Journal of Computer Science*, 35(1):1–10, 2008. ISSN 1819-656X.

[46] Tim Mackinnon, Steve Freeman, and Philip Craig. Endo-testing: unit testing with mock objects. *Extreme programming examined*, pages 287–301, 2001.

[47] Arnold Maier, Bernhard Rinner, and Helmut Schwabach. A hierarchical approach for energy-aware distributed embedded intelligent video surveillance. In *11th International Conference on Parallel and Distributed Systems (ICPADS'05)*, volume 2, pages 12–16. IEEE, 2005. ISBN 0769522815.

[48] Emanuele Menegatti, Enzo Mumolo, Massimiliano Nolich, and Enrico Pagello. A surveillance system based on audio and video sensory agents cooperating with a mobile robot. In *Proc. of 8th International Conference on Intelligent Autonomous Systems (IAS-8)*, pages 335–343, 2004.

[49] C. Micheloni, G.L. Foresti, and L. Snidaro. A cooperative multicamera system for video-surveillance of parking lots, 2003.

[50] Rich Miller, 2013. URL `http://www.securitysystemsnews.com/article/demand-security-equipment-projected-rise-7-percent-year-through-2016`, 21-06-2016 (Accessed: 21-06-2016).

[51] J. Glenford Myers, Tom Badgett, and Corey Sandler. *The art of software testing*. John Wiley & Sons, Inc., 3 edition, 2004. ISBN 978-1-118-03196-4.

[52] Herbert A. Nye. The problem of combat surveillance. *IRE Transactions on Military Electronics*, 4(MIL-4): 551–555, 1960. ISSN 0096-2511.

[53] Maja Pantic and Léon J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on pattern analysis and machine intelligence*, 22(12):1424–1445, 2000. ISSN 0162-8828.

[54] Mirela Popa, Léon J.M. Rothkrantz, Zhenke Yang, Pascal Wiggers, Ralph Braspenning, and Caifeng Shan. Analysis of shopping behavior based on surveillance system. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 2512–2519. IEEE, 2010. ISBN 1424465869.

[55] Mirela C. Popa, Léon J.M. Rothkrantz, Caifeng Shan, Tommaso Gritti, and Pascal Wiggers. Semantic assessment of shopping behavior using trajectories, shopping related actions, and context information. *Pattern Recognition Letters*, 34(7):809–819, 2013. ISSN 0167-8655.

[56] Mirela C. Popa, Léon J.M. Rothkrantz, Pascal Wiggers, and Caifeng Shan. Shopping behavior recognition using a language modeling analogy. *Pattern Recognition Letters*, 34(15):1879–1889, 2013. ISSN 0167-8655.

[57] Carlo S Regazzoni, Visvanathan Ramesh, and Gian Luca Foresti. Special issue on video communications, processing, and understanding for third generation surveillance systems. *Proceedings of the IEEE*, 89(10): 1355–1367, 2001. ISSN 0018-9219.

[58] Nathanael Rota and Monique Thonnat. Video sequence interpretation for visual surveillance. In *Proceedings of the Third IEEE International Workshop on Visual Surveillance (VS'2000)*, page 59. IEEE Computer Society, 2000. ISBN 0769506984.

[59] Léon J. M. Rothkrantz. Een digitale engelbewaarder. 2010.

[60] Léon J.M. Rothkrantz. Crisis management using multiple camera surveillance systems. In *ISCRAM 2013: Proceedings of the 10th International Conference on Information Systems for Crisis Response and Management, Baden-Baden, Germany, 12-15 May 2013*. ISCRAM, 2013. ISBN 3923704801.

[61] Léon J.M. Rothkrantz. Smart surveillance systems. *Network Topology in Command and Control: Organization, Operation, and Evolution: Organization, Operation, and Evolution*, page 270, 2014. ISSN 1466660597.

[62] Léon J.M. Rothkrantz. Surveillance angels. *Neural Network World*, 24(1):3, 2014. ISSN 1210-0552.

[63] Léon J.M. Rothkrantz and Iulia Lefter. Risk analysis of a video-surveillance system. In *Proceedings of the 12th International Conference on Computer Systems and Technologies*, pages 387–392. ACM, 2011. ISBN 1450309178.

[64] Léon J.M. Rothkrantz and Krispijn Scholte. A surveillance system of a military harbour using an automatic identification system. In *Proceedings of the 14th International Conference on Computer Systems and Technologies*, pages 169–176. ACM, 2013. ISBN 145032021X.

[65] Léon J.M. Rothkrantz, Pascal Wiggers, Jan-Willem A. van Wees, and Robert J. van Vark. Voice stress analysis. In *International conference on text, speech and dialogue*, pages 449–456. Springer, 2004.

[66] Tomi D. Räty. Survey on contemporary remote surveillance systems for public safety. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 40(5):493–515, 2010. ISSN 1094-6977.

[67] M. Sankari and C. Meena. Estimation of dynamic background and object detection in noisy visual surveillance. *International Journal of Advanced Computer Sciences and Applications*, 6(2), 2011.

[68] Krispijn A. Scholte. *Detecting suspicious behavior in Marine traffic using the Automatic Identification System*. Thesis, TU Delft, Delft University of Technology, 2013.

[69] Rene Schuster, Roland Mörzinger, Werner Haas, Helmut Grabner, and Luc Van Gool. Real-time detection of unusual regions in image streams. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1307–1310. ACM, 2010. ISBN 1605589330.

[70] Maarten Somhorst. *Multi-camera video surveillance system*. Thesis, TU Delft, Delft University of Technology, 2012.

[71] William M. Thames. From eye to electron—management problems of the combat surveillance research and development field. *IRE Transactions on Military Electronics*, 1051(4):548–551, 1960. ISSN 0096-2511.

[72] Maria Valera and Sergio A. Velastin. Intelligent distributed surveillance systems: a review. *IEE Proceedings-Vision, Image and Signal Processing*, 152(2):192–204, 2005. ISSN 1350-245X.

[73] Jean Vaucher and Ambroise Ncho, 2003. URL `http://www-labs.iro.umontreal.ca/~vaucher/Agents/Jade/Tutorial/Ontologies.htm`, 21-06-2016 (Accessed: 21-06-2016).

[74] Gerd Wagner. Towards agent-oriented information systems. *T echnical report, F reie U niversit iat Berlin, Institut f iur Informatik*, 1999.

[75] Xiaogang Wang. Intelligent multi-camera video surveillance: A review. *Pattern recognition letters*, 34(1): 3–19, 2013. ISSN 0167-8655.

[76] Christopher D. Wickens. Multiple resources and mental workload. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, 50(3):449–455, 2008. ISSN 0018-7208.

[77] Rob Wijnhoven, Kris van Rens, Egbert G.T. Jaspers, and Peter H.N. de With. Online learning for ship detection in maritime surveillance. In *Proc. of 31th Symposium on Information Theory in the Benelux*, pages 73–80. Citeseer, 2010.

[78] Yi Xie, Liang Lin, and Yunde Jia. Tracking objects with adaptive feature patches for ptz camera visual surveillance. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 1739–1742. IEEE, 2010. ISBN 1424475422.

[79] Zhenke Yang and Léon J.M. Rothkrantz. Surveillance system using abandoned object detection. In *Proceedings of the 12th International Conference on Computer Systems and Technologies*, pages 380–386. ACM, 2011. ISBN 1450309178.

[80] Wojtek Zajdel, Johannes D. Krijnders, Tjeerd Andringa, and Dariu M. Gavrila. Cassandra: audio-video sensor fusion for aggression detection. In *Advanced Video and Signal Based Surveillance, 2007. AVSS 2007. IEEE Conference on*, pages 200–205. IEEE, 2007. ISBN 1424416965.