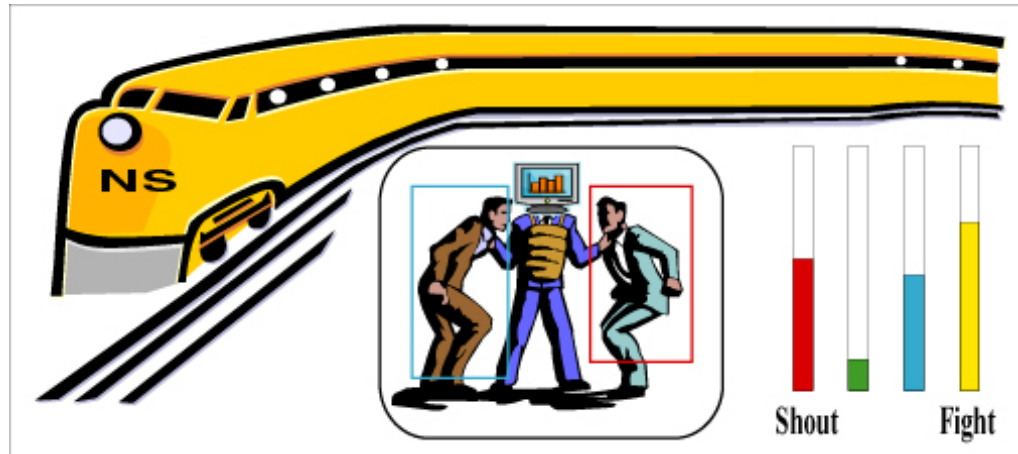


Video content analysis & aggression detection system for a train environment



Mohannad Ismail

November 2007



Delft University of Technology
Faculty of Electrical Engineering, Mathematics and Computer Science
Mediamatics: Man-Machine Interaction

Video content analysis & aggression detection system for a train environment

Master's Thesis in Media & Knowledge Engineering

Man-Machine Interaction Group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Mohannad Ismail
1233246

November 2007

Man-Machine Interaction Group

Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Members of the Supervising Committee

Dr. drs. L.J.M. Rothkrantz
Ir. H.J.A.M. Geers
Dr. ir. C.A.P.G. van der Mast
Ir. Z. Yang

Abstract

Video content analysis & aggression detection system for a train environment

Mohannad Ismail, student number: 1233246
Delft, November 2007

Man-Machine Interaction Group

Faculty of Electrical Engineering, Mathematics,
and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Members of the Supervising Committee

Dr. drs. L.J.M. Rothkrantz
Ir. H.J.A.M. Geers
Dr. ir. C.A.P.G. van der Mast
Ir. Z. Yang

Aggression in trains is increasing annually, the aggression is costing the Dutch railroads NS a lot of money due to destroying, or drawing graffiti on the interior of the train. On the other hand, the aggression towards the train conductors is causing conductors to quit their job, which is not what the NS wants due to the shortage they have in conductors. Passengers are also not feeling safe on the train. Current video-surveillance systems have limited intelligence, necessitating the employment of human operators who are able to interpret the images. This job is tedious and has a low efficiency: Only a small fraction of the image stream contains interesting information.

At Delft University of Technology, there is an ongoing project on aggression detection in trains by multiple single modality devices that capture data from different modalities, hence “multimodal”. Using video and sound input, an intelligent system, has to be developed that is able to make a context sensitive interpretation of human behavior on a train. The aim is to detect aggression as it is about to happen.

In this thesis we investigate the behavior of the human in the train. We designed and implemented a system with high usability standards with which users can annotate situations in the train compartments. We are particularly interested in aggression. The input to the annotating process is images that are captured from video data of recorded scenarios of aggressive and non aggressive situation. We implemented a user interface that is connected to a rule-based expert system to handle the incoming data from the annotating process. The output gives an aggression level and an aggression classification.

We designed and implemented a demonstration and have tested the system. The model, implementation and test results will be described in this thesis.

Acknowledgements

First and foremost, I would like to thank my thesis supervisors, Leon Rothkrantz and Zhenke Yang for their inspirations that they gave me during my thesis project. Their wide knowledge and great foresight have provided a good basis for the present thesis. Their knowledge and instructions helped me to conquer a lot of challenges in the process of my research.

Furthermore, I would like to thank the other PhD. and MSc. students and other people from the university who helped me during the project and gave me useful advices during my thesis project.

Last, but certainly not least, I would like to thank my parents, my brother, and my fiancée for supporting me day and night and even in the most difficult times.

Table of content

ABSTRACT	IV
ACKNOWLEDGEMENTS	V
TABLE OF CONTENT	VI
LIST OF FIGURES.....	VIII
LIST OF TABLES.....	IX
1 INTRODUCTION	1
1.1 RELEVANCE	2
1.2 PROBLEM DEFINITION	3
1.3 GOALS	3
1.4 APPROACH.....	4
1.5 THESIS OVERVIEW	5
2 BACKGROUND.....	7
2.1 HUMAN AGGRESSION	7
2.2 AGGRESSION IN TRAIN	12
2.3 THE CURRENT STATE OF THE ART	13
3 PROBLEM ANALYSIS.....	17
3.1 AGGRESSION IN THE TRAIN	17
3.2 AGGRESSION DETECTION BY HUMANS	18
3.3 AN ANALYSIS OF THE TRAIN COMPARTMENT	20
3.4 AN ANALYSIS OF THE TRAIN COMPARTMENT: THE STATIC CONTEXT	21
3.5 AN ANALYSIS OF THE TRAIN COMPARTMENT: THE DYNAMIC CONTEXT	21
3.6 AN ANALYSIS OF THE TRAIN COMPARTMENT: ACTIVITIES & BEHAVIORS	23
3.7 AN ANALYSIS OF THE SYSTEM NEEDED	26
3.8 REQUIREMENTS	28
4 ANNOTATION LANGUAGE FOR TRAINS	33
4.1 ANNOTATION LANGUAGE	33
4.2 REASONING MODEL	35
4.3 LANGUAGE SYNTAX.....	37
4.4 WHAT TO ANNOTATE	38
4.5 FUTURE OF THE LANGUAGE	41

5 DESIGN.....	43
5.1 ARCHITECTURE.....	43
5.2 THE GUI COMPONENT	44
5.3 THE VALIDATOR COMPONENT	45
5.4 REASONING COMPONENT	45
5.5 THE XML/ IO COMPONENT	48
5.6 CLASS DIAGRAMS	50
5.7 DYNAMIC BEHAVIOR	53
5.8 PROGRAM FLOW AND GLOBAL USAGE.....	56
6 IMPLEMENTATION	59
6.1 TOOLS & SOFTWARE	59
6.2 IMPLEMENTATION OF THE EXPERT SYSTEM.....	60
6.3 GUI & SCREENSHOTS	64
6.4 FLEXIBILITY.....	68
7 TESTING & EXPERIMENT RESULTS	73
7.1 DATA COLLECTING	73
7.2 TEST PLAN	74
7.3 EXPERIMENT RESULT	80
8 CONCLUSIONS & RECOMMENDATIONS.....	85
8.1 CONCLUSION	85
8.2 RECOMMENDATIONS.....	87
REFERENCE	89
APPENDIX A	93
APPENDIX B.....	95
APPENDIX C	97
APPENDIX D	103
APPENDIX E.....	109
APPENDIX F	119
APPENDIX G	123
APPENDIX H	127
APPENDIX G	133

List of figures

Figure 1.1: Surveillance camera	2
Figure 2.1: The location of the human Hypothalamus in the brain	11
Figure 2.2: the general aggression model episodic processes	12
Figure 2.3: Flowchart of the learning unit.....	14
Figure 3.1: Feature points (fiducials of the contours of the face components and of the profile contour)	24
Figure 3.2: Exemplar set for walking	25
Figure 3.2: Use Case diagram	27
Figure 4.1: Parse tree “Mike hit the ball”	38
Figure 4.2: Example of annotating a beggar	41
Figure 4.3: overview of the multimodal language.....	42
Figure 5.1: System architecture showing the different components and how they are related.....	44
Figure 5.2: The reasoning system.....	46
Figure 5.3: Reasoning architecture.....	47
Figure 5.4: The XML file hierarchy (Human attributes).....	49
Figure 5.5: system main packages.....	51
Figure 5.6: Class diagram of the package GUI	52
Figure 5.7: XML/IO Package class diagram	53
Figure 5.8: Sequence diagram: annotating process	54
Figure 5.9: Progress bar updating the Control and begging levels.....	55
Figure 5.10: Sequence diagram: Loading XML file.....	56
Figure 5.11: Flowchart of the annotating process.	57
Figure 5.12: global usage of the system	58
Figure 6.1: The user interface.....	65
Figure 6.2: The file menu	66
Figure 6.3: The fighting level is above 80. The text “fighting” flash white and red	67
Figure 7.1: Testing in the MMI lab (the observer right, the tester left).....	75
Figure 7.1: Fighting level is flashing.....	82

List of tables

Table 2.1: Distribution of the correct recognition ratio.....	15
Table 3.1: Passengers that give a score of 7 or higher for the safety in the train	17
Table 3.2: unsafely feeling in public transportation.....	18
Table 3.3: Average travel by train.....	18
Table 3.4: Static objects that can be found on a train compartment.....	21
Table 3.5: Objects that are brought by passengers	21
Table 3.6: People on the train.....	22
Table 3.7: Description of the user actions.....	27
Table 4.1: The different alternatives weak/strong points	34
Table 5.1: Description of the human fact list	49
Table 5.2: Description of the activity fact list	50
Table 5.3: Description of the objects fact list.....	50
Table 5.4: Description of the relation fact list.....	50
Table 6.1: A description of the user interface.	65
Table 7.1: Description of the usability measurements	76
Table 7.2: Test subjects characteristics	80
Table 7.3: Top three characteristics.....	81

Introduction

Aggression is a behavior that is intended to threaten or inflict physical injuries on another person or organism; a broader definition may include such categories as verbal attack, discriminatory behavior, and economic exploitation [1]. To increase passengers safely from aggression in trains, the Dutch railways (NS) and the Man-Machine Interaction research group at Delft University of Technology started the Aggression detection project. The aggression detection project focuses on developing a system that detects human aggression using a multi-modal system that fuses sound and video signals, recorded in train compartments.

Image processing techniques are ongoing researches, especially the semantic interpretation of images i.e. the automated recognition of facial expressions or object recognition. We want to investigate the fusion between video and audio data, to analyse the features that contribute aggression. We want to investigate/ acquire knowledge / see how an expert detects aggression. To investigate the process of security experts, we created an acquisition tool.

In this thesis, we will research the possibility of designing and implementing a system with high usability standards to annotate the actual modal in the train compartments, on the other hand, we will also implement an rule-base expert system to handle the incoming data from the annotating process, and the output will give an aggression level.

1.1 Relevance

Surveillance is monitoring the behavior or activities of people. Surveillance cameras (figure 1.1) are placed in now days almost in every shop, school, hospital and other public places. Aggression detection using the video and audio output stream of surveillance cameras is possible by watching the monitors.

Automatic aggression detection is very young field and from the larger perspective it is just starting up, however, if significant gains can be made in performance, accuracy of detection then a bright future lies ahead in the sense of less incidents and injuries due to aggression.



Figure 1.1: Surveillance camera

Numerous examples can be given where aggression detection can be used to increase the safety. For example, using aggression detection in a football stadium or other sport stadium, where aggression by hooligans may occur. Another example is in music or huge pop/rock events where drunk people walk around. As for now, the focus is on aggression detection in a train environment.

The bottleneck in today's systems is that people have to watch the monitors of the security surveillance cameras all day long. And sometimes these are ten or twenty monitors and the human is not able to track everything that happens on all monitors.

Automatic aggression detection advances can be made in every aspect of this multi-disciplinary field, from the very start (detection of objects, detection of

groups), to a more advanced human, sound tracking and facial, speech recognition, the expert system itself and the rules for detecting the aggression.

1.2 Problem definition

The Dutch railways (NS) and the Man-Machine Interaction research group at Delft University of Technology started a project of aggression detection in 2005. The current human, objects tracking and recognition techniques are not very accurate and fully automatic in the sense that one can depend and trust their output. Therefore we need a system to annotate the situation on a train compartment. Even if those techniques are trustable and accurate there are still some questions:

- Ø *What will we do with all the processed data from the multimodal devices?*
- Ø *How to annotate a certain situation in a train compartment?*
- Ø *What to annotate?*
- Ø *What is the aggression level?*

1.3 Goals

The main goal for this project is to design and implement a semi-automatic annotation system for a train environment. A secondary goal is to design and implement an expert system to detect the level of aggression after annotating a certain situation in a train compartment.

A summary of the goals is listed below:

- Research the requirements for the system.
- Design annotation system for a train environment.
- Design the user interface.
- Implement an annotation system for a train environment.
- Implement an expert system to detect the aggression level.
- Test the usability of the system.
- Evaluate the system.

This leads us to formulate the main goal of the project:

To implement an annotation system for multimodal video / recordings of aggression scenarios for a train environment and to return a certain aggression level as an output.

1.4 Approach

We divided the project into phases.

1- The first phase of the project is to research the current techniques to annotate a situation in a certain environment. The goal for this phase is to collect enough information so we will be able to decide how we will implement our system.

2- The second phase is analyzing all the data we have, we watched all the recorded video's of the scenarios that were made by the MMI group. The scenarios were predefined and were acted in a real train compartment. Notes about the objects and humans relations that can be found in a train compartment were made. After that the design of the annotating language is started. This phase is necessary to do before the actual design of the system, since we need to know the syntax of the language, we also need to know what will be annotated and how.

3- The third phase is the actual design of the system, using UML to make class diagrams and sequence diagrams to understand the system.

4- Phase four is the implementation of the system. We chose Java as a programming language and Jess as an expert system and XML to store the annotations that were made by the users. After the implementation, we began to test the system. We asked students to test the usability of the system. The testers had to fill a debriefing form to tell us what they think about the system, and from there we analyzed the data and concluded the testing results and the recommendations to improving the system.

The final result is a system the user can use to annotate a loaded sequence of images captured from a video file. After annotating, the user will get as a result

an aggression level and aggression classification. The annotation can be stored and loaded to/from an XML file.

1.5 Thesis overview

The thesis report consists of eight chapters. The first chapter gives an introduction about the project and the problem definition that needs to be solved. Chapter 2 provides background information about human aggression and especially human aggression on train compartments. Chapter 3 defines the problem and lists the requirements. Chapter 4 contains an explanation of why we need an annotation language for aggression detection in train's compartments, and why we chose the use of CLIPS and a rule-base approach. Chapter 5 presents the design of system using UML models. First the global system design is discussed, followed by a detailed overview of the design of the main classes that will be used. Finally, the user interfaces is discussed. Chapter 6 describes the actual implementation of the system. An overview of the used software and tools is given and afterwards the implementation of expert system is discussed and finally some screenshots of the GUI will be shown. Chapter 7 describes the test plan and presents the testing results of the implemented system. First an explanation about how the data was collected, thereafter the test plan will be described and finally the experiments results will be presented. In Chapter 8 the results of the project will be assessed against the requirements and goals that were defined at the start and the results of the implemented system of the train aggression project will be discussed. Furthermore, recommendations for future work will be given.

2

Background

This chapter provides background information about human aggression in general, and the aggression types. After that we will discuss about the aggression in train. Then a discussion about the related projects that concerns aggression, annotating, usability, and expert systems.

2.1 Human aggression

Aggression is a perplexing phenomenon. Because in some situations it is unknown of why are people motivated to hurt each other. How does violence help organisms to survive and reproduce? After two centuries of theories [21] and technological advances, psychologists and other scientists have been able to look deeply into aggression's biological and evolutionary roots, as well as its consequences in society.

2.1.1 Definition

Aggression is defined as physical or verbal behavior intended to harm. Aggression can either be directed inward by self-mutilation or suicide, or directed outwardly at another person. There are many things that human aggression has been caused on, including broken homes, poverty, chemical imbalances in the brain, toy guns, inequality, TV violence, sexual repression, sexual freedom, and bad genes [2]. Some believe that all of these potential causes have one thing in common: unfulfilled human needs and desires. Fortunately, when most people's needs are not met they do not turn to violence to deal with their frustrations.

Nevertheless, self-control sometimes breaks down, resulting in aggression ranging from petty theft to murder.

One of the problems in researching aggression is how to classify the different type of aggression. It is an easier task with animals, which tend to display stereotyped patterns of violence such as killing to gain food or territory. With humans and non-human primates, classifying aggression becomes more difficult because there is complication of intent. Punishment, for example, represents an especially gray area. Should spanking be considered an aggressive act? What about capital punishment? Indeed, almost all acts we consider aggressive have been socially sanctioned by some cultures over the years. Moyer (1968) [14] presented an early and influential classification of seven different forms of aggression, from a biological and evolutionary point of view:

- Predatory aggression: attack on prey by a predator.
- Inter-male aggression: competition between males of the same species over access to resources such as females, dominance, status, etc.
- Fear-induced aggression: aggression associated with attempts to flee from a threat.
- Irritable aggression: aggression induced by frustration and directed against an available target.
- Territorial aggression: defense of a fixed area against intruders, typically conspecifics.
- Maternal aggression: a female's aggression to protect her offspring from a threat. Paternal aggression also exists.
- Instrumental aggression: aggression directed towards obtaining some goal, considered to be a learned response to a situation.

Currently, there is a consensus for at least two broad categories of aggression, variously known as hostile, affective, or retaliatory aggression, versus instrumental, predatory, or goal-oriented aggression. Empirical research indicates that this is a critical difference, both psychologically and physiologically. Some

research indicates that people with tendencies toward affective aggression have lower IQs than those with tendencies toward predatory aggression [15].

As can be expected, humans express their aggression differently than animals. Although humans are similar to animals in some aspects of aggression, they differ from most animals in the complexity of their aggression because of factors such as culture, morals, and social situations. A wide variety of studies have been done on these situations. Alcohol, drugs, pain and discomfort, frustration, and violence in the media are just a few of the factors that influence aggression in humans.

In this project we will focus on instrumental aggression, since it is a planned, goal directed aggression. Also, this form of aggression is known to be the most common aggression that may occur on a train.

2.1.2 Instrumental aggression

The two cardinal characteristics of instrumental aggression are goal-directedness and planning. The instrumental aggressor acts to obtain a readily apparent goal such as power, money, sexual gratification, or some other objective beyond inflicting injury on the victim. Examples of instrumental aggression include shooting a police officer in the course of a bank robbery, stabbing a homeowner during a burglary, and strangling a rape victim. Instrumental aggression is initiated as a means to an end rather than as an act of retaliation or self-defense. As a result, instrumental aggression involves planning or preparation. However, in some cases instrumental aggression involves relatively little planning, such as in the case of a criminal who engages in an opportunistic offense (e.g., unexpected opportunity to rob someone that involves assaulting the victim). In some cases, a subject may plan a robbery or burglary, and when something goes wrong, engages in an act of aggression, such as shooting someone in order to get away. In these cases the coder should consider that the subject's plans include the possibility of violence, even if there was no specific plan to shoot someone.

Instrumental aggression usually involves little or no provocation by the victim. In some cases subjects may be "provoked" into violence in the course of another

crime, e.g., a robbery victim who insults the subject or resists the robbery in some way. These acts are still considered instrumental acts of aggression.

Instrumental aggressors are motivated by goals, not emotions. It follows that their level of emotional arousal, especially anger, is relatively low or is secondary to the act. Some instrumental aggressors try to calm themselves prior to an offense through drug use or drinking. In extreme cases, instrumental aggressors are not angry toward their victims and may have a cold, "business-like" attitude about their behavior. Nevertheless, many less hardened instrumental aggressors are nervous and highly aroused while committing a crime, even though it is not their arousal which motivates their actions.

2.1.3 Biological & Social research

Reading a daily newspaper, shows us how serious the problem of violence is in today's society. There is an 80% chance that a person will be the victim of a violent crime during his or her lifetime [3].

Researchers claim that we are coming closer to predicting from a brain scan or a blood test whether a person is at risk for committing an act of violence. Ethical complications aside, a closer look at the neurobiology of aggression shows why we are unlikely to find a conclusive test for potential violent behavior. While there are many biological factors associated with aggression, their predictive value remains still quite low.

The area from which all emotion originates is the brain. While scientists continue to test various areas of the brain for their effects on aggression, two areas that directly regulate or affect aggression have been found. The amygdale has been shown to be an area that causes aggression. Stimulation of the amygdale, results in augmented aggressive behavior (Bauman et al 2006) [16]. Another area, the hypothalamus figure 2.1, is believed to serve a regulatory role in aggression. The hypothalamus has been shown to cause aggressive behavior when electrically stimulated but more importantly has receptors that help determine aggression levels based on their interactions with the neurotransmitters serotonin and vasopressin.

The hypothalamus and pituitary gland are important parts of the brain's limbic system associated with emotional response and arousal. These structures, along with the septum and amygdala, may play a role in mediating aggression. [3].

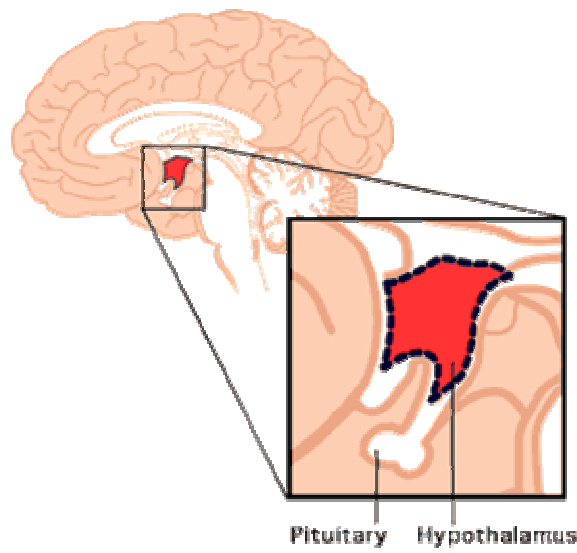


Figure 2.1: The location of the human Hypothalamus in the brain

On the other side, Craig A. Anderson and L. Rowell Huesmann [17] tried to understand the human aggression in a social-cognitive view. They researched other human aggression model that has been developed in the last 30 years, they tried to identify what are the causes of the aggression and in which environments they may occur. They used the general human aggression model figure 2.2, which illustrates the point that all of the social-cognitive models agree that aggression results from the way in which person variables and current situational variables combine to influence the individual's present internal state. Sometimes person and situation variables combine interactively, as in K.B. Anderson et al.'s (1998) finding the pain and the trait hostility interactively affect aggressive cognitions. The present internal state then influences a host of appraisal and decision processes. Eventually (sometimes very quickly), an action emerges, which in turn moves the social encounter along to its next cycle.

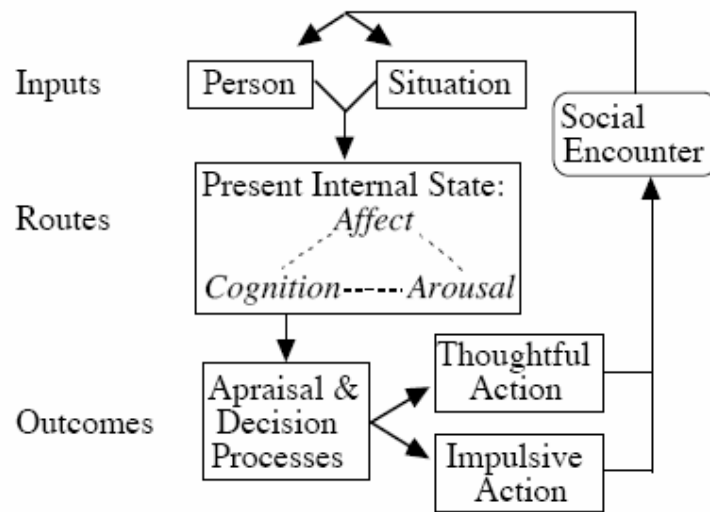


Figure 2.2: the general aggression model episodic processes

They concluded that the current human aggression models coverage on a common set of theoretical assumptions, largely because those assumptions have proved useful in organizing the vast research literature and in generating new, testable hypotheses concerning the development, persistence, and change of aggression.

2.2 Aggression in train

The aggression in train compartments can take many different forms. It can be directed towards the interior of the train, the passengers, conductors and train mechanist. The interior can be damaged or painted, and sometimes burned, or written on it. Passengers, conductors and train mechanist can be robbed, threatened, intimidated or physically or mentally hurt. The train aggression is causing a decrease in the number of conductors.

The number of intimidating and threatens towards the conductors is growing. In the first 8 months of 2003, the number of aggression cases was more then 15% than a year earlier in the Netherland. The statistics are showing that the aggression problems are getting worse, in 2001 there were only 6.944 incidents and in 2006 that number was 10.500. The aggression is directed in 75% of the cases towards the conductors and within that 75%, 1 out of 10 incidents includes

physical violence [4]. About 10% the aggression is directed towards the passengers, 67% of the total aggression is in the form of threatening or swearing. The Dutch railroad needs at least 10% more conductors and they can't achieve that by not doing anything to the aggression on the trains. NS tried to solve this problem by hiring guards as conductor assistant during rush hours.

The train interior damage is costing the Dutch railroad about 20 million euros on a yearly basis, according to Mr. Schultz van Haegen the state secretary at the Dutch ministry of Water and Traffic. The damage is mainly caused by vandalism and graffiti that are drawn on the interior of the train. The eradication costs of these graffiti cost about 10 million euros. The misuse of the fire extinguishers and the cleaning of the damage afterwards are costing the NS about 5 million [6].

2.3 The Current state of the art

This part provides a list of some studies about human aggression, annotating tools, rule-base and expert systems. Below the topics are discussed into chapters.

2.3.1 Automated aggression detection

Video Annotation for Content-based Retrieval using Human Behavior Analysis and Domain Knowledge

This paper [18] proposes an automatic annotation method of sports video for content-based retrieval. Conventional methods using position information of objects such as locus, relative positions, their transitions, etc. as indices have drawbacks that tracking errors of a certain object due to occlusions cause recognition failures, and that representation by position information essentially has limited number of recognizable events in the retrieval.

The authors approach incorporates human behavior analysis and specific domain knowledge with conventional methods, to develop integrated reasoning module for richer expressiveness of events and robust recognition. Based on the proposed method, they implemented content-based retrieval system which can identify several actions on real tennis video.

2.3.2 A Rule-Based Video Annotation System

A generic system for automatic annotation of videos is introduced [19]. The proposed approach is based on the premise that the rules needed to infer a set of high-level concepts from low level descriptors cannot be defined a priori. Rather, knowledge embedded in the database and interaction with an expert user is exploited to enable system learning. Underpinning the system at the implementation level is pre-annotated data that dynamically creates signification links between a set of low-level features extracted directly from the video dataset and high-level semantic concepts defined in the lexicon. The lexicon may consist of words, icons, or any set of symbols that convey the meaning to the user. Thus, the lexicon is contingent on the user, application, time, and the entire context of the annotation process. The main system modules use fuzzy logic and rule mining techniques to approximate human-like reasoning. A rule-knowledge base is created on a small sample selected by the expert user during the learning phase. Using this rule-knowledge base, the system automatically assigns keywords from the lexicon to non-annotated video clips in the database. Using common low-level video representations, the system performance was assessed on a database containing hundreds of broadcasting videos. The experimental evaluation showed robust and high annotation accuracy. The system architecture offers straightforward expansion to relevance feedback and autonomous learning capabilities.

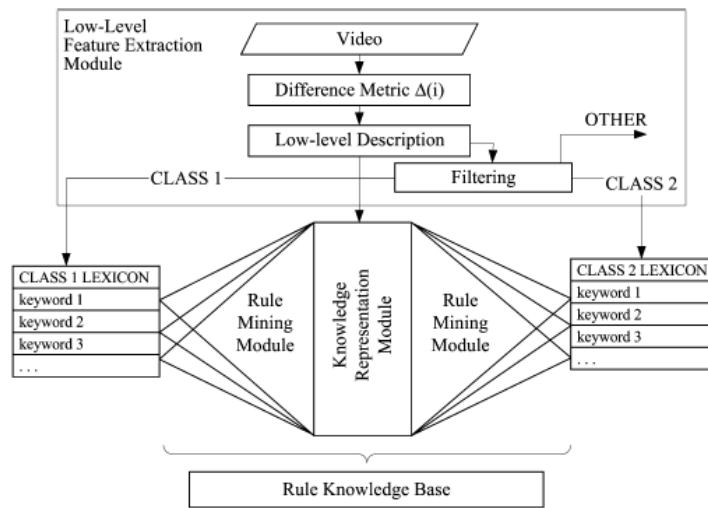


Figure 2.3: Flowchart of the learning unit

2.3.3 Expert system for automatic analysis of facial expressions

This paper discusses an expert system that was made by the authors called Integrated System for Facial Expression Recognition (ISFER), which performs recognition and emotional classification of human facial expression from a still full-face image [20]. The system consists of two major parts. The first one is the ISFER Workbench, which forms a framework for hybrid facial feature detection. Multiple feature detection techniques are applied in parallel. The redundant information is used to define unambiguous face geometry containing no missing or highly inaccurate data. The second part of the system is its inference engine called HERCULES, which converts low level face geometry into high level facial actions, and then this into highest level weighted emotion labels.

Table 2.1: Distribution of the correct recognition ratio

Produced expression	Classified as	Produced expression	Classified as	Produced expression	Classified as	Produced expression	Classified as
1 + 2	95% surprise	1	98% sadness	23 + 17	72% anger	10 + 17	75% disgust
2	77% anger	4	100% anger	23 + 26	49% anger	10 + (25/26)	67% disgust
6	98% happiness	5	79% surprise	23	88% anger	10	82% disgust
1 + 4 + 5 + 7	90% fear	7	92% anger	24 + 17 + 26	85% anger	9 + (25/26)	93% disgust
1 + 4 + 5	93% fear			24 + 17	77% anger	9 + 17	95% disgust
1 + 4 + 7	82% sadness	27	81% surprise	24 + 26	53% anger	9	89% disgust
1 + 5 + 7	96% fear	20 + (25/26)	100% fear	24	92% anger	12 + (25/26)	100% happiness
1 + 4	79% sadness	20	80 % fear	10 + 16 + (25/26)	75% anger	12	100% happiness
1 + 5	81% fear	15 + (25/26)	100% sadness	10 + 17 + (25/26)	79% disgust	16 + (25/26)	68% anger
1 + 7	77% sadness	15	100% sadness	9 + 17 + (25/26)	98% disgust	17	83% sadness
5 + 7	86% fear	23 + 17 + 26	82% anger	12 + 16 + (25/26)	100% happiness	26	70% surprise
							Average: 85.02%

We can use a similar table to classify the aggression in this project.

3

Problem analysis

The problem in the train environment is that there are two types of objects, static and dynamic. In this chapter, first we will be discussing some statistical information about the aggression in trains. Then we will give a description of the problem of the train environment, the static and dynamic objects and activities will be listed. Thereafter the requirements will be discussed.

3.1 Aggression in the train.

It is important to know how much aggression occurs in the public transportation and especially in trains. The NS year report [12] shows that the passengers who give a score of 7 or higher to the safety in the train are 69% as average over the last 5 years Table 3.1. This means that 31% of the passengers do not feel safe in the train, and this is due to aggression.

Table 3.1: Passengers that give a score of 7 or higher for the safety in the train

% passenger that give 7/10 or higher for safety	2002	2003	2004	2005	2006
Social safety in trains	63	66	69	71	74

In addition, table 3.2 (source the CBS [13]) shows that about 7.2% of the total population in The Netherland do not feel safe in public transportation, once again due to aggression that is made.

Table 3.2: unsafely feeling in public transportation

Subject	Feels occasionally unsafely	Feels often unsafely	Unsafely feelings situations				
			Public transport	Public places	In own house	On the street	In the mall
Netherland	27%	4.8%	7.2%	13.5	3.5	9.4	7.7

These numbers show us that people do not like to travel by trains or other public transportations. However, not everyone has any other option, and from the total population we can see in table 3.3 that the average distance traveled per person per day using the train is about 2.45 km.

Table 3.3: Average travel by train

Subject	Travel per person per day	Distance per Travel	Travel distance per person per day
Number	0.06	42.65 km	2.45 km

3.2 Aggression detection by humans

To understand what is needed to detect aggression automatically, it is important to understand how human detect an aggressive situation. Humans are very well able to detect aggression in verbal expressions. The problem here is that speech recognition techniques are still ongoing researches, and there is no technique available that provides enough accuracy to depend on given the large amount of noise occurring in the train. Therefore, we need to identify other aspects that humans use to detect aggression e.g. emotions, shouting and location. Humans detect aggression easily when he/she knows the environment, e.g. when a human sees two people fighting in a karate school, he/she knows this is part of the sport and its probably not a form of aggression, but when he/she sees two people fighting on a train, then this means it is a form of aggression, another example is when he/she sees someone smoking on the street he/she does not think about

aggression, and that is different when someone is smoking in the train, this might lead to aggression situation since smoking is not allowed on the train. These examples imply that the environment where the aggressive or non aggressive situation arises is very important.

The human does not only use the environment to detect aggression, the human detect objects such as a knife, but simply detecting a knife is not enough, e.g. a person holding a knife is not enough, the human identify the activity the person holding the knife, the person could be holding a knife to peel an apple which isn't an aggressive situation, but holding a knife when the person is angry is a different situation. In the examples above we used the word activity and we refer it to peeling an apple, this means that the human have to know the activity to detect aggression. Also, the word angry is used which is an emotion.

Another example of how human detect aggression is when a human sees a person holding a gun, then he does not immediately sees that as an aggressive situation, e.g. a police officer with a gun is not an aggressive situation, in fact it is a safe situation since people tend not to do an aggressive acts when a police officer is around. Although a passenger or a hooligan holding a gun is referd to be very aggressive situation. This implies that knowing the people plays an important role in the human aggression detection.

Another important issue is that humans have memory, and they can easily memorize the changes in time e.g. a person was drinking beer ten minutes ago in the train and they saw him shouting at others, later they see that person again and sees him walking towards them, they immediately remember him and they detect a possible aggression.

Thus, the human reasons about aggressive situations using the knowledge of the environment, the people types, objects, the activity, the emotional state of the person doing the act of aggression and finally they use their memory to identify people [46]. A computer system needs the same knowledge about the environment to reason about aggression. For that reason a world model is needed. The context of that world model should be the same set of information the human

uses to detect aggression, which is people types, activity, objects, relations and emotions.

3.3 An analysis of the train compartment

The environment we will be focusing on is the train compartment. In this environment we have specific objects such as seats, pathways, windows, doors and objects that are brought by humans, and we also have different type of humans that can be found in a train. Humans such as normal passengers, conductors, police, hooligans, and beggars play a specific role e.g. traveler, controller, and information provider. Occasionally unwanted behavior arises for instance, aggressive behavior towards conductors or other passengers, demolition of objects. Our goal is to design an aggression detecting system by multi-modal camera's which are able to observe their environment, communicate with the main computer and reason about the observed data, and conclude an aggression level as an output.

To reason about the observed data in the environment, a reasoning model have to be chosen.

We can choose between, an expert system, Bayesian networks and neural networks. An expert system, also known as a knowledge based system, a computer program that contains some of the subject-specific knowledge, and contains the knowledge and analytical skills of one or more human experts. A Bayesian network (or a belief network) is a probabilistic graphical model that represents a set of variables and their probabilistic independencies. An artificial neural network (ANN), often just called a "neural network" (NN), a mathematical model or computational model based on Biological neural networks. It consists of an interconnected group of artificial neurons and process information using a connectionist approach to computation. In this project we chose to use a rule-base expert system as a reasoning model, for the reason that we can represent the situation in a train compartment and put the facts in the knowledge base by annotating the situation.

The expert system needs a model for the representation of the situation in the train compartment. Therefore, we have to define which objects can be found that play a role in aggression. The world is dynamic, this means that objects can appear in one certain moment, and disappear in another. As a result the reasoning mechanism has to use a database to store all objects, people and relations in a database, and of course the time where objects and people appear or disappear will be stored as well.

3.4 An analysis of the train compartment: The static context

In order to represent the environment to the expert system, we must identify the objects within the static environment that can play a role in aggression. The static objects are objects that are always in the train compartment. Table 3.4 provides a list of the static objects.

Table 3.4: Static objects that can be found on a train compartment

Object	Description
Train seat	Seats are found in the hallway and in the compartment. They may be solid or folding benches. Aggressive use: drawing upon, putting feet on, occupying more then one seat, and putting objects on the seat when there are not many seats available.
Train table	Found in compartments, on the wall or on the back of a bench, where it can be pulled out. Aggressive use: drawing upon, mutilation and folding in aggressive manner.

3.5 An analysis of the train compartment: The dynamic context

The dynamic objects are objects that are moving, in other words, objects that are not always on a train compartment. The dynamic objects in this environment can be split into two categories, the first one is objects that are brought by people onto the train table 3.5, and the second one is the people who are on the train table 3.6.

Table 3.5: Objects that are brought by passengers

Object	Description
Bag	Bags should be stored on people's laps, in the luggage rack or under the benches, where sometimes a storing room is provided. Putting

	bags on a bench occupies seating space and may be considered an act of aggression when there are not many seating space available, causing people to travel standing up. Aggressive use: keeping on seats, throwing, or touching other passengers with.
Cigarette	It is not allowed to smoke in the train, and this might create an aggressive situation when someone is smoking. Aggressive use: smoking when it's not allowed.
Beer can or beer bottle	Drinking beer is allowed in the train. Aggression situation may occur when someone gets drunk. Aggressive use: getting drunk, throwing at another passengers.
Book	These can be put on the table, on benches or on the luggage rack. Aggressive use: putting it on fire, reading in a way that hinders others, throwing at others.
Baby pram	A pram will take up seating space or stand in the way in the gang way, but clearly there is no way around this. Aggressive use: putting on seats, throwing at others, touching others.
Knife	Knives are most of the times hidden by the passengers. Aggression situation may occur when someone is holding one. Aggressive use: hitting others with it, threatening others.
Tickets	Tickets are used as transportation prove. Aggression situation may occur when someone does not have a ticket.
Cell phone	Phones can be put on the table or in the carrier's clothing or bag. Some people get irritated by hearing other people conversing over the phone loudly. Aggression use: Throwing, Loud conversation.
Doll	Dolls are brought by children. The only aggressive situation that may occur is when a child throws the doll at someone. Aggressive use: Throwing.
Mp3-player	Wearable audio can be put on the table or listener's lap or in his pockets. Turning the value up too loud may irritates others. Aggressive use: Storing elsewhere, Turning the volume up too loud and throwing.
Money	Money is stored in wallets or in the carrier's clothing. Aggressive situation may occur when someone tries to steal the money. Aggressive use: Stealing from the carrier.
Paint	Painting or writing on the train interior may cause aggressive situation. Aggressive use: Painting on the train interior.

Table 3.6: People on the train

People	Description
Conductor	One who is in charge of a railroad train. A conductor checks the tickets of the passengers and provides information's.
Passenger	A person who travels using the train, without participating in its operation.
Beggar	One who solicits alms for a living.
Police officer	A member of a law-enforcement agency.
Artist	A person whose work shows exceptional creative ability or skill. In this project he asks for money after his performance.
Hooligan	A person who treats others violently and roughly. Mostly a soccer club fan.

3.6 An analysis of the train compartment: Activities & behaviors

We analyzed the actions that can be made on a train, and we found: Sitting, smoking, talking, walking, hitting, checking tickets, putting, holding, shouting, and putting legs on the seat. Combinations between these actions and objects that are found, aggression situation can be created, like:

A hooligan is holding a knife, < very unsafe situation >

A hooligan is holding paint, < damaging the interior >

A passenger is smoking a cigarette, < may cause a fight >

Etc...

Emotions also play an important role in aggression detection. Emotion, in its most general definition, is a psychophysical process that arises spontaneously, rather than through conscious effort, and evokes either a positive or negative psychological response and physical expressions, often involuntary, related to feelings, perceptions or beliefs about elements, objects or relations between them, in reality or in the imagination [24] e.g.:

Intimidating aggression \mathbb{E} Anger

While

Aggression \mathbb{E} Fear

We need human emotions in the system to reason more accurately about aggression. Ekman [18] categorized 6 basic emotions: anger, disgust, fear, joy, sadness and surprise. Automatic recognition is rapidly becoming an area of intense interest in the research field of machine vision. Several projects are trying to develop the best possible rate of recognition. M. Panic and L. M. Rothkrantz [25] presented an automated system that they developed to recognize facial gestures in static, frontal- and/or profile-view color face images. A multi-detector approach to facial feature localization is utilized to spatially sample the profile contour and the contours of the facial components such as the eyes and the mouth. From the extracted contours of the facial features, they extract ten profile-contour fiducial points and 19 fiducial points of the contours of the facial components. Based on these, 32 individual facial muscle actions (AUs) occurring

alone or in combination are recognized using rule-based reasoning. With each scored AU, the utilized algorithm associates a factor denoting the certainty with which the pertinent AU has been scored. A recognition rate of 86% is achieved.

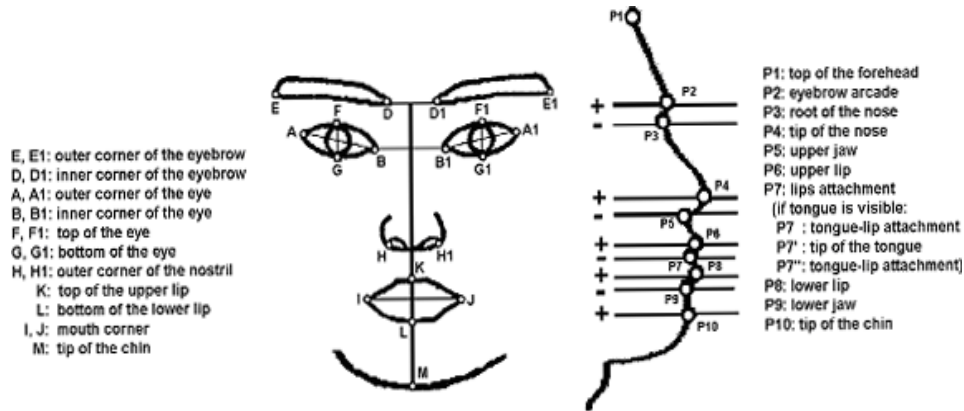


Figure 3.1: Feature points (fiducials) of the contours of the face components and of the profile contour)

The aggression detection project is a large project, it is divided in sub projects and the automated facial recognition part is not part of this sub project, However, because we want to take the next step in this project and take a look at what we will do when we have the emotion recognition ready, we will use emotions as if we have them. We used only five of Ekman six basic emotions which are: anger, fear, joy, sadness and surprise. We didn't use disgust since it's not common on a train compartment.

Body language also plays a major and important role in aggression detection. Body language is a term for communication using body movements or gestures instead of, or in addition to, sounds, verbal language or other communications. Neil Robertson and Ian Reid [26] developed a system for human behavior recognition in video sequences. Actions are described by a feature vector comprising both trajectory information (position and velocity), and a set of local motion descriptors. Action recognition is achieved via probabilistic search of image feature databases representing previously seen actions. Hidden Markov Models (HMM) which encode scene rules are used to smooth sequences of actions.



Figure 3.2: Exemplar set for walking

High-level behavior recognition is achieved by computing the likelihood that a set of predefined HMMs explains the current action sequence.

Thus, human actions and behavior are represented using a hierarchy of abstraction: from person-centered actions, to actions with spatio-temporal context, to action sequences and finally, general behaviors. While the upper levels all use Bayesian networks and belief propagation, the lowest level uses non-parametric sampling from a previously learned database of actions. The combined method represents a general framework for human behavior modeling.

The use of body language is not standardized globally, but locally – such as to a particular country, continent or region. The use of body language is one of the most variable forms of expression in the world, just as spoken and written languages are [27]. Take insults, for example, forms of communication that say, “in your face, jerk!” Aussies (citizens of Australia) often extend the thumb as if to say yes, except as if to mean to say “you idiot!” Meanwhile, in North America and Europe, they give offensive people the middle finger because it looks like a certain male body part when formed with our hands. In other cultures it may insult people with the finger-thumb zero sign (“A-OK”), a v-shape formed with the first two fingers after the thumb (called the “victory sign”) or the index finger and the small little finger all the way back from the thumb.

Another common gesture people can mix up in different countries is “he is acting like he is crazy” motion. In North America and Europe, they point to their heads and swirl their fingers clockwise to indicate such inferior unintelligent behavior. If you do that in Japan, however, you make it look to locals as if you mean otherwise (as if he is thinking intelligently). However, reversing the motion does indicate to a man native to, say, Tokyo or Nagasaki, that he should correct his behavior to correct standards. The point is clear: different cultures call for

different body language standards. We choose body language gestures that are used in Europe: Giving the middle finger, fighting gesture (attack signals, exposing oneself, insulting gestures) and “no ticket” gestures (searching pockets, knocking the head “No”, running away from the conductor).

Finally, since we have many people on the train compartments, we have relations between people in the train: shouting at, throwing an object at, hitting, touching, and invading one’s private space.

3.7 An analysis of the system needed

One of the main goals of the system is to annotate the situation on the train compartment. This is why we need a system to make that wish come true. A good way to start the analysis about what is needed in the system is to sketch what a typical annotating session will look like.

The user needs to see the video file he will annotate, this means that the system must be able to open a video file. Since the annotation will be on static images (not moving video) the user must be able to load a set of frames (pictures) that were captured from a video file. The annotating process can be started now. During annotating the user should be able to select different kinds of people types, objects, emotions and relations. Because the train environment is dynamic and objects appear at one time and disappear at other, the user should be able to remove objects.

After that the user should be able to save the work that has been made, and because there is a save function, there must be a load function as well. When loading a file, the user should be able to play the file and watch what was annotated on each frame, and he/she should be able to pause, stop, forward and rewind between the frames.

Now that we know the important aspects of the system, a use case diagram can be drawn. A Use Case diagrams identify the functionality the system will provide, the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system. The primary goals of Use Case diagrams include:

- Providing a high-level view of what the system does.

- Identifying the users ("actors") of the system.
- Determining areas needing human-computer interfaces.

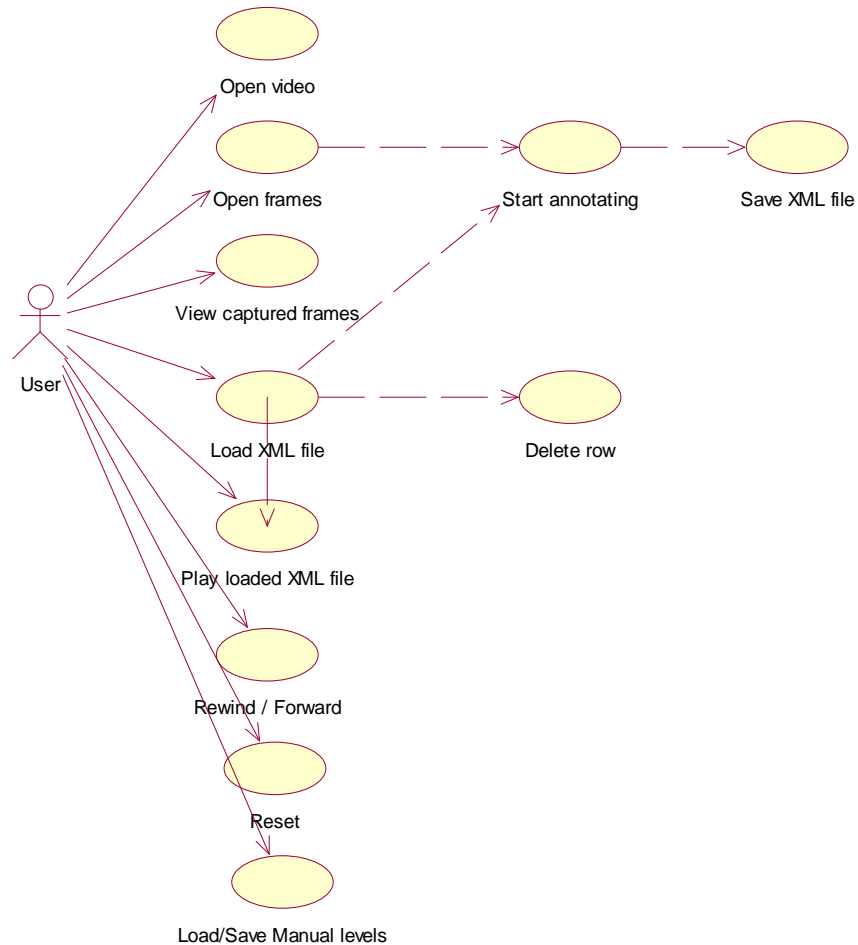


Figure 3.2: Use Case diagram

To summarize the actions that can be made, table 3.7 provides an overview of the functionality of the system.

Table 3.7: Description of the user actions

Action	Description
Open Video	The user can view a video before starting the annotating process.
Open frames	When the program starts the user have to load/open a set of frames that has been captured from a video file.

Load XML File	The user can load a saved session from a XML file.
Save XML File	The user can save a session into a XML file.
Annotating	After opening the frames or loading XML file, the user can now start annotating.
View captured frames	The user can see what is been captured while annotating the world model.
Play loaded XML file	The user can play the loaded XML file; the system will start from frame 1 to frame 10, and will show on every frame what exactly is happening.
Rewind / Forward	The user can rewind or forward a loaded XML file to see a specific frame.
Load / Save manual levels	The user can use the plus and minus button to manually decide what the aggression level on every frame, he can save these data and load it again in another session.
Reset	Reset all the annotations in the current session.
Delete row	The user can select a row from the table and delete it, when an object is disappeared from the scene.

3.8 Requirements

The goal of the system is to make it possible to the end user to manually annotate the situation on a train compartment. The annotation has to be made of a sequence of captured frames from a video file. The system should be capable to automatically classify the aggression and come up with an aggression level. Other capacities of the system are:

- The user should be able to annotate aggression as well as non aggression situations.
- The user should be able to annotate different types of humans/objects/actions that are possible on a train environment.
- Saving annotations to XML file.
- Loading annotations from XML file.
- The user should be able to load a sequence of captured frames from a video file.
- Playing a video file.
- Saving the manually edited levels to a text file.

- Loading the manually edited levels form a text file.
- The user should be able to set the aggression level manually.
- Load and saving the manual set levels.

3.8.1 End users

It is important to identify the end users that will use the system. After making interviews with the supervisors and studying the requirements of the system we found three user groups that will use the system, these are:

- Computer science or related area students, with basic knowledge of programming Java and expert systems.
- People with a personal interest in artificial intelligence that have at least basic programming skills.
- Researchers who will continue working on the project of aggression detection in a train environment.

To design a system that will be useful to these three groups of people, it is crucial that a flexible system is built that can be modified to suit personal preferences. As these groups are likely to use the system for different reasons and with different background knowledge, it is important that the requirements for all groups are incorporated in the system.

3.8.2 Functional requirements

Functional requirements define the internal work of the system, or in other words what the system will be able to do. We created a list of these requirements:

- Giving a list of annotations, the system should automatically classify the aggression.
- Giving a list of annotations, the system should automatically return an aggression level.
- The user should be able to draw a rectangle of the location of the people in the train compartment.
- The drawn rectangles have to generate JPEG pictures.
- The user should be able to see all the generated pictures.

- The user should be able to undo a wrong drawn rectangle.
- The system should be able to play a loaded XML file.
- The system should allow to pause, stop, forward or rewind the demo.
- The system should be able to load 10 frames of a video sequence.
- The system should show the aggression level on every frame individually.
- The user should be able to reset everything.

3.8.3 Non functional requirements

Non functional requirements are requirements which specify criteria that can be used to judge the operation of a system, rather than specific behaviors:

- The interface should be intuitive and easy to use.
- The programming language should be OS independent, and preferably in Java.
- The reasoning should be done using an expert system, preferably using Jess.
- The aggression classification should be the same as the expected classification within the used scenario.
- The level of aggression should be reasonable and close to the expected values within the used scenario.
- The system should be error free.
- The system should be efficient in use.
- The system should be of a high performance.

4

Annotation language for trains

This chapter contains a discussion of why we need an annotation language for aggression detection in train's compartments. And why we chose the use of CLIPS and a rule-base approach.

Afterwards we continue with a description of the language and its syntax, we also discuss the language capabilities and its future prospects. Finally an illustrative example on how the language should be used to annotate scenarios in the train compartment is given.

4.1 Annotation language

One of the goals of the aggression detection project in trains is to annotate the situation on the train compartment. To enable the use of context knowledge we have to develop a world model. In other words, we have to describe all the objects with their characterizing features which play a role in the world of aggression detection in the train compartments, therefore, an annotation language is needed. A good way to describe the world model is using Extensible Markup Language (XML) [28] or similar languages e.g. XSL Transformations (XSLT) [29]. Although, not everyone has knowledge of XML, and we cannot assume that all the users of the system know XML and can describe the world using XML.

Using speech to describe the world model would be a good choice, but the problem here is that every user will describe the world model in a different manner, and this will lead to endless amount of possibilities a user can express

the model. Therefore, using speech is too expressive for such a task and is a bad choice after all.

The use of Object Oriented (OO) to describe the world model is also a good choice, but if we use OO then it will not be easy to define new objects. And this means that OO is too restrictive for this task.

Using logic is also an option. Logic programming is, like functional programming, a declarative way of composing programs. In brief, declarative programming is much more concerned about what should be computed and much less with how something should be done. Moreover an important issue in logic programming is not only to make certain that the syntax of the code is right, but also that the semantics exactly correspond to the semantics of the problem whose solution is desired.

Table 4.1 summarizes the strong and weak points between the possibilities for the annotation language.

Table 4.1: The different alternatives weak/strong points

Approach	Problem solving	Syntax free	Clear code	declarative	Flexible
XML	-	-	++	+	++
Speech	-	++	--	+	+
OO	-	+	+	-	+
Logic	++	--	+	++	-

The above table shows that XML and Logic have the highest score (both with five +). And this leads us to a clear choice which is a mix between logic and XML approach. Logic will be used as a purely declarative representation language and a model-generator is used as the problem-solver.

4.2 Reasoning model

The simplest form of artificial intelligence (AI) which is generally used in industry is the rule-based system, also known as the expert system. Before we discuss in details what these are, let's take a step back and point out that there are different opinions as to what really constitutes artificial intelligence. Some people, when they use the term AI, are referring to systems which have some ability to learn. That is, the system will improve its performance over time as it gains experience in solving problems, just as a human would. Others, when they use the term AI, are referring just to systems which are capable of exhibiting human-level performance in some very narrow area, but which are incapable of learning or expanding their expertise.

Different people are always going to disagree about what AI is, but this is fairly simple form of AI which we want to discuss about right now. A rule-based system is a way of encoding a human expert's knowledge in a fairly narrow area into an automated system. There are a couple of advantages to doing so. One is that the human expert's knowledge then becomes available to a very large range of people. Another advantage is that if you can capture the expertise of an expert in a field, then any knowledge which they might have is not lost when they retire or leave the firm. Rule-based systems differ from standard procedural or object-oriented programs in that there is no clear order in which code executes. Instead, the knowledge of the expert is captured in a set of rules, each of which encodes a small piece of the expert's knowledge. Each rule has a left hand side and a right hand side. The left hand side contains information about certain facts and objects which must be true in order for the rule to potentially fire (that is, execute). Any rules whose left hand side match in this manner at a given time are placed on an agenda. One of the rules on the agenda is picked (there is no way of predicting which one), and right hand side is executed, and then it is removed from the agenda. The agenda is then updated (generally using a special algorithm called the Rete algorithm (Appendix C)), and new rules are selected to execute. This continues until there are no more rules on the agenda.

Another reasoning model we can use is BDI agents, BDI stands for Belief-Desire-Intention [35]. A BDI agent is a particular type of bounded rational

software agent, imbued with particular mental attitudes, viz: Beliefs, Desires and Intentions. The BDI model has some philosophical basis in the Belief-Desire-Intention theory of human practical reasoning, expounded by Michael Bratman. Wooldridge [36] lists four characteristics of intelligent agents which naturally fit the purpose and design of the BDI model:

- Ø Situated - they are embedded in their environment.
- Ø Goal directed - they have goals that they try to achieve.
- Ø Reactive - they react to changes in their environment.
- Ø Social - they can communicate with other agents (including humans).

One can say that choosing BDI agents as a reasoning system is a good choice, since it can be situated to the train environment, with the goal to detect aggression, and they can react to the dynamic changes of the train environment, and finally they can communicate with the human when an aggressive situation occurs. But what would be the beliefs of the agents? We are using an annotating system and therefore when someone annotates a situation, then these are facts and not beliefs. Desire could be to detect aggression, but what would be the desire when there will not be an aggressive situation? Then the agent will have the intention to try and search for aggression. Therefore, BDI agent reasoning system is not suited in our system, it can be used when the system can control the cameras in the train, e.g. when the agent believes there is aggression, he could move the cameras towards the aggressor with the desire to detect the aggressor.

It is clear that a rule based approach will be used to reason about the world. We developed a special language to describe the world and save the world model in XML file.

Jess [30] provides a cohesive tool for handling a wide variety of knowledge with support for rule-based programming. Rule-based programming allows knowledge to be represented as heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation.

4.3 Language syntax

Syntax refers to the sets of rules we use in constructing grammatical, understandable phrases and sentences. Syntax is generally regarded as being independent of sensibility; the sentence, "Colorless green ideas sleep furiously" (written by the famous linguist Noam Chomsky), certainly does not make any sense, but its syntax is correct.

4.3.1 Phrase structures

Every sentence is composed of some combination of phrases. In almost every language, including English, a sentence must have a noun phrase and a verb phrase. Sentences may also have object phrases that specify who did what to whom. For example, we can parse the sentence, "Mike hit the ball." The noun phrase of this sentence is "Mike," the verb phrase is "hit," and the object phrase is "the ball." Phrases can be moved around for stylistic reasons or to draw more attention to a certain part of the sentence.

A parse tree or concrete syntax tree is a tree that represents the syntactic structure of a string according to some formal grammar. A program that produces such trees is called a parser. A parse tree is made up of nodes and branches. Below (figure 4.1) is a linguistic parse tree, here representing the English sentence "Mike hit the ball". (Note: this is only one possible parse tree for this sentence; different kinds of linguistic parse trees exist.) The parse tree is the entire structure, starting from S and ending in each of the leaf nodes (Mike, hit, the, ball).

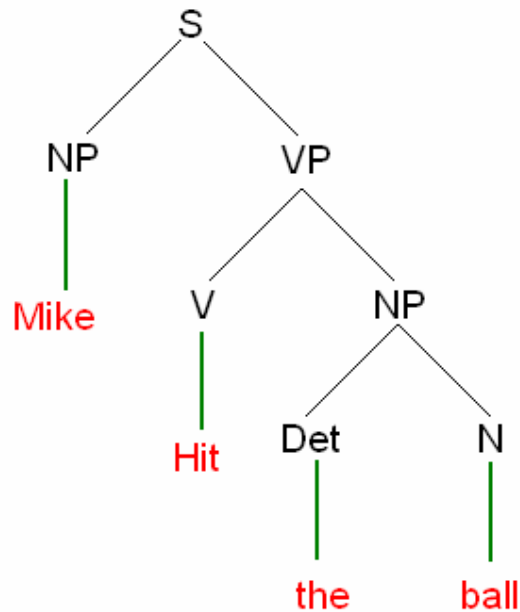


Figure 4.1: Parse tree "Mike hit the ball"

Since we don't need a language that is very complex, we limited our syntax, so it will be easier to create rules in the expert system:

<Det> <Noun> <Verb> <Object> <PP> <Det><Subject>

A passenger puts legs on the table

<Det> <Noun> <Verb> <Det> <Object>

A hooligan holding a knife

<Det> <Noun> <Verb>

A Conductor walking

<Det><Noun> <Verb> <Expression>

A beggar is angry

<Det> <Noun> <Verb> <Det> <Subject>

A passenger hits a conductor

should the user pay attention to? The users of the system have to annotate every situation that is related to aggression, or it is an aggression situation already. An

aggressive situation is when a behavior of one of the people on the train that is intended to threaten or inflict physical injuries on another person or organism.

Annotation has to be done in order to describe the situation on the train compartments. This is done to extract the features that have something to do with aggression situations. The user will annotate a sequence of frames that came from aggressive scenarios. Using the mouse, the user has to locate the location of the people involved in the scene, and use the keyboard to type in the description of the people involved.

Next, is an example Figure 4.2 of a situation where a beggar is on the train compartment begging for money from a passenger, where it end up in a fight.



 A surveillance camera shot from inside a train car. A man wearing a bright yellow t-shirt is leaning over the back of a passenger's seat. The passenger is seated and facing away from the camera. Other passengers are visible in the background, seated in rows of orange seats. A red rectangular box highlights the man in the yellow shirt.	<p>A beggar is invading private space of the passenger</p>
 A surveillance camera shot from inside a train car. A man in a yellow t-shirt and a woman in a dark jacket with a white collar are standing and facing each other. The man is holding something in his hands. A passenger is seated in the foreground, looking towards them. A red rectangular box highlights the man and the woman.	<p>A passenger is angry</p> <p>A passenger is surprised</p>

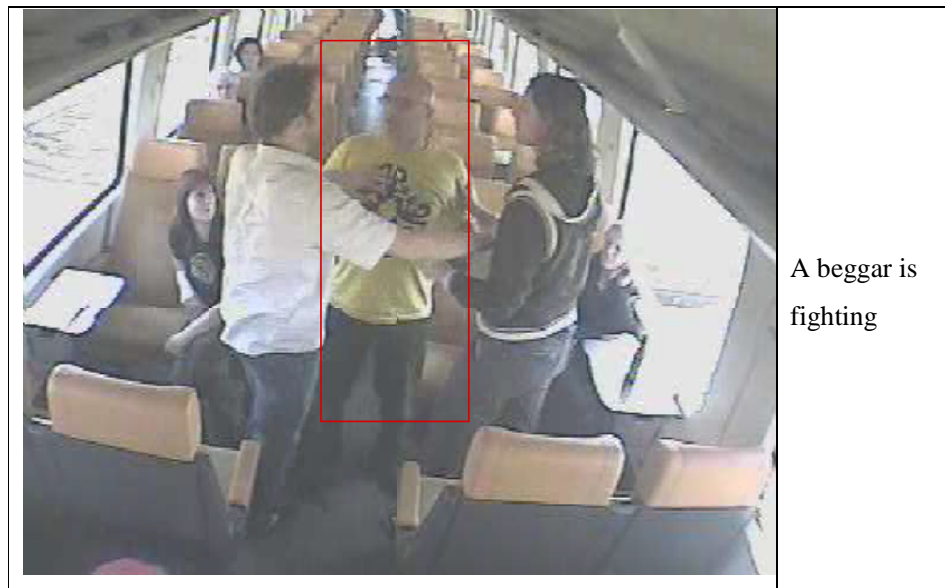


Figure 4.2: Example of annotating a beggar

As we can see, that we didn't annotate all the passengers, because they are not involved in this situation.

4.5 Future of the language

The number of sentences that can be made using this structure is very large due to the fact that we have the choice between different relations and activities between objects and people. We believe this is enough to annotate most of the common actions that can be made by people in the train compartments, because we covered a variety of scenarios which were aggressive and non aggressive scenarios, and we were able to annotate almost everything that is related to aggression. However, it is very easy to extend this language in the future. Because the use of XML. XML is easy extendable language since it is readable and well structured.

The language can also be seen as multimodal language. It can be extended in the future so it can split into: Spoken text and picture annotations. The picture annotation can also be split into: Features (e.g. movement), and semantic interpretation (running away, walking)(figure 4.3).

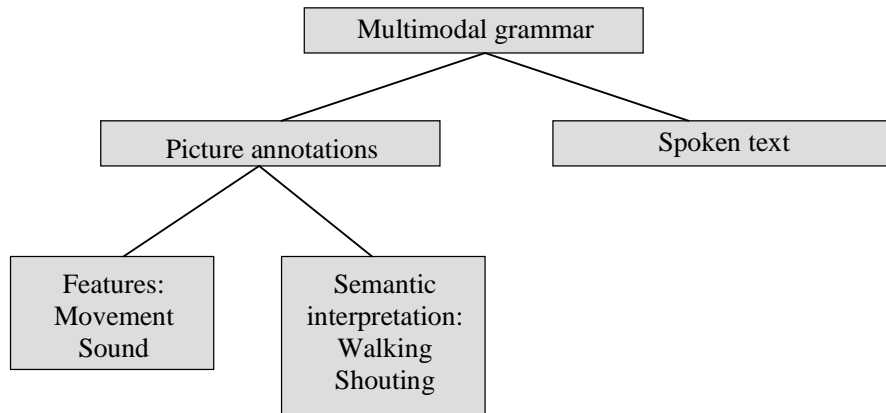


Figure 4.3: overview of the multimodal language

5

Design

This chapter presents the design of the video content analysis & aggression detection system for a train environment. First, the architecture of the system is given, followed by a detailed overview of the reasoning model. Then we will discuss the inputs and outputs of the system. Subsequently the user interface is discussed. Finally the program flow is provided.

5.1 Architecture

The system has four main components (figure 5.1):

1. GUI
2. Validator
3. Reasoning
4. XML/IO

The GUI allows the user to interact with the system. All the user commands will go through the GUI. The user interface contains facilities for the user to perform all the actions that were shown in the use-case diagram in chapter 3. The Validator validates the annotations resulting from user actions. Only valid annotations are allowed to the reasoning component, invalid annotations are rejected with a message. The Validator sends the valid annotations as facts into to the reasoning part, where Jess will reason about the aggression and send the output back to the GUI, where the output is displayed. The XML/IO is used for the saving and loading the XML file. It is also used for generating JPEG images of the locations of the people or objects as specified by the annotations. Playing a

video file of the scenario to be annotated is also the work of the XML/IO component.

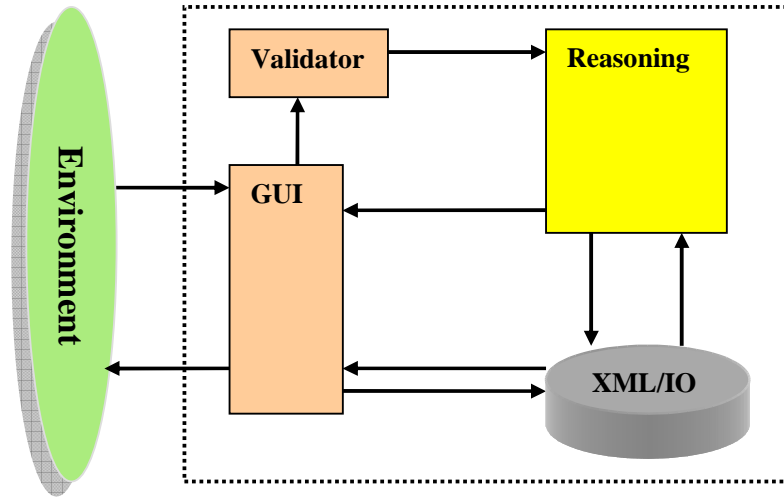


Figure 5.1: System architecture showing the different components and how they are related

5.2 The GUI component

The GUI component contains classes for the user interface. The design is based on use cases, where we defined what the user interaction with the system is, and what functionalities the system should provide. We try to keep the user interface design simple and easy to use so that users quickly see the breadth of their options, grasp how to achieve their goals, and do their work. Work is carefully and continuously saved, with full option for the user to undo any activity at any time. The GUI design is close to most windows applications design, with a file menu where one can save/load work, reset, exit etc...

Furthermore, we use icons to present the people types, objects, emotions and relations. The use of icons makes it easy to construct annotations according the annotating language that was described in Chapter 4. The GUI cooperates with the Validator to make sure that the user cannot make annotating sentences that are incorrect according to the syntax of the language.

Note, that the icons are used purely to present concepts in the annotation language in a visual way, we also could use a normal text for that function, and this means that icons are unrelated to the annotating language we use.

Following the annotation language, we divided the icons into groups and provided the user of the system with five different classes of Icons in the GUI:

- **People:** This class contains the people that are in the train compartment
- **Expression:** This class contains the emotions and body languages of the people in the train.
- **Actions:** This class contains the actions or activities people can make in the train.
- **Objects:** This class contains some static and dynamic objects that are in the train.
- **Relations:** This class contains relations between people in the train.

The list of icons is listed in Appendix F.

5.3 The Validator component

As the Validator component is strongly related to the GUI component. We have included it in the GUI package. Because it will be easier to validate the user actions when he is interacting with the system, by giving immediate feedback on how it should be done when the user try to annotate something incorrect. The Validator makes sure that everything that is send to the expert system Jess is correct of syntax and error free.

5.4 Reasoning component

As mentioned in chapter 4, an expert system will be used for the reasoning.

We defined five scripts for the aggression levels: normal, fight, shout, control and begging:

Normal: The normal level is when everything on the train is normal, that is when there is no shouting, begging or fighting.

Fight: A fight will occur when 2 or more people are arguing.

Shout: When one or more persons are screaming or talking very loudly the Shout level will occur.

Control: When the conductor is checking the tickets this will be the result.

Begging: When a beggar is asking for money, begging will be the result.

The reasoning system works with numbers, the higher the number the most likely it is the correct case (Normal, Fighting, Shouting, Begging or Control). Figure 5.2 shows how the reasoning system works.

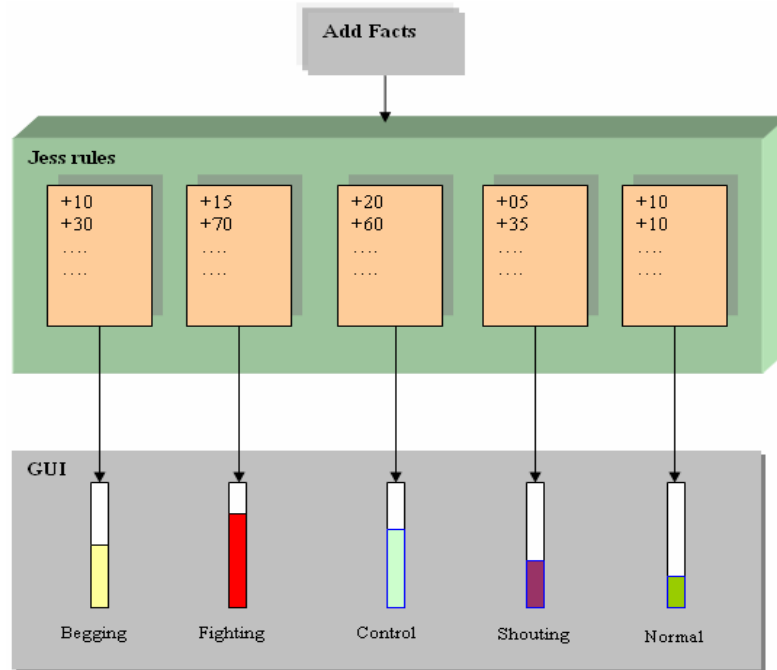


Figure 5.2: The reasoning system

When a fact is asserted, all the rules that have this fact in the rule base will be activated, and waiting for another fact to determine whether the rule will be fired or not, some rules have just one fact, for example when someone is holding a knife, one of the fighting rule will be fired. If all facts are asserted at once, then the rules that have the highest priority will be fired first.

There are also rules that subtract a certain amount of aggression from each level, this situation occur, for example when a hooligan is screaming and making some trouble the fight level will go up, two or three frames later a conductor arrives checking the tickets, the hooligan is now very quite and behaving well, then the fight level will decrease.

5.4.1 Rules design

When the user start annotating, then this mean that the user is firing facts into the rule-base. From there the expert system, Jess uses the Rete algorithm to determine an output. Figure 5.3 shows how it works.

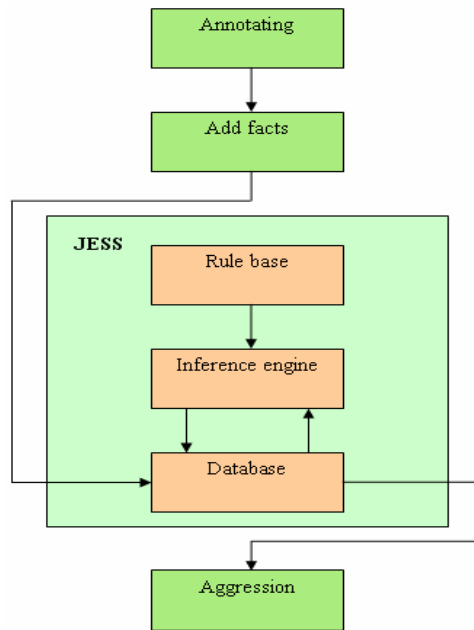


Figure 5.3: Reasoning architecture

5.4.2 Rule base

Rule-based systems, also called production systems, form a well-known architecture for implementing systems based on artificial intelligence techniques [31]. The heart of a rule-based system consists of a database, a rule-base and an inference engine.

The database contains a representation of the state of the environment in asserted facts. Upon annotating the environment the user asserts corresponding facts in the fact database. The rule base consists of a set of rules, each of which maps a specific state in the environment to one or more possible aggression levels and classifications. The rules take the following form:

```
if <list of conditions> then <list of aggression level and
classification>
```

Where <list of conditions> is associated with asserted facts in the database and <list of aggression level and classification> are possible outputs that may update other facts in the database or in the external environment. The connection between the facts in the database and the rules in the rule-base is made by the inference engine. Upon assertion of facts, the inference engine considers all rules in the rule-base. When a state of the world matches a rule, the rule is said to be fired.

5.5 The XML/ IO component

This component contains the inputs and outputs of the system. As we mentioned before we will use XML to store the annotations that are made by the user. We tried to make the structure of the XML file as readable as possible. Figure 5.4 shows the hierarchy of the XML file. The attributes under the slot “name” are different between human, activity, objects and relations. The example shown in figure 5.4 is taken from the human facts list. An example of XML file is provided in Appendix G. A DTD is the grammar of an XML [33]. It is an acronym that stands for Document Type Definition. It contains the elements, attributes, entities, and notations used in the XML document. The DTD of our XML file is shown below.

```
<?xml version='1.0' encoding='UTF-8'?>
<!ELEMENT fact-list (fact)*>
<!ELEMENT fact (slot|name)*>
<!ELEMENT name (#PCDATA)>
<!ELEMENT slot (value|name)*>
<!ELEMENT value (#PCDATA)>
<!ATTLIST value
  type CDATA #IMPLIED
>
```

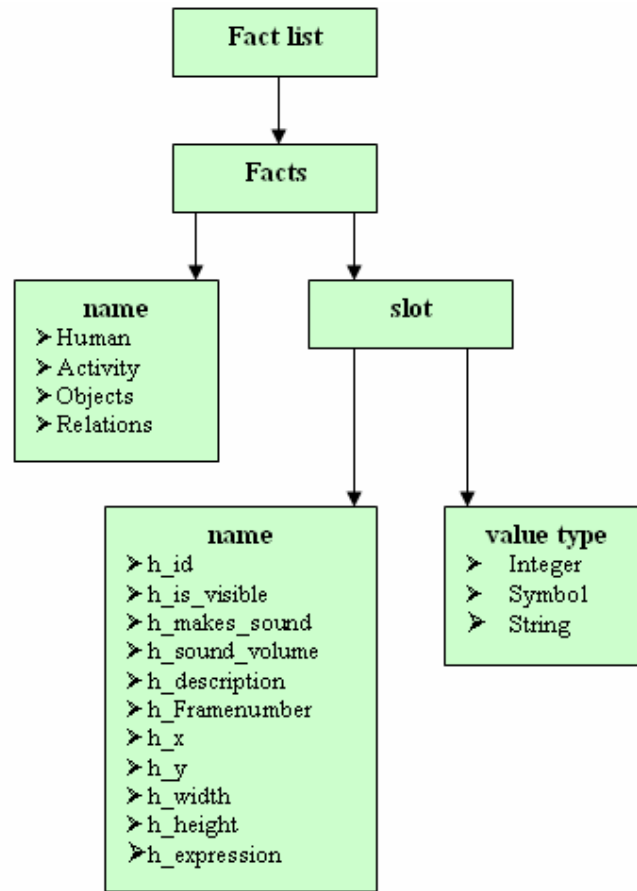


Figure 5.4: The XML file hierarchy (Human attributes)

The description of the attributes that are in slot “name” of human, activity, objects and relation are listed in the table’s resp. 5.1, 5.2, 5.3 and 5.4.

Table 5.1: Description of the human fact list

h_id	The unique Id of the human.
h_is_visible	Whether the human is visible or not.
h_makes_sound	Making sound or not.
h_sound_volume	The sound level.
h_description	Description of the human, entered by the user.
h_Framenumber	In which frames is the human visible.
h_x	The x coordinates of the top left corner.
h_y	The y coordinates of the top left corner.
h_width	Width of the human rectangle to be drawn.

h_height	Height of the human rectangle to be drawn.
h_expression	The expression of the human.

Table 5.2: Description of the activity fact list

a_id	The unique Id of the activity.
a_is_visible	Whether the activity is visible or not. Default is true.
a_object	The human Id performing this activity.
a_action	The action of the activity.
a_subject	The object Id that comes with the activity.
a_makes_sound	Making sound or not.
a_sound_volume	The sound level.
a_Framenumber	The frame of the activity.

Table 5.3: Description of the objects fact list

o_id	The unique Id of the object.
o_description	Description of the object.
o_is_visible	Whether the object is visible or not. Default is true.
o_makes_sound	Making sound or not.
o_sound_volume	The sound level.
o_framenumber	The frame of the object.
o_endframe	The frame where the object disappear from the scene.

Table 5.4: Description of the relation fact list

r_id	The unique Id of the relation.
r_object	The human Id involving in the relation.
r_relation	The relation type.
r_subject	The object involving in the relation.
r_is_visible	Whether the relation is visible or not. Default is true.
r_makes_sound	Making sound or not.
r_sound_volume	The sound level.

5.6 Class diagrams

Class diagrams identify the class structure of a system, including the properties and methods of each class. In addition, depicted are the various relationships that can exist between classes, such as an inheritance relationship. The Class diagram is one of the most widely used diagrams from the UML specification. Since there are many classes in the package, we will discuss the important classes only.

Moreover, we grouped related classes into packages as shown in figure 5.5

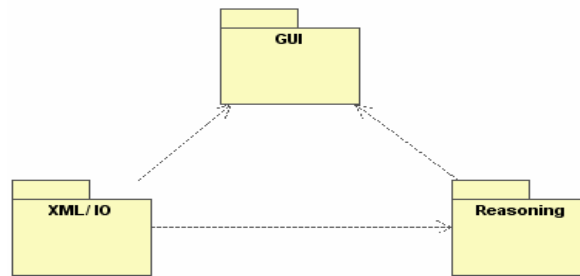


Figure 5.5: system main packages

The three packages are:

1. Reasoning: This package contains all the classes to reason about aggression.
2. GUI: This package contains the user interface and the Validator component.
3. XML/IO: The inputs and outputs of the system, XML is used to save and load annotation.

5.6.1 GUI package

Figure 5.6 shows part of the content of the package GUI. We created a Java Object class of each type of the facts we have in the XML file: Human, Object, Activity and Relations. As well as for each fact we have a set and a get function, this is done so we can easy acquire each fact separately from other. Note, that we did not draw other parts of the class diagram due to the large size that could not fit in one page.

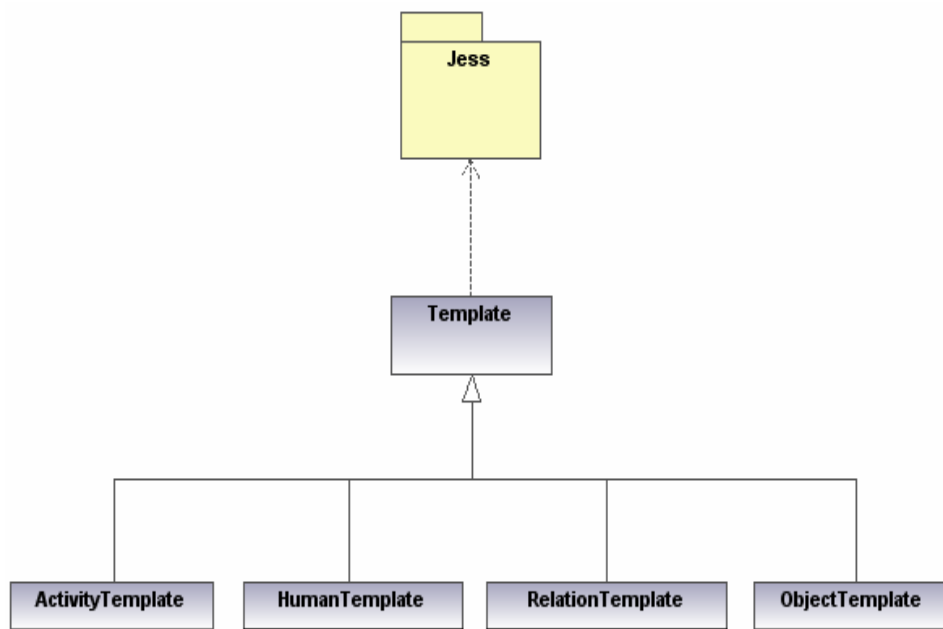


Figure 5.6: Class diagram of the package GUI

5.6.2 XML/IO Package

The XML/IO component (figure 5.7) is where the handling of the XML file is done. When loading XML file, all the facts that are in the XML file has to be converted back to Java objects, and this is done using a binder. This package also contains the media panel that is connected with the user interface where the user can play the video of the scenario he/she will annotate. Also when the user locates the people in the scene a JPEG file is created using this package.

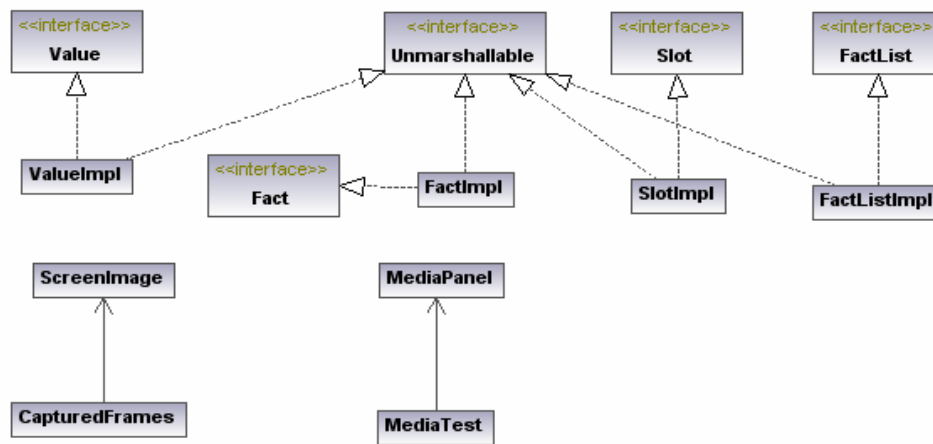


Figure 5.7: XML/IO Package class diagram

5.7 Dynamic behavior

To show what the system does when the user is annotating we used a sequence diagram figure 5.8. Showing the invocation of methods in each object, and the order in which the invocations occur. This represents the dynamic behavior of the system. On the horizontal axis, it shows the life of the object that it represents, while on the vertical axis, it shows the sequence of the creation or invocation of these objects.

5.7.1 GUI

The user starts by selecting a human icon, then the user has to type the description of the human chosen and then the user has to locate the location of the chosen human on the current frame. When the location of the human is set, the system has to create an image of the human, this image is created so it can be

shown when the user desire to see what the chosen human is doing during the whole scene. The annotated human will be added to the knowledge base, and Jess will print out the output. Note that we did not draw Objects, Activities and Relation on the sequence diagram, due to the fact that the diagram will be too large and it will not fit on one page. But Objects, Activities and Relations also have sets and gets methods, and they function the same as `setHuman()` and `getHuman()`.

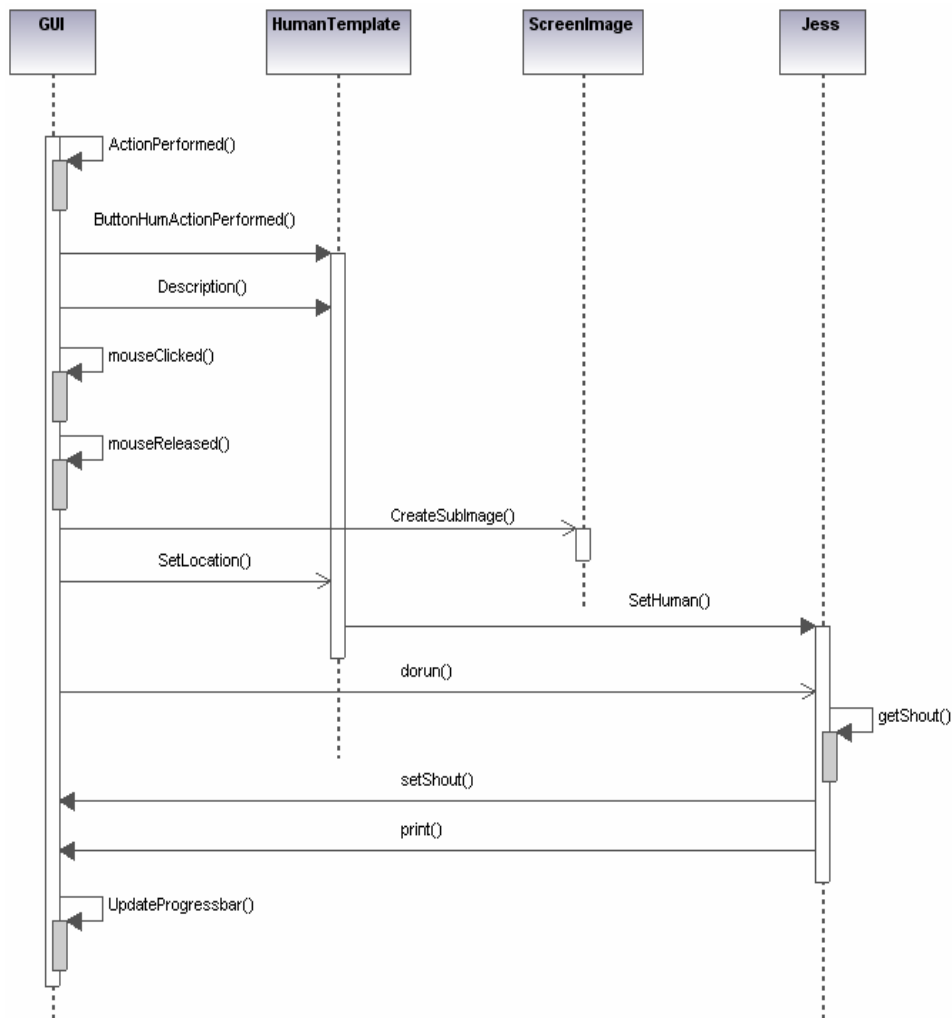


Figure 5.8: Sequence diagram: annotating process

5.7.2 XML

Figure 5.10 shows the sequence diagram for loading an XML file. First the method `ReadXML()` will read the file. The humans, objects, activities and relations are stored in their corresponding object type of file. After setting the XML file data as Java Object type using `setHuman()` and `setObjects()`, the method `DrawloadedObjects()` and `ListActivities` are called to draw the objects and activities on the GUI. Note that we did not draw `getRelations()`, and `getActivity()` because the figure will be too large. Jess is started using the method `dorun()`, All the facts that were in the XML file will be send to Jess. In the example we only used the fighting aggression level as an example. `getFight()` is called to get the level of aggression of fight. Then `setFight()` is used to set the fighting level in the GUI. The GUI now updates the progress bar (figure 5.9) to show the fighting level.

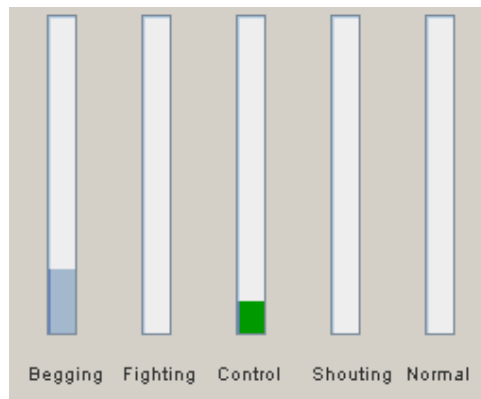


Figure 5.9: Progress bar updating the Control and begging levels

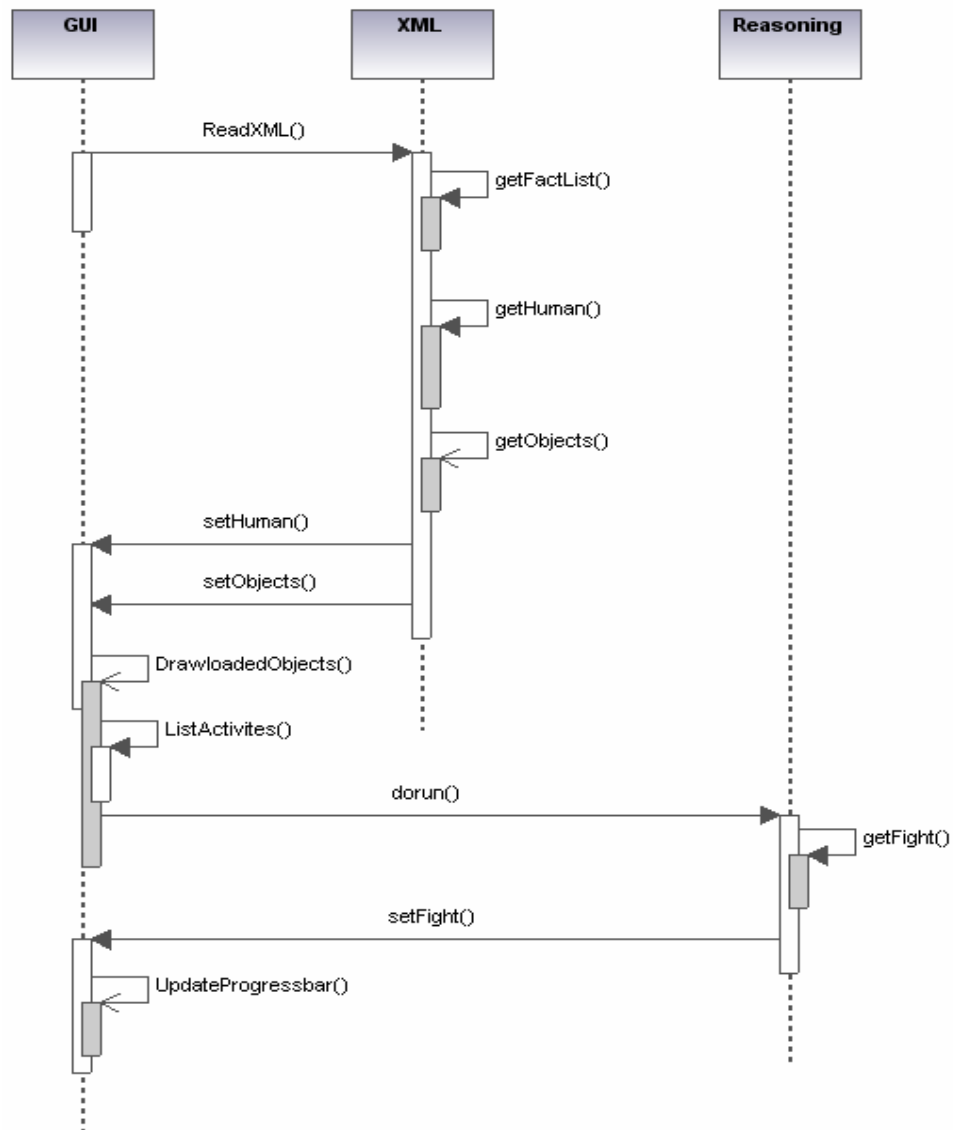


Figure 5.10: Sequence diagram: Loading XML file

5.8 Program Flow and Global usage

The flowchart in figure 5.11 presents the annotating process. The user first chooses a type of people; the user will be asked to type a small description of the chosen people type. After that, using the mouse, the user has to draw a rectangle

of the location of the human. An expression, relation or an action can be given after that. Finally clicking the Add button asserts the facts into Jess.

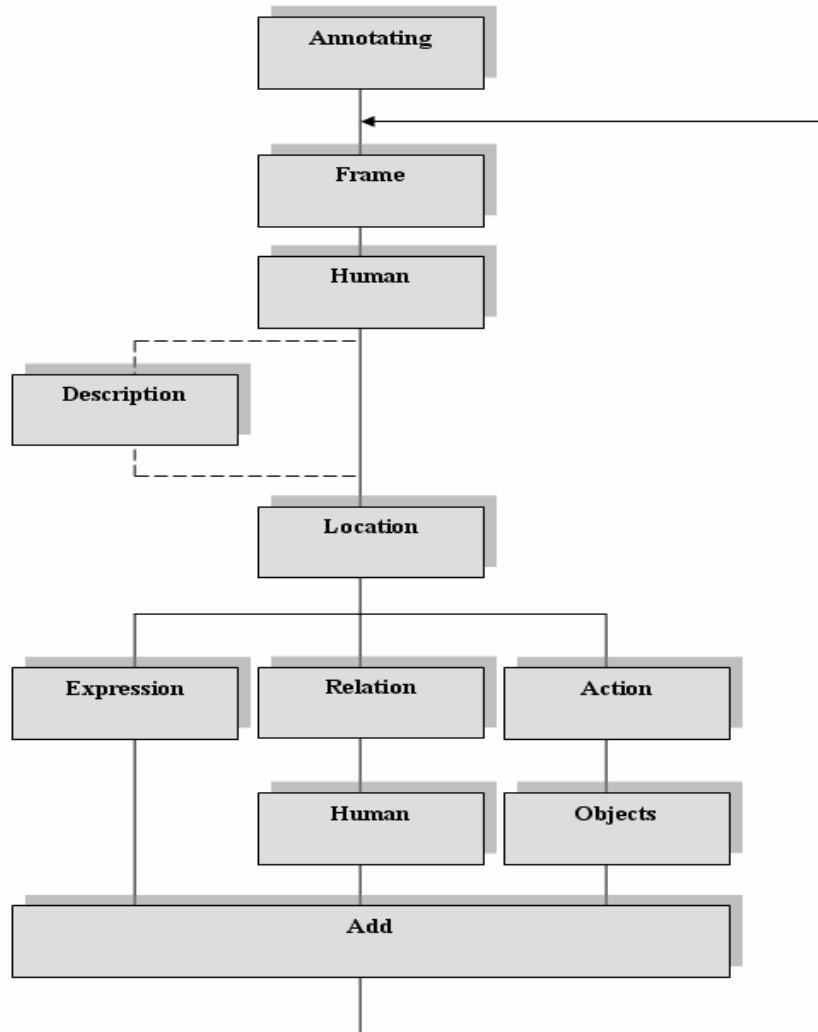


Figure 5.11: Flowchart of the annotating process.

Figure 5.12 shows a global use of the system and the options that can be performed. The user can watch the video of the scene he/she wants to annotate. After that the user has to open the frames he/she wants to annotate. Then the user can load an XML file that contains annotating he/she has made before. Or he/she can start annotating, and after that the user can save the XML file.

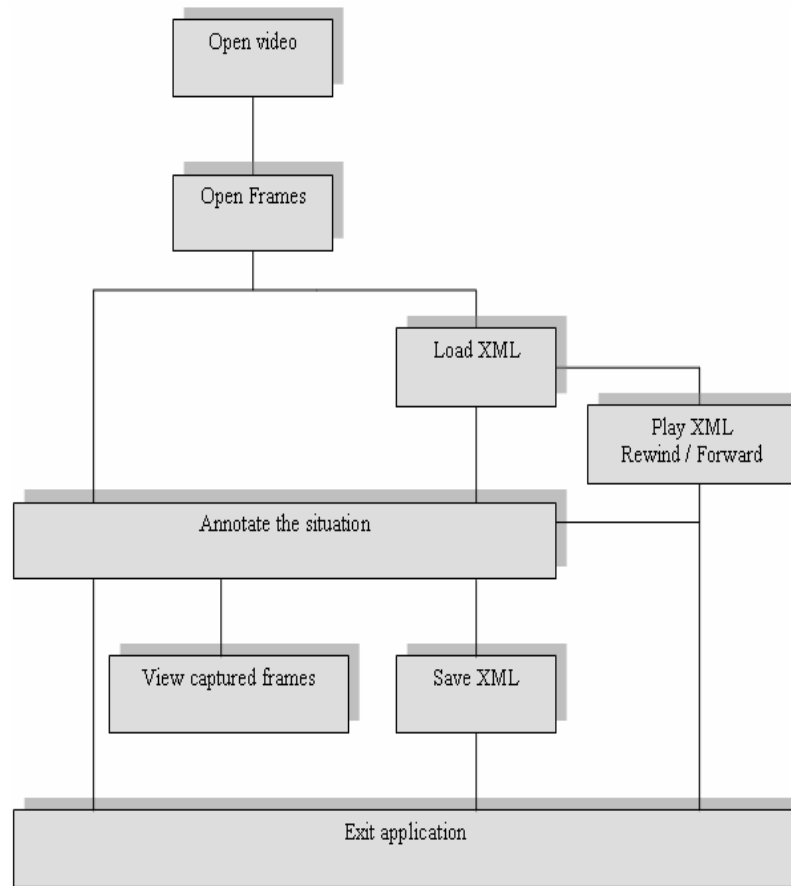


Figure 5.12: global usage of the system

6

Implementation

This chapter describes the actual implementation of the system. The system is implemented accordance with the design that is specified in Chapter 5. First an overview of the used software and tools is given and afterwards the implementation of expert system is discussed and finally some screenshots of the GUI will be shown.

6.1 Tools & Software

The overall used programming language for this project is Java. Java was chosen above other languages because, the expert system that will be used for the project Jess, is also implemented in Java, this mean that the communication between the expert system and the GUI will be easier when the GUI and the whole system is implemented in Java. The used Java version is 1.5.

6.1.1 NetBeans IDE

Netbeans IDE 5.0 is chosen for as a development environment. NetBeans provides a rich set of features and a more productive work environment than other IDE's. The NetBeans IDE is a free, Open-Source Integrated Development Environment for software developers. The IDE runs on many platforms including Windows, Linux, Solaris, and the MacOS.

6.1.2 Jess - Java Expert System Shell

Jess is a rule engine and scripting environment written entirely in Java. It was originally inspired by the CLIPS expert system shell, but has grown into a complete, distinct Java-in environment of its own. Because of its complete implementation in Java, the rule-engine can be easily embedded within the Java simulation environment. For detailed information about Jess [32].

6.1.3 XML

XML (eXtensible Markup Language) is a markup language for documents containing structured information. XML is used to store the annotation that had been made when using the system. Not only had the annotation saved in the XML file, but also the configuration of the system. Using the system an XML file can be loaded to continue the work on the previous session.

Xerces [8] is used to parse an XML file, and Zeus [9] used as a binder.

A binder is tool that binds the XML data into Java object models. Zeus is, in a nutshell, an open source Java-to-XML Data Binding tool. It provides a means of taking an arbitrary XML document and converting that document into a Java object representing the XML. That Java object can then be used and manipulated like any other Java object in the VM (virtual machine). Then, once the object has been modified and operated upon, Zeus can be used to convert the Java object back into an XML representation.

6.1.4 Version control

TortoiseSVN [10] is used for version control. TortoiseSVN is easy to use Revision control / version control / source control software for Windows. It is based on Subversion. TortoiseSVN provides a nice and easy user interface for Subversion. It is developed under the GPL. Which means it is completely free, including the source code.

6.2 Implementation of the expert system

As described in Chapter 5, the scripts have rules, when the user is annotating he/she will assert facts into the Jess database and from there a result will be given. Those rules are similar to CLIPS rules. There is also a script defined which provides the methods that are used in the five scripts described above. This script

```
(deftemplate humans
(slot h_id)
(slot h_is_visible)
(slot h_makes_sound)
(slot h_sound_volume)
(slot h_description)
(slot h_Framenumber)
(slot h_x)
(slot h_y)
(slot h_width)
(slot h_height)
(slot h_expression))

(deftemplate activity
(slot a_id)
(slot a_is_visible)
(slot a_object)
(slot a_action)
(slot a_subject)
(slot a_makes_sound)
(slot a_sound_volume)
(slot a_Framenumber))

(deftemplate objects
(slot o_id)
(slot o_description)
(slot o_is_visible)
(slot o_makes_sound)
(slot o_sound_volume)
(slot o_framenumber)
(slot o_endframe))

(deftemplate relation
(slot r_id)
(slot r_object)
(slot r_relation)
(slot r_subject)
(slot r_is_visible)
(slot r_makes_sound)
(slot r_sound_volume))
```

As we can see that each template has the same attributes as the XML file described in Chapter 5. That is done so it will be easy to bind the XML data into Jess facts and vice versa.

The rules that are in the aggression script, which is the engine of the whole expert system is shown below:

```
(defrule update-begging
  (update-begging ?c)
  =>
  (call ?s setBeg (+ (call ?s getBeg) ?c)))

(defrule update-shouting
  (update-shouting ?c)
  =>
  (call ?m setShout (+ (call ?m getShout) ?c)))

(defrule update-fighting
  (update-fighting ?c)
  =>
  (call ?f setFight (+ (call ?f getFight) ?c)))

(defrule update-fighting2
  (update-fighting2 ?c)
  =>
  (call ?f setFight (- (call ?f getFight) ?c)))

(defrule update-normal
  (update-normal ?c)
  =>
  (call ?n setNormal (+ (call ?n getNormal) ?c)))

(defrule update-normal2
  (update-normal2 ?c)
  =>
  (call ?n setNormal (- (call ?n getNormal) ?c)))

(defrule update-control
  (update-control ?c)
  =>
```

```
(call ?d setControl (+ (call ?d getControl) ?c)))

(defquery object-id-query
  "Look up fact id."
  (declare (variables ?id))
  ?fact <- (objects (o_id ?id)))

(defun modify-object-endframe
  "Modify objects instance endframe"
  (?id ?endframe)
  (bind ?results (run-query* object-id-query ?id))
  (?results next) ; Assumes exactly one objects
instance with ?id
  (modify (?results get fact) (o_endframe ?endframe))
  (close ?results)
  nil)
```

The rules shown above are rules to update the score of the corresponding script or aggression type. For example the rule: update-begging will update the begging level by first calling the current level and then adding the giving number and finally set the total as an output.

The query object-id looks for the fact id in the database of the inserted object id, this is used to update the o_endframe. It's used with the rule modify-object-endframe

Each of the 5 aggression scripts (normal, control, begging, shouting and fighting) has rules that contain facts. E.g. the control-rule 4, shown below will detect a conductor, and control-rule-1 will add 30 points more to the control level since the conductor is checking the tickets.

```
(defrule control-rule-4
(objects (o_description conductorclothing))
(relation (r_relation wear))
=>
(assert (update-control 10))
(printout t "control !" crlf)
)

(defrule control-rule-1
(activity (a_action check)(a_subject ticket))
=>
(assert (update-control 30))
(printout t "control !" crlf)
)
```

All the script files and rules are located in Appendix E

6.3 GUI & Screenshots

Figure 6.1 shows the user interface, on the right we can see what is annotated so far. We also can see that people which are currently in the scene are marked with a blue rectangle, and also their corresponding button is highlighted with black, if we choose another frame, the button will not be highlighted. We also can see a passenger with a white shirt holding a beer bottle which is drawn in the GUI.

There are 5 tabs, People, Expression, Relation, Actions and Objects. These tabs contain People/ objects relations that can be found on a train. Table 6.1 describes the content of the user interface shown in figure 6.1.

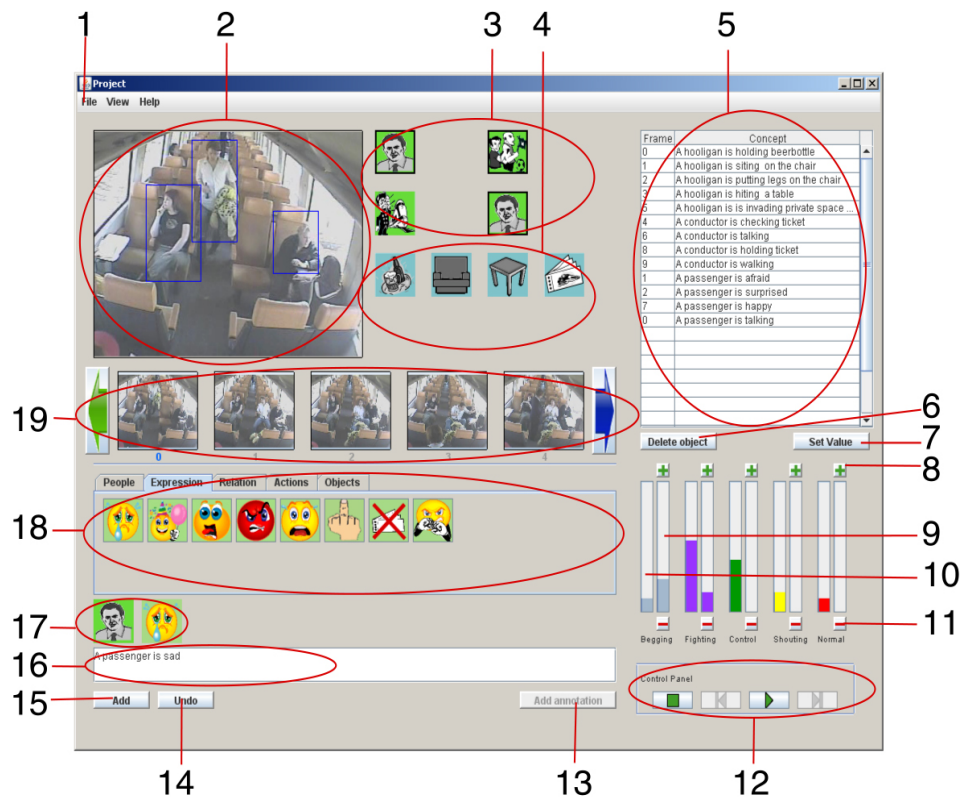


Figure 6.1: The user interface

Table 6.1: A description of the user interface.

Number	Description
1	The file menu. Contains: Opening video, Opening frames, saving and loading options, reset and exit.
2	The main frame, where annotations goes.
3	The people who are in the scene. They can be selected again when desired.
4	Objects in the scene. They can be selected again when desired.
5	The annotations that has been made.
6	Deleting an object in the scene, after selecting the object from the table.
7	Setting the value of the manual aggression level for the current frame.
8	Increase manual aggression level.
9	Manual aggression level meter.
10	Automatic aggression level meter.
11	Decrease manual aggression level.

12	Control panel. It will be functional when an XML file is loaded. It contains a play button, stop, forward and rewind frames buttons.
13	To add annotations after playing a loaded XML file.
14	Undo wrong drawn rectangle.
15	Add the current annotation to the database.
16	The current annotation.
17	The current selected buttons.
18	The content of the tabs.
19	Overview of the loaded frames.

The file menu is shown in figure 6.2. Table 6.2 describes the options in the menu

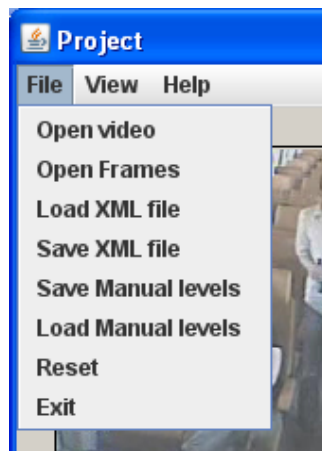


Figure 6.2: The file menu

Menu option	Description
Open video	A file browser will be opened to locate the video file the user wants to see before starting to annotate.
Open Frames	A file browser will be opened to locate the text file that contains the names of the images (frames) the user wants to load into the system.
Load XML file	Using a file browser the user can locate the desired XML file to load into the system.
Save XML file	Using a file browser the user can locate the

	desired location to save the XML file after typing the name of the file.
Save manual levels	Using a file browser the user can locate the desired location to save the file that contains manual set levels.
Load manual levels	Using a file browser the user can locate the location of the file to load into the system.
Reset	Reset the current annotating session.
Exit	Exit application.

Saving and loading manual levels are implemented so it will be easier for the user to compare what he/she thinks the aggression should be in a certain situation, comparing with what the system gives as output.

Loading and playing annotations

The user can load an annotation file and plays it. When playing the file the system will automatically use the loaded XML file to show what was annotated on every frame and what the aggression level is. The user also can rewind/forward frames using the rewind/forward buttons.

Thresholds

There are several thresholds implemented in the system. The thresholds will trigger an alarm sound and will make the aggression level text to flash when the aggression level is above 80. Figure 6.3

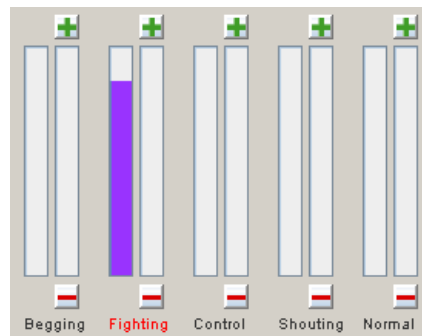


Figure 6.3: The fighting level is above 80. The text “fighting” flash white and red

6.4 Flexibility

One of the important goals and requirements of the system is the flexibility of the system. This means that the system should be easy to modify by inserting new rules/editing rules and inserting more icons into the GUI of the system. This paragraph provides the basic lines for modifying and adding new rules, and inserting new icons.

6.4.1 Adding new rules

Adding or editing rules is quite easy. In the CLP directory six files can be found. “aggression.clp” is the main file. The other five files are files for every script or aggression classification. The main file should not be edited because it’s the engine of the whole expert system, the main file consist of templates of the human, relations, objects and relations. Also the main rules for adding, subtracting and searching are located there.

Adding rules to each script should be located in the corresponding file.

A new rule e.g. for fighting, should be located in the fighting script “fight.clp” etc...

The rules form of the new rules should look similar to the rules that are already there e.g.

New fighting rule:

If a beggar is holding a knife

Then the fight level should go up by 60+

The rule can be added in the fight.clp as follow:

```
(defrule fight-rule-100
  (activity (a_action "hold"))
  (objects (o_description beggarclothing))
  (relation (r_relation wear))
  (objects (o_description knife))
  =>
  (assert (update-fighting 60))
)
```


Different update rules can be used:

- update-begging x : Update the begging level by adding the x amount to the current level.
- update-shouting x : Update the shouting level by adding the x amount to the current level.
- update-fighting : Update the fighting level by adding the x amount to the current level.
- update-fighting2 : Update the fighting level by subtracting the x amount from the current level.
- update-normal : Update the normal level by adding the x amount to the current level.
- update-normal: Update the normal level by subtracting the x amount from the current level.
- update-control : Update the control level by adding the x amount to the current level.

6.4.2 Adding new icons

Adding new icons to the GUI of the system should ONLY be done by a Java programmer or by someone who have knowledge of Java and programming languages.

Icon requirement:

Size: 50x50 pixels.

Format: Gif

Location: build\classes\images

In general every new icon has to be placed in one of the tabs (people, expression, relation, object, action and object) and when placing a new icon everything has to be copied from an icon of the same type.

For example adding a new expression has to be done as follow:

After designing the gif new.gif

In initComponents():

```
ExpressionButton[X].setIcon(new  
ImageIcon(getClass().getResource("/images/new.gif")));  
ExpressionButton[X].setToolTipText("NEW");  
ExpressionButton[X].setBounds(X1, X2, 50, 50);  
jDesktopPaneExpression.add(ExpressionButton[X],  
JLayeredPane.DEFAULT_LAYER);
```

X is the number of the new button we currently have 7 expression buttons so X should be 8. X1 and X2 are the new locations of the button.

If we compile and run the file, we will see the new icon displayed in the expression tab. The action listener is auto defined so there is no need to add it.

We now need to add action to the listener when the user clicks on the new icon. In the method “actionPerformed(ActionEvent e)” we add:

```
if(e.getSource() == ExpressionButton[X]){  
    if(clicked[0].getIcon()!=null){  
        for ( int i=0; i<43; i++ ){  
            HumanButton[i].setEnabled(false);  
            ActionButton[i].setEnabled(false);  
            ObjectButton[i].setEnabled(false);  
            ExpressionButton[i].setEnabled(true);  
            RelationButton[i].setEnabled(false);  
        }  
        jTextArea2.setText(jTextArea2.getText()+ " " + "is  
new");  
        clicked[1].setIcon(ExpressionButton[0].getIcon());  
        Pressed_Action = "new";  
        h_expression="new";  
    }  
}
```

When clicked disable other tabs. Note that we should use instead if the word “new” the function of the real expression e.g. crying and we should replace the word “new” by “crying”.

h_expression is what will go to the expert system.

This means that our rule for the new icon should consist of “crying” e.g.:

Rule fighting 101:

If a passenger is crying

Then update fighting to +20

Then we add the following lines in the fight.clp:

```
(defrule fight-rule-101
(humans (h_expression crying))
=>
(assert (update-fighting 20))
)
```

We are done with inserting this icon in the expression tab. The same technique can be used for adding other types of icons.

7

Testing & Experiment results

This chapter describes the test plan and presents the test results of the implemented system. First an explanation about how the data was collected is provided, afterward the test plan will be described and finally the experiments results will be presented.

7.1 Data collecting

Data Collection is an important aspect of any type of research study. Inaccurate data collection can impact the results of a study and ultimately lead to invalid results. The project group of aggression detecting had collected data using scenarios [11]. The MMI project group used real train compartments to perform those scenarios. Using multimodal cameras they recorded video and sound of actors acting out a certain predefined scenario.

There were two types of scenarios recorded:

- Scenarios containing aggressive situation.
- Scenarios containing non aggressive situation.

The scenarios performed were: Shouting, smoking, hooligans, beggar, conductor, mobile phone, drunkard, shoe on seat, mother with child and graffiti and vandalism.

Later, the videos were analyzed and for every video a number of notes were made, these notes helped the design of the system a lot since they describe what is in the scene and what action is being performed by the actors.

7.2 Test plan

Usability is one of the important aspects which determines product's acceptance by users and its exhaustive use. The ease of use is now becoming an increasingly important factor in the purchase of software. A software product may have great features and wide array of options but if it is difficult to use and is complicated, then there is a strong chance of an otherwise 'good' product to get branded as 'bad' product. While developing and testing the software, one must take into consideration is that both a novice and a seasoned user are able to use it with relative ease.

A focused testing effort on usability aspects of the product leads to development of more usable product and satisfied customers.

The testing will be carried out by students who work at the MMI lab. A task list (Appendix A) is giving where testers have to complete a certain series of tasks using the system. The first few tasks that were given were relatively easy to do, and everything was explained exactly, but later on the tasks were open and the user can do what he/she thinks is the right action, this is done so we can measure the learn-ability of the system. After finishing the test, the users have to fill in a debriefing form (Appendix B), where we ask the testers on their opinion regarding the system, and to give a grade about several things like (difficulty, user friendly). We also asked the testers to give us three things they like and dislike about the system. This is done to get a better idea about how the testers think about the system and the usability of it.

7.2.1 Test setup

The tests were carried out in the MMI lab (figure 7.1). The tester will carry out the test on the computer where the system is installed on. The observer will sit next to him/her, taking notes about what the user is actually doing and where things went wrong. The observer also comments on errors when they occur and keeps a timeline about the user actions.



Figure 7.1: Testing in the MMI lab (the observer right, the tester left)

7.2.2 Usability testing

To identify discrepancies between the user interface of the system and the human engineering requirements of users, usability testing has to be performed. The user interface is the module of the software which comes in direct interaction with the user so its logical organization design and correctness plays an important part in the better usability of product.

A good way to define usability testing is "testing the software from a users point of view", which boils down to testing the software to ensure that it is 'user-friendly'. Usability testing is different from functionality testing as the later involves validating the software against the requirements only. In simple words

usability testing includes ergonomic considerations, screen design, consistency in user interface, proper naming conventions of the menus and options.

The benefits of usability testing are:

- Overall quality of the software improves as the system becomes easy to use efficient and responsive.
- Shortens the learning curve and training efforts for new users and hence saves time for the user.

7.2.3 User experience measures

What to look for and how to ensure that usability is built in the system. There are many points which should be kept in mind while writing test plans and creating test cases.

Every test will include usability measures such as: learn-ability, effectiveness, efficiency, memorability and satisfaction. Those measures can be distinguished and tested in particular evaluations. Table 7.1 describes what every measurement is.

Table 7.1: Description of the usability measurements

Measurement	Description
Learn-ability	Time to learn, here we will see how long it takes the participant to learn how to use the system by making two similar tasks; one at the beginning, one at the end and compare the time needed to accomplish the task.
Efficiency & Accessibility	The product should be efficient and it features and options should be easily accessible. The main UI should have all the major functions and options but at the same time it should not be cluttered in menus. All the readily used options and important menus should be easily accessible. While testing any feature one must check how many steps are required to complete a task. Tester should ascertain whether these steps are justified or there is a scope of reducing the steps and

	increasing the efficiency.
Effectiveness	Rate of errors by the participants where we will record how many and what kind of errors do the participants make while carrying out the given tasks.
Memorability	Retention over time where we will observe and see how well do the participants maintain their knowledge over the system and the use of it after a demonstration, for example if the participants will remember the paths they have taken to do a given task and the way to use the system.
Satisfaction	Subjective satisfaction where we will record how much did the participants like using the different aspects of the user interface of the system. The answer can be determined by a closing interview or a questionnaire that include satisfaction scales and space for free comments.
Consistency	It should be ensured that all error and warning messages are intuitive and consistent .They should convey meaningful messages that enables user to understand the error condition and remedial actions. The output should also be consistent when facing the same scenario.

7.2.4 Test procedure

When carried out the tests we used the following procedure:

- 1- Give some background information about train aggression.
- 2- Introduce the system to the participants, what the purpose of the system is and how you can use it.
- 3- Give a short demo and explain what you are doing.

- 4- Train the participants to "think out loud" so that you are able to better understand their experiences with the system.
- 5- Begin the test by giving the participants a task list that they must carry out. Appendix A.
- 6- Carry out a closing interview to review overall impressions of the Health-Pal system.
- 7- Give the participants a questionnaire about the tasks they carried out, and about the system it self. Appendix B.
- 8- Thank the participants.

7.2.5 Data analysis

Once the usability test sessions are complete, using the saved session by the testers and the data-logging of the tests, the following activities are carried out to compress the data and present the findings. Nominally, we expect to perform the following kinds of analyses to measure the objective and subjective performance of the system:

1. Evaluate the efficacy of the system information architecture and navigation,
2. Seek an understanding of the system strengths and weaknesses regarding branding and information offerings,
3. Organize anecdotal comments in a manner that provides a clear picture of the system strengths and weaknesses,
4. Classify and diagnose errors (e.g., navigational, procedural, data entry, interpretation, etc.), and
5. List observed usability problems.

7.2.6 Test Case

We also used a test case to test the functionality of the system. The most common definition of a test case is a set of conditions or variables under which a tester will determine if a requirement or use case upon an application is partially or fully satisfied. It may take many test cases to determine that a requirement is fully satisfied. In order to fully test all the requirements of an application are met, there must be at least one test case for each requirement unless a requirement has sub

requirements. In that situation, each sub requirement must have at least one test case.

Test case activity:

- **Initialization** describes actions, which must be performed before test case execution is started. For example, we should open some files.
- **Finalization** describes actions to be done after test case is performed. For example if test case crashes database, tester should restore it before other test cases will be performed.
- **Actions** step by step to be done to complete test.
- **Input data** description.
- **Results :**
 - § **Expected results** contains description of what tester should see after all test steps has been completed.
 - § **Actual results** contain a brief description of what the tester saw after the test steps has been completed. This is often replaced with a **Pass/Fail**. Quite often if a test case fails, reference to the defect involved should be listed in this column.

We created four different test cases these are:

1. Threshold for fighting.
2. Annotating a hooligan holding a bottle of beer.
3. Saving XML file.
4. Loading XML file.

7.2.7 Testing goals

It is important to know why we are testing. Below is a summary of the testing goals from the previous chapters:

- Ø To test the user-friendliness of the system.
- Ø To test for errors.
- Ø To test the learn-ability of the system.
- Ø To test the efficiency & accessibility of the system.

- Ø To test the effectiveness of the system.
- Ø To test the consistency of the system.
- Ø To test the memorability of the system.
- Ø To test the satisfaction of the system.
- Ø To test the functionality of the system.
- Ø To test the system outputs.

7.3 Experiment result

Before we reveal the testing results, we first give a short overview of our test subjects. Afterwards we present the expectation, and finally the results will be presented.

7.3.1 Test subjects

For our experiment we had 5 test subjects, all of the 5 subjects are students from the MMI group. The following table gives a summary of the participant

Table 7.2: Test subjects characteristics

Amount:	5 participants.
Age:	Varies from 21 to 28 years old.
Gender:	1x Female and 4x Male.
Highest level of education:	Bachelor of science (BSc).

These test subjects presents our user group.

7.3.2 Expectations

We expect small problems with the following activities:

- Ø Making annotations (syntax errors).
- Ø Annotate irrelevant things.
- Ø Annotate too much.
- Ø Not annotate everything that is relevant.
- Ø Drawing a rectangle around the human.

We also expect that the time to accomplish the task list will be between 10-15 minutes and the debriefing form 10 minutes.

7.3.3 Results

All of our expectations were justified when we looked at the experiment results, and even some more came to our attention. Some expectations we had were not justified but these were all duration time expectations. We expected that the time which the test subject needed to finish the task list would be about 10-15 minutes but it took them 15-20 minutes. We expected that the debriefing would take 5 minutes but it took 10 minutes.

During the first 2 tests we found many bugs and errors we did not find before, because we did not make mistakes using the system, but the testers made some mistakes and we found those bugs. We fixed the bugs and errors and continued with the test with the other three test subjects.

The entire test subjects were able to accomplish all the tasks during the test. The average grade that was given by the test subjects about the overall system was 7.8.

The word “Clear” was the most mentioned characteristic of the system. Table 7.3 provides the top three things the test subject liked or disliked about the system.

Table 7.3: Top three characteristics

	3 characteristics that describe the system	3 best things about the system	3 things disliked about the system
1	Clear	Annotation goes quick once you know the icons	Sometimes slow
2	Easy to use	The aggression level that was given as a result was close to the real scenario.	No Voices in the videos
3	Nice	Icons	Too many help windows

All the four test cases we passed. Below is the output of the test case “Threshold for fighting”. The complete test case results can be found in appendix D.

- **Initialization:** video frames must be opened. Annotation has to be made to the system.
- **Finalization** None.
- **Actions** Annotate an aggressive situation where the fighting level goes up.
- **Input data** None.
- **Results**
 - **Expected results** The fighting level goes up, the text Fighting must flash and a sound of police siren must be played.
 - **Actual results :**

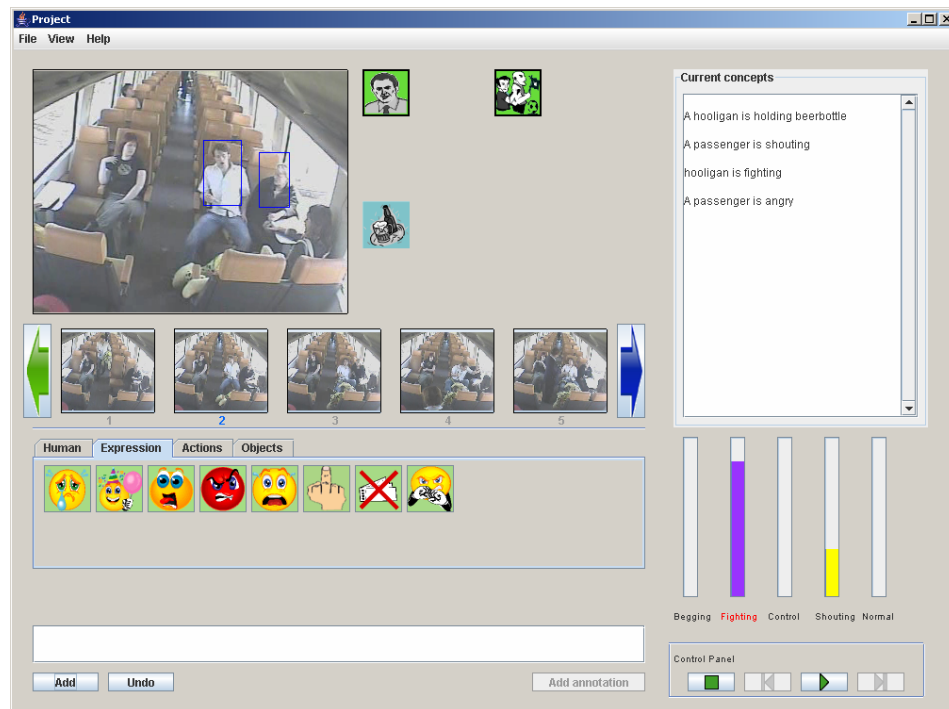


Figure 7.1: Fighting level is flashing

Test case: PASS

System data logging:

```
f-0    (MAIN::initial-fact)
f-1    (MAIN::humans (h_id 9) (h_is_visible null) (h_makes_sound
null) (h_sound_volume null) (h_description "white shirt")
(h_Framenumber 0) (h_x 121) (h_y 15) (h_width 57) (h_height 95)
(h_expression null))
f-2    (MAIN::activity (a_id 1) (a_is_visible null) (a_object 9)
(a_action "hold") (a_subject 806) (a_makes_sound null)
(a_sound_volume null) (a_Framenumber 0))
f-3    (MAIN::objects (o_id 806) (o_description beerbottle)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
f-4    (MAIN::objects (o_id 100) (o_description hooliganclothing)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
f-5    (MAIN::relation (r_id 1) (r_object 9) (r_relation wear)
(r_subject 100) (r_is_visible null) (r_makes_sound null)
(r_sound_volume null))
f-6    (MAIN::update-fighting 10)
f-7    (MAIN::humans (h_id 133) (h_is_visible null)
(h_makes_sound null) (h_sound_volume null) (h_description "")
(h_Framenumber 0) (h_x 234) (h_y 94) (h_width 40) (h_height 52)
(h_expression null))
f-8    (MAIN::activity (a_id 133) (a_is_visible null) (a_object
133) (a_action "shout") (a_subject null) (a_makes_sound null)
(a_sound_volume null) (a_Framenumber 0))
f-9    (MAIN::update-shouting 30)
f-10   (MAIN::humans (h_id 9) (h_is_visible null) (h_makes_sound
null) (h_sound_volume null) (h_description "white shirt")
(h_Framenumber 1) (h_x 181) (h_y 75) (h_width 40) (h_height 69)
(h_expression fighting))
f-11   (MAIN::activity (a_id 133) (a_is_visible null) (a_object
null) (a_action "null") (a_subject null) (a_makes_sound null)
(a_sound_volume null) (a_Framenumber 1))
f-12   (MAIN::update-fighting 60)
f-13   (MAIN::humans (h_id 133) (h_is_visible null)
(h_makes_sound null) (h_sound_volume null) (h_description "")
(h_Framenumber 1) (h_x 240) (h_y 88) (h_width 32) (h_height 58)
(h_expression angry))
f-14   (MAIN::update-fighting 15)
```

Fighting: 85
Sound Played.

8

Conclusions & Recommendations

In this chapter the results of the implemented system of the train aggression project will be discussed. Next the project will be assessed against the requirements and goals that were defined in the beginning. Furthermore, recommendations for future work will be given.

8.1 Conclusion

This thesis report presents the “Video content analyze & aggression detection system for a train environment”. The architecture, the reasoning system and the annotation language was the main focus of this project. The resulting system is a system that makes it possible for the user to:

- Save an annotation on XML file for a certain scene.
- See what aggression classification is best fitted in a certain scene based on predefined rules.
- The level of aggression according to the situation.
- Load a saved session and continue to analyze or add new annotations.
- Compare between the automatic level of aggression output that is generated by the system and the manual aggression levels that the user set according to his/her thoughts about the situation.
- Compare the level of aggression between the frames.

8.1.1 Literature

The literature survey introduced what aggression is and specially the human aggression. This was necessary in this project, since it is all about human aggression in a train environment.

The literature survey also introduced a few examples of annotating tools, rule base, and expert systems. The information that was available in those papers is very good and very common to our problem.

8.1.2 Design of the system

The system is designed from scratch. Therefore, the design that has been described in chapter 5 is very common and reasonable for such a system. Most of the requirements have been fulfilled. The user interface is designed to guide the user through the annotation process giving feedback whenever possible and disabling options that is not logical. Many help tips is been implemented in the system so that the user can work efficiently and minimize errors.

The user interface was designed to be simple, intuitive, and visually appealing.

The developed system is flexible, new rules can be easy added by editing the CLP file. Also new icons of people, expressions, relations, actions and objects can be added to the interface by modifying the GUI file.

8.1.3 Design of the reasoning algorithm

An important part of the system the reasoning algorithm of the “Video content analyze & aggression detection system for a train environment”.

Based on the annotation that is done by the user an output of aggression level is generated by the expert system. This requirement is also fulfilled. The developed reasoning model is implemented as a rule based system. Inserting facts to the database will return a certain level of aggression and classification between, fighting, shouting, control, normal and begging. The reasoning model has been tested extensively and the results were quantified, which means that the reasoning model is quite accurate.

8.1.4 Implementation of the system

Based on the design, the system was developed incrementally. First, a basic work frame was implemented. Next, all objects, people, relations, and actions are

implemented. Then, the user interface was developed. Finally, more complicated features such as the capturing images, drawing the rectangles, saving and loading to XML file were implemented. We can say that the approach for the implementation was a good choice, since we were able to implement in phases and test in phases.

Developing the system using the programming language Java, made it easy to accomplish the goal that the system should be OS independent.

8.2 Recommendations

Throughout the individual chapters of this thesis, a number of suggestions for future work have been given. This section summarizes these.

8.2.1 Approach

The final goal was to develop a model to annotate a video sequence of images and to detect the level of aggression.

Dynamic Bayesian networks are probably the most appropriate approach, since we will be able to use pattern recognizing to classify aggression, based on either *a priori* knowledge or on statistical information extracted from the patterns [34]. But because of its simplicity and availability of data we use a rule based system as a first approach. It would be nice to know whether Dynamic Bayesian networks will be more accurate and more efficient than using the expert system and a rule based approach that is used in this report. It would be clever if we could compare both outputs in one system.

8.2.2 Annotation language

The annotation language that was described in Chapter 4 is recent, and therefore it is up to date and not too complex. By adding more features it can become a language that can be used as a standard language for annotating in a train environment.

New features could be:

- Adding “and” in the sentence form e.g. “the passenger is sitting on a seat and holding a cell phone”, the current language can only handle “the passenger is sitting on a seat” and “the passenger is holding a cell phone”.

- Make distinguish between male and female by adding a gender slot to the human template.
- Add objects “belong” to passenger. This way we can define a theft when someone pick up an object that belong to other.
- Grouping people: by adding this feature it will be easier to detect which people can be seen as one group, and here we can define a group of hooligan e.g. or a group of passengers that are fighting with one person, which is an aggressive situation.
- Grouping objects: This feature can be used along with the “belong” feature. It will be easier to detect theft when it occurs, since we know which objects belong to one passenger.

Reference

- [1] Sci-Tech Encyclopedia: Aggression, <http://www.answers.com/topic/aggression>, Last visited at September 25, 2007.
- [2] Aggression in Human Beings, Factors of Aggression and What Can Be Done to Lesson it, http://www.associatedcontent.com/article/4600/aggression_in_human_beings.html, Last visited at September 25, 2007.
- [3] Joanna Schaffhausen , “*The Biological Basis of Aggression*”, Published at <http://www.brainconnection.com> September 1, 2007.
- [4] Agressie in de trein neemt toe, Algemeen Dagblad, 1 november 2003.
- [5] Treinkapitein heeft het zwaar, <http://www.nrc.nl/W2/Lab/Spoorwegen/001014a.html>, Last visited at September 25, 2007.
- [6] Spoorvandalisme kost zeker 20 miljoen per jaar, <http://www.home.nl/nieuws/binnenland/artikel/00114122>, Last visited at September 25, 2007.
- [7] Rete algorithm, http://en.wikipedia.org/wiki/Rete_algorithm, Last visited at September 25, 2007.
- [8] Xerces Java Parser, <http://xerces.apache.org/xerces-j/>, Last visited at September 25, 2007.
- [9] Enhydra Zeus Project, <http://forge.objectweb.org/projects/zeus/>, Last visited at September 25, 2007.
- [10] TortoiseSVN, <http://tortoisesvn.net/>, Last visited at September 25, 2007.
- [11] Z. Yang, L.J.M. Rothkrantz, “*Aggression Detection in Train, Compartments Data Collection Plan*” April 2006.
- [12] Jaar cijfers NS, http://www.ns.nl/servlet/Satellite?cid=1171010494432&pagename=www.ns.nl%2FPage%2FArtikelPage_www.ns.nl&p=1171010494432&lang=nl&c=Page, Last visited at September 25, 2007.
- [13] Centraal Bureau voor de Statistiek, Thema veiligheid, verkeer, www.cbs.nl, Last visited at September 25, 2007.
- [14] Moyer, KE. ”*Kinds of aggression and their physiological basis. Communications in Behavioral Biology*” 2A:65-87, 1968.

- [15] Behar, D., J. Hunt, A. Ricciuti, D. Stoff, and B. Vitiello. "Subtyping Aggression in Children and Adolescents" *The Journal of Neuropsychiatry & Clinical Neurosciences* 2 (1990): 189-192. 7 Dec. 2006.
- [16] Crews, D, N Greenberg, and M Scott. "Role of the Amygdala in the Reproductive and Aggressive Behavior of the Lizard" *Physiology & Behavior* 32: 147- 151, 1984.
- [17] Craig A. Anderson & L. Rowell Huesmann, "Human Aggression: A Social-Cognitive View", in *THE SAGE HANDBOOK OF SOCIAL PSYCHOLOGY* 296-323, Michael A. Hogg & Joel Cooper eds., 2003.
- [18] Hisashi Miyamori, Shun-ichi Iisaku , "Video Annotation for Content-based Retrieval using Human Behavior Analysis and Domain Knowledge", *Automatic Face and Gesture Recognition*, 2000 Page(s):320 – 325, Proceedings. Fourth IEEE International Conference on 28-30 March 2000 .
- [19] Dorado, A.; Calic, J.; Izquierdo, E , "A rule-based video annotation system", *circuits and Systems for Video Technology*, IEEE Transactions on Volume 14, Issue 5, Page(s):622 – 633, Digital Object Identifier 10.1109/TCSVT.2004.826764, May 2004.
- [20] M. Pantic and L.J.M. Rothkrantz, "Expert system for automatic analysis of facial expressions" *Image and Vision Computing*, vol. 18, no. 11, pp. 881--905, 2000.
- [21] Moyer, K. E. (1968) "Kinds of aggression and their physiological basis", *Communications in Behavioral Biology* 2:65-87.
- [22] Ekman, P. Basic Emotions, in T. Dalgleish and T. Power (Eds.) *the Handbook of Cognition and Emotion* Pp. 45-60. Sussex, U.K.: John Wiley & Sons, Ltd. 1999.
- [23] McKenna, S.J. et al. "Tracking Groups of People." *Computer Vision and Image Understanding*, 80(1):42-56, 2000.
- [24] J. Russell and J. Fernandez-Dols, "The Psychology of Facial Expression", New York: Cambridge Univ. Press, 1997.
- [25] M. Pantic, L.J.M. Rothkrantz , "Facial Action Recognition for Facial Expression Analysis from Static Face Images", *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics* Volume 34, Issue 3, pp.1449–1461, June, 2004 .
- [26] Neil Robertson, Ian Reid, "A General Method for Human Activity Recognition in Video", *Computer Vision and Image Understanding* 29 June 2006

- [27] Jeffrey Davis, “*The Culture of Body Language: How a Person's Gestures Take on Meaning in Different Parts of the World*”, The People media company, Published Mar 01, 2006.
- [28] Extensible Markup Language (XML), <http://www.w3.org/XML/>, Last visited at September 25, 2007.
- [29] XSL Transformations (XSLT) , <http://www.w3.org/TR/xslt>, Last visited at September 25, 2007.
- [30] Jess the Rule Engine for the Java Platform, <http://herzberg.ca.sandia.gov/jess/>, Last visited at September 25, 2007.
- [31] Production system, http://en.wikipedia.org/wiki/Production_system, Last visited at September 25, 2007.
- [32] Ernest Friedman-Hill, Jess in Action. Java Rule-based Systems, ISBN: 1930110898, 2003.
- [33] Introduction to DTD, http://www.w3schools.com/dtd/dtd_intro.asp, Last visited at September 25, 2007.
- [34] Sergios Theodoridis, Konstantinos Koutroumbas , Pattern Recognition, 3rd Edition, ISBN 0123695317, February 2006
- [35] Bratman, M. E., Intention, Plans, and Practical Reason, CSLI Publications. ISBN 1-57586-192-5, [1987] (1999).
- [36] Wooldridge, M, “*Reasoning About Rational Agents*”, The MIT Press. ISBN 0-262-23213-8, 2000.
- [37] P.Klahr and D.Waterman, “*Expert systems techniques, tools and applications*”, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1986
- [38] S. Fitrianie and L.J.M.Rothkrantz, “*An Icon-Based Communication Tool on a PDA*”, Euromedia 2005, volume 11, pages 83-90, ISBN 90-77381-17-1, 2005
- [39] P.Schooneman, “*ISME – Icon based System for Managing Emergencies*”, Master Thesis, TU Delft, 2005.
- [40] Sabrina Sestito, Tharam S. Dillon, “*Automated Knowledge Acquisition*”, Prentice Hall of Australia Pty Ltd, 1994.
- [41] Han Reichgelt, “*Knowledge Representation: an AI perspective*”, Ablex Publishing Corporation, 1991.
- [42] Hector J. Levesque and Gerhard Lakemeyer, “*The logic of knowledge bases*”, The MIT Press, 2000.

- [43] Peter Jackson, "*Introduction to expert systems*", 2nd edition, Addison-Wesley Publishingcompany, 1990.
- [44] Waterfall model, http://en.wikipedia.org/wiki/Waterfall_model, Last visited at September 25, 2007.
- [45] Fuller, RW, "*The influence of fluoxetine on aggressive behavior*" *Neuropsychopharmacology*, 14: 77-81, 1996.
- [46] Robert A. Baron, Deborah R. Richardson, "*Human aggression*" Second edition, ISBN: 978-0306444586, Plenum Publishers, Nov 1993

APPENDIX A

Tasks list

The following list presents the activities the testers have to perform during the test:

1. Start the application using F6
2. Open video file “beggar2.mpg” located in the desktop.
3. Watch the video.
4. Open frames using the file “03_beggar.txt”

We want to annotate the man in the yellow shirt as a beggar holding money

5. Click in the human tab on a beggar
6. Type as description for the beggar, yellow shirt and click Ok
7. Read the popup message, and draw a rectangle around the beggar, using the mouse and click ok.
8. Click on the action tab, and choose walking.
9. Click on Add.
10. Go to frame 2.
11. Choose the beggar you just annotated, and draw a rectangle in his new location
12. In the action tab choose Holding, and in the objects tab, choose money and click on add.
13. Click on frame 3. Choose in the human tab, a passenger.
14. Type as description, long hair and draw a rectangle around the passenger with the long hair.
15. In the action tab, choose sitting, and in the object tab, a seat. And click on Add
16. In frame 4, choose the passenger with the long hair, locate the new location, and in the expression tab, choose afraid and then add.
17. In frame 5, the passenger with the long hair is angry.
18. In frame 6, the male passenger in the white shirt, is surprised.
19. In frame 7, the passenger with long hair is fighting.
20. In frame 7, the beggar with long hair is fighting.
21. In frame 8, the passenger in the white shirt is shouting.
22. In frame 9, the beggar is afraid.

23. In frame 10, the beggar is walking.
24. view captured images
25. save XML file
26. exit

We want you to annotate on your own now:

27. Start the application using F6
28. Open video file “hooligan.mpg” located in the desktop.
29. Open frames using the file “01_associale_hooligan.txt”
30. Start annotating until you reach frame 5
31. Save XML file
32. Close the program
33. Start application
34. Open frames using the file “01_associale_hooligan.txt”
35. Load the saved XML file
36. View captured images
37. Continue annotating starting from frame 6
38. Save XML file
39. Exit

Thank you for your time, please proceed with the debriefing form

APPENDIX B

Debriefing form

1. Were you able to accomplish the tasks?

☐ Yes ☐ No

2. How easy or difficult was it for you to accomplish this task?

☐ very easy ☐ easy ☐ normal ☐ difficult ☐ very difficult

3. What barriers did you encounter?

.....
.....

4. Is the information provided what you expected to find?

☐ Yes ☐ No

Why?

.....
.....

5. Is this information displayed in an appropriate manner?

☐ Yes ☐ No

Why?

.....
.....

6. What is your impression of this system overall? Please give it a grade from 1 to 10. (1= very bad, 10=perfect). Grade:

Why, that grade?

.....
.....

7. Name three words or characteristics that describe the system.

...
...

...

8. What were the 3 things you liked best about the system?

...

...

...

9. What were the 3 things you liked least about the system?

...

...

...

11. Are there things you would like added to the system? Which ones and why?

...

...

...

12. How easy or difficult was it for you to make annotation?

☐very easy ☐easy ☐normal ☐difficult ☐very difficult

13. Were the results after annotating as expected?

☐Yes ☐No

14. Remarks?

.....

.....

APPENDIX C

The Rete Algorithm

Jess is a rule-based expert system shell. In the simplest terms, this means that Jess's purpose is to continuously apply a set of if-then statements (*rules*) to a set of data (the *knowledge base*).

The typical expert system has a fixed set of rules while the knowledge base changes continuously. However, it is an empirical fact that, in most expert systems, much of the knowledge base is also fairly fixed from one rule operation to the next. Although new facts arrive and old ones are removed at all times, the percentage of facts that change per unit time is generally fairly small. For this reason, the obvious implementation for the expert system shell is very inefficient. This obvious implementation would be to keep a list of the rules and continuously cycle through the list, checking each one's left-hand-side (LHS) against the knowledge base and executing the right-hand-side (RHS) of any rules that apply. This is inefficient because most of the tests made on each cycle will have the same results as on the previous iteration. However, since the knowledge base is stable, most of the tests will be repeated. You might call this the *rules finding facts* approach and its computational complexity is of the order of $O(RF^P)$, where R is the number of rules, P is the average number of patterns per rule LHS, and F is the number of facts on the knowledge base. This escalates dramatically as the number of patterns per rule increases.

Jess instead uses a very efficient method known as the Rete (Latin for net) algorithm. The classic paper on the Rete algorithm ("Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Charles L. Forgy, Artificial Intelligence 19 (1982), 17-37) became the basis for a whole generation of fast expert system shells: OPS5, its descendant ART, and CLIPS. In the Rete algorithm, the inefficiency described above is alleviated (conceptually) by remembering past test results across iterations of the rule loop. Only new facts are tested against any rule LHSs. Additionally, as will be described below, new facts are tested against only the rule LHSs to which they are most likely to be relevant. As a result, the computational complexity per iteration drops to something more like $O(RFP)$, or linear in the size of the fact base. Our discussion of the Rete algorithm is necessarily brief. The interested reader is referred to the Forgy paper or to Giarratano and Riley, "Expert Systems:

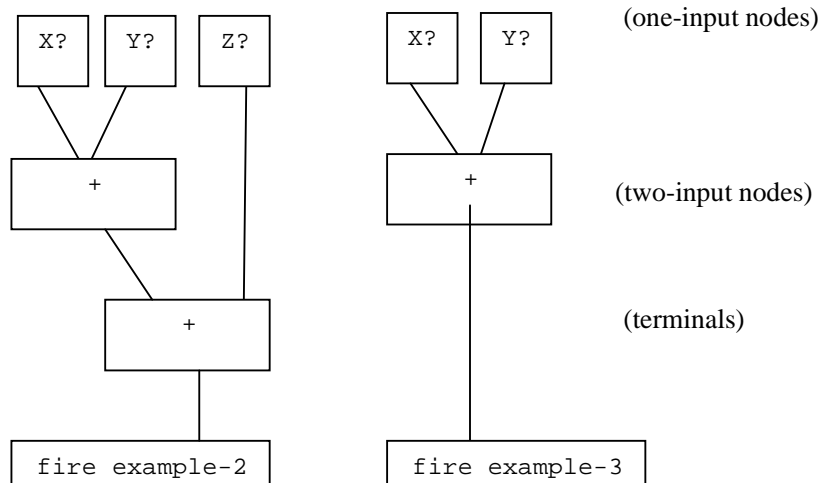
Principles and Programming", Second Edition, PWS Publishing (Boston, 1993) for a more detailed treatment.

The Rete algorithm is implemented by building a network of nodes, each of which represents one or more tests found on a rule LHS. Facts that are being added to or removed from the knowledge base are processed by this network of nodes. At the bottom of the network are nodes representing individual rules. When a set of facts filters all the way down to the bottom of the network, it has passed all the tests on the LHS of a particular rule and this set becomes an activation. The associated rule may have its RHS executed (fired) if the activation is not invalidated first by the removal of one or more facts from its activation set. Within the network itself there are broadly two kinds of nodes: one-input and two-input nodes. One-input nodes perform tests on individual facts, while two-input nodes perform tests across facts and perform the grouping function. Subtypes of these two classes of node are also used and there are also auxilliary types such as the terminal nodes mentioned above.

An example is often useful at this point. The following rules:

```
(defrule example-2    (defrule example-3
  (x)                (x)
  (y)                (y)
  (z)                => )
=> )
```

might be compiled into the following network:

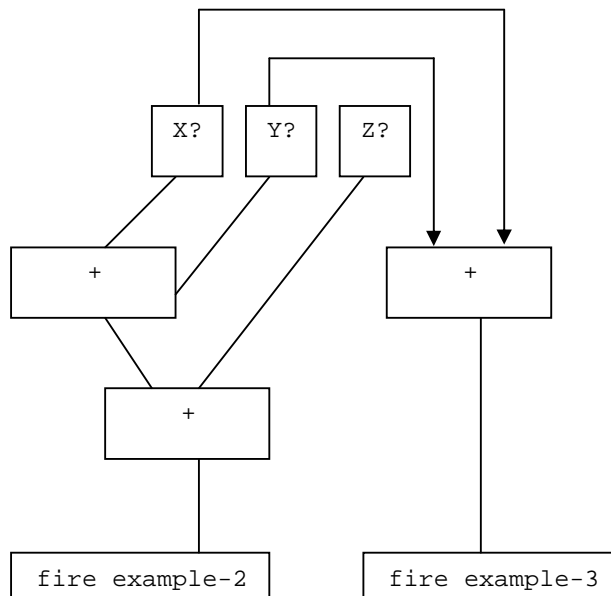


The nodes marked x?, etc., test if a fact contains the given data, while the nodes marked + remember all facts and fire whenever they've received data from both their left and right inputs. To run the network, Jess presents new facts to each node at the top of the network as they added to the knowledge base. Each node takes input from the top and

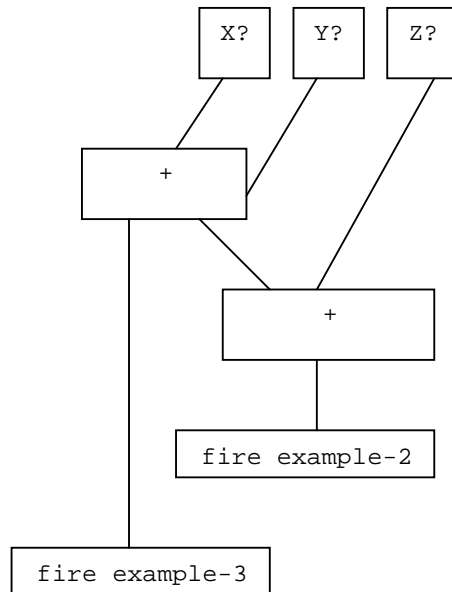
sends its output downwards. A single input node generally receives a fact from above, applies a test to it, and, if the test passes, sends the fact downward to the next node. If the test fails, the one-input nodes simply do nothing. The two-input nodes have to integrate facts from their left and right inputs, and in support of this, their behavior must be more complex.

First, note that any facts that reach the top of a two-input node could potentially contribute to an activation: they pass all tests that can be applied to single facts. The two input nodes therefore must remember all facts that are presented to them, and attempt to group facts arriving on their left inputs with facts arriving on their right inputs to make up complete activation sets. A two-input node therefore has a *left memory* and a *right memory*. It is here in these memories that the inefficiency described above is avoided. A convenient distinction is to divide the network into two logical components: the single-input nodes comprise the *pattern network*, while the two-input nodes make up the *join network*.

There are two simple optimizations that can make Rete even better. The first is to share nodes in the pattern network. In the network above, there are five nodes across the top, although only three are distinct. The second is by modifying the network to share these nodes across the two rules (the arrows coming out of the top of the x? and y? nodes are outputs):



But that's not all the redundancy in the original network. Now we see that there is one joined node that is performing exactly the same function (integrating x,y pairs) in both rules, and we can share that also:



The pattern and joined networks are collectively only half the size they were originally. This kind of sharing comes up very frequently in real systems and is a significant performance booster!

We can see the amount of sharing in a Jess network by using the `watch compilations` command. When a rule is compiled and this command has been previously executed, Jess prints a string of characters something like this, which is the actual output from compiling rule example-2, above:

example-2: +1+1+1+1+1+1+2+2+t

Each time `+1` appears in this string, a new one-input node is created. `+2` indicates a new two-input node. Now watch what happens when we compile example-3:

example-3: =1=1=1=1=2+t

Here we see that `=1` is printed whenever a pre-existing one-input node is shared; `=2` is printed when a two-input node is shared. `+t` represents the terminal nodes being created. (Note that the number of single-input nodes is larger than expected. Jess creates separate nodes that test for the head of each pattern and its length, rather than doing

both of these tests in one node, as we implicitly do in our graphical example.) No new nodes are created for rule example-3. Jess shares existing nodes very efficiently in this case.

Jess's Rete implementation is very literal. Different types of network nodes are represented by various subclasses of the Java class `jess.Node`: `Node1`, `Node2`, `NodeNot2`, `NodeJoin`, and `NodeTerm`. The `Node1` class is further specialized because it contains a command member which causes it to act differently depending on the tests or functions it needs to perform. For example, there are specializations of `Node1` which test the first field (called the head) of a fact, test the number of fields of a fact, test single slots within a fact, and compare two slots within a fact. There are further variations which participate in the handling of multifields and multislots. The Jess language code is parsed by the class `jess.Jesp`, while the actual network is assembled by code in the class `jess.ReteCompiler`. The execution of the network is handled by the class `Rete`. The `jess.Main` class itself is really just a small demonstration driver for the Jess package, in which all of the interesting work is done.

APPENDIX D

Test cases

1- Annotating a hooligan holding a bottle of beer

- **Initialization:** a video frames must be opened.
- **Finalization** Clicking the add button, if there is something went wrong, a popup message giving a warning will appear.
- **Actions:**
 - Select the hooligan button.
 - Type description of the hooligan.
 - Draw using the mouse a rectangle around the hooligan
 - Go to the actions tab
 - Select the icon “holding”
 - Go to the objects tabs
 - Select the beer bottle icon.
- **Input data** description of the hooligan
- **Results**
 - **Expected results** The fighting level will go up.
 - **Actual results** The fighting level went up:

```
f-0 (MAIN::initial-fact)
f-1 (MAIN::humans (h_id 9) (h_is_visible null) (h_makes_sound
null) (h_sound_volume null)
(h_description "white shirt") (h_Framenumber 0) (h_x 121) (h_y
15) (h_width 57) (h_height 95) (h_expression null))
f-2 (MAIN::activity (a_id 1) (a_is_visible null) (a_object 9)
(a_action "hold") (a_subject 806) (a_makes_sound null)
(a_sound_volume null) (a_Framenumber 0))
f-3 (MAIN::objects (o_id 806) (o_description beerbottle)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
f-4 (MAIN::objects (o_id 100) (o_description hooliganclothing)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
f-5 (MAIN::relation (r_id 1) (r_object 9) (r_relation wear)
(r_subject 100) (r_is_visible null) (r_makes_sound null)
(r_sound_volume null))
```

f-6 (MAIN::update-fighting 10)

PASS

2- Saving an XML file

- **Initialization:** a video frames must be opened. Annotation have to be made to the system.
- **Finalization** None
- **Actions** Click on File > Save XML file, Type the file name
- **Input data** File name
- **Results**
 - **Expected results** the file is saved in C:/
 - **Actual results :**

Saving file....

Saved_Test is saved in C:\

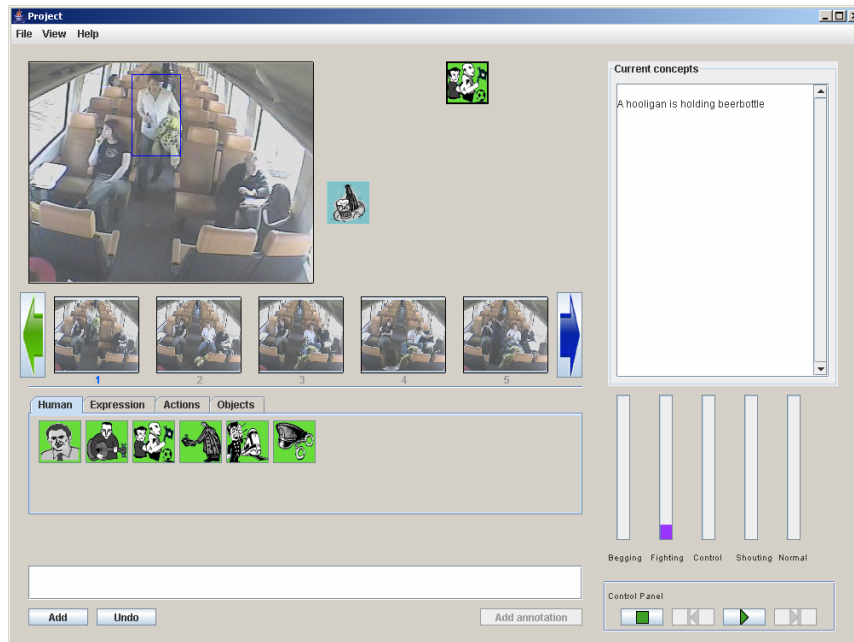
PASS.

3- Loading an XML file

- **Initialization:** a video frames must be opened.
- **Finalization** None
- **Actions** Click on File > Load XML file, Choose the desired file.
- **Input data** Choosing the file, or typing the name of the file to be loaded.
- **Results**
 - **Expected results** the file is loaded into the system and all the annotation and results must be shown.
 - **Actual results :**
 -

Opening: SAVED_Test.xml

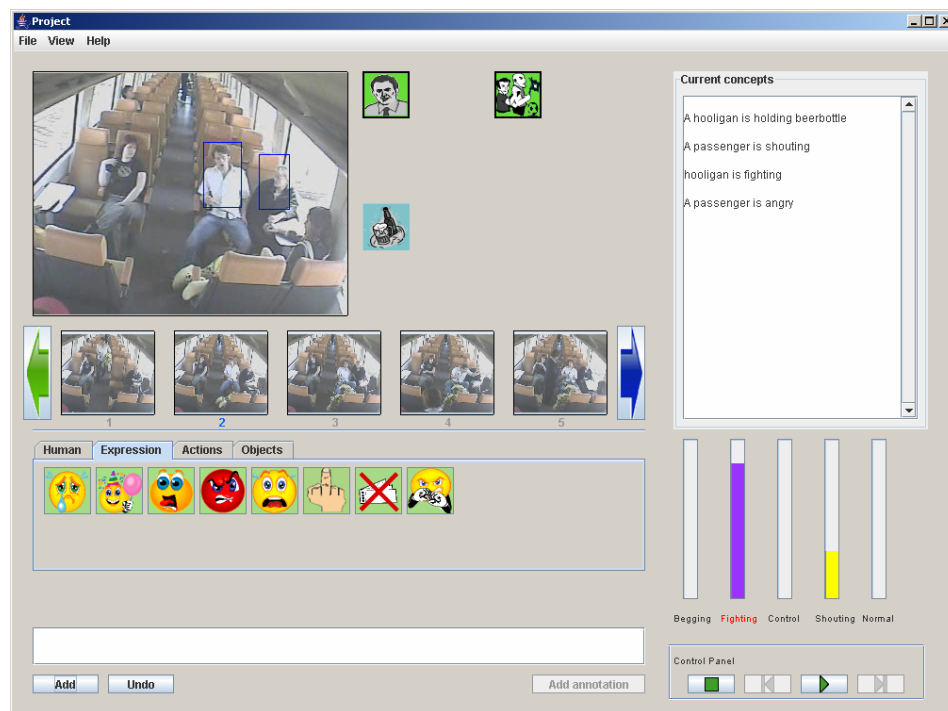
Loading..C:\SAVED_Test.xml



PASS.

4- Threshold for fighting.

- **Initialization:** a video frames must be opened. Annotation has to be made to the system.
- **Finalization** None
- **Actions** Annotate an aggressive situation where the fighting level goes up.
- **Input data** None
- **Results**
 - **Expected results** The fighting level goes up, the text Fighting must flash and a sound of police siren must be played.
 - **Actual results :**



```
f-0 (MAIN::initial-fact)
f-1 (MAIN::humans (h_id 9) (h_is_visible null) (h_makes_sound null)
(h_sound_volume null) (h_description "white shirt") (h_Framenumber 0)
(h_x 121) (h_y 15) (h_width 57) (h_height 95) (h_expression null))
f-2 (MAIN::activity (a_id 1) (a_is_visible null) (a_object 9)
(a_action "hold") (a_subject 806) (a_makes_sound null) (a_sound_volume
null) (a_Framenumber 0))
f-3 (MAIN::objects (o_id 806) (o_description beerbottle)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
f-4 (MAIN::objects (o_id 100) (o_description hooliganclothing)
(o_is_visible null) (o_makes_sound null) (o_sound_volume null)
(o_framenumber 0))
```

f-5 (MAIN::relation (r_id 1) (r_object 9) (r_relation wear)
(r_subject 100) (r_is_visible null) (r_makes_sound null)
(r_sound_volume null))
f-6 (MAIN::update-fighting 10)
f-7 (MAIN::humans (h_id 133) (h_is_visible null) (h_makes_sound null)
(h_sound_volume null) (h_description "") (h_Framenumber 0) (h_x 234)
(h_y 94) (h_width 40) (h_height 52) (h_expression null))
f-8 (MAIN::activity (a_id 133) (a_is_visible null) (a_object 133)
(a_action "shout") (a_subject null) (a_makes_sound null)
(a_sound_volume null) (a_Framenumber 0))
f-9 (MAIN::update-shouting 30)
f-10 (MAIN::humans (h_id 9) (h_is_visible null) (h_makes_sound null)
(h_sound_volume null) (h_description "white shirt") (h_Framenumber 1)
(h_x 181) (h_y 75) (h_width 40) (h_height 69) (h_expression fighting))
f-11 (MAIN::activity (a_id 133) (a_is_visible null) (a_object null)
(a_action "null") (a_subject null) (a_makes_sound null) (a_sound_volume
null) (a_Framenumber 1))
f-12 (MAIN::update-fighting 60)
f-13 (MAIN::humans (h_id 133) (h_is_visible null) (h_makes_sound
null) (h_sound_volume null) (h_description "") (h_Framenumber 1) (h_x
240) (h_y 88) (h_width 32) (h_height 58) (h_expression angry))
f-14 (MAIN::update-fighting 15)

Fighting: 85

Sound Played

PASS.

APPENDIX E

Rules

```
; aggression.clp
; the general rule-engine of aggression level detection
;
; This file contains several general definitions
; that apply to all kinds of aggression. The code contains a
; defclass that is an external address, which
; makes exchange of data between Jess and Java
; possible.
;
; The aggression levels that can be detected are:
;   begging
;   fighting
;   control
;   shouting
;   normal
;
;
; Upon asserting an action, the defclass (a Java-bean)
; will be updated by asserting the corresponding
; score.
;(defclass aggression RunJess.Aggression)
; In the java source, the bean is passed to Jess as follows:
;
;   r.store("AGGRESSION", new Aggression());
;   r.executeCommand("(bind ?s (fetch AGGRESSION))");
;
; Following we can use the following jess commands:
;   (call ?s getAgr)
;   (call ?s setAgr value) and
; as getters and setters for the bean
; This will make sure that the globals will keep their
; values upon issuing a `reset'
;(printout t "The test is succeed" crlf)

(deftemplate humans
(slot h_id)
(slot h_is_visible)
(slot h_makes_sound)
(slot h_sound_volume)
(slot h_description)
(slot h_Framenumber)
(slot h_x)
(slot h_y)
(slot h_width)
(slot h_height)
(slot h_expression))
```

```

(deftemplate activity
(slot a_id)
(slot a_is_visible)
(slot a_object)
(slot a_action)
(slot a_subject)
(slot a_makes_sound)
(slot a_sound_volume)
(slot a_Framenumber)
)

(deftemplate objects
(slot o_id)
(slot o_description)
(slot o_is_visible)
(slot o_makes_sound)
(slot o_sound_volume)
(slot o_framenumber)
(slot o_endframe)
)

(deftemplate relation
(slot r_id)
(slot r_object)
(slot r_relation)
(slot r_subject)
(slot r_is_visible)
(slot r_makes_sound)
(slot r_sound_volume)
)

(defrule update-begging
  (update-begging ?c)
  =>
  (call ?s setBeg (+ (call ?s getBeg) ?c)))

(defrule update-shouting
  (update-shouting ?c)
  =>
  (call ?m setShout (+ (call ?m getShout) ?c)))

(defrule update-fighting
  (update-fighting ?c)
  =>
  (call ?f setFight (+ (call ?f getFight) ?c)))

```

```

(defrule update-fighting2
  (update-fighting2 ?c)
  =>
  (call ?f setFight (- (call ?f getFight) ?c)))

(defrule update-normal
  (update-normal ?c)
  =>
  (call ?n setNormal (+ (call ?n getNormal) ?c)))

(defrule update-normal2
  (update-normal2 ?c)
  =>
  (call ?n setNormal (- (call ?n getNormal) ?c)))

(defrule update-control
  (update-control ?c)
  =>
  (call ?d setControl (+ (call ?d getControl) ?c)))

(defquery object-id-query
  "Look up fact id."
  (declare (variables ?id))
  ?fact <- (objects (o_id ?id)))

(defun modify-object-endframe
  "Modify objects instance endframe"
  (?id ?endframe)
  (bind ?results (run-query* object-id-query ?id))
  (?results next) ; Assumes exactly one objects instance
with ?id
  (modify (?results get fact) (o_endframe ?endframe))
  (close ?results)
  nil)

(batch CLP/fight.clp)
(batch CLP/shout.clp)
(batch CLP/begging.clp)
(batch CLP/normal.clp)
(batch CLP/control.clp)

```

```

;normal.clp
(defrule nomal-rule-1
  (activity (a_action "sit"))
  =>
  (assert (update-normal 20))
  ;(printout t "normal !" crlf)
  )

```

```

(defrule nomal-rule-2
(activity (a_action "hold"))
(objects (o_description mp3player))
=>
(assert (update-normal 21))
;(printout t "normal !" crlf))

(defrule nomal-rule-3
(activity (a_action "hold"))
(objects (o_description doll))
=>
(assert (update-normal 10))
;(printout t "normal !" crlf)
)

(defrule nomal-rule-4
(activity (a_action "hold"))
(objects (o_description bag))
=>
(assert (update-normal 30))
;(printout t "normal !" crlf))

(defrule nomal-rule-5
(activity (a_action "hold"))
(objects (o_description book))
=>
(assert (update-normal 24))
;(printout t "normal !" crlf))

(defrule nomal-rule-6
(activity (a_action "talk"))
(objects (o_description cellphone))
=>
(assert (update-normal 27))
;(printout t "normal !" crlf))

(defrule nomal-rule-7
(humans (h_expression happy))
=>
(assert (update-normal 15))
)

(defrule nomal-rule-8
(activity (a_action "hold"))
(objects (o_description ticket))
=>
(assert (update-normal 10))
(assert (update-fighting2 10))
;(printout t "normal !" crlf))

```

```
; shout.clp
(defrule shout-rule-1
(activity (a_action shout)(a_subject cellphone))
=>
(assert (update-shouting 30))
)

(defrule shout-rule-2
(activity (a_action "shout"))
=>
(assert (update-shouting 30))
)

(defrule shout-rule-3
(activity (a_action "shouting at"))
=>
(assert (update-shouting 40))
)

(defrule shout-rule-4
(humans (h_expression afraid))
=>
(assert (update-shouting 15))
)

(defrule shout-rule-5
(humans (h_expression surprised))
=>
(assert (update-shouting 15))
)
```

```

; fight.clp
(defrule fight-rule-1
(activity (a_action "hold"))
(objects (o_description knife))
=>
(assert (update-fighting 40))
(assert (update-normal2 30))
;(printout t "fight !" crlf)
)

(defrule fight-rule-2
(activity (a_action "hit"))
=>
(assert (update-fighting 30))
(assert (update-normal2 20))
)

(defrule fight-rule-3
(activity (a_action "smoking"))
(objects (o_description cigarette))
=>
(assert (update-fighting 15))
)

(defrule fight-rule-4
(activity (a_action "shout"))
=>
(assert (update-fighting 10))
)

(defrule fight-rule-5
(activity (a_action "hold"))
(objects (o_description beerbottle))
=>
(assert (update-fighting 10))
)

(defrule fight-rule-6
(activity (a_action "hold"))
(objects (o_description beercan))
=>
(assert (update-fighting 10))
)

(defrule fight-rule-7
(activity (a_action "hold"))
(objects (o_description paint))
=>
(assert (update-fighting 10))
)

```

```

(defrule fight-rule-8
(activity (a_action "hit"))
(objects (o_description hooliganclothing))
(relation (r_relation wear))
=>
(assert (update-fighting 25))
)

(defrule fight-rule-9
(humans (h_expression fighting))
=>
(assert (update-fighting 60))
)

(defrule fight-rule-10
(humans (h_expression afraid))
=>
(assert (update-fighting 10))
)

(defrule fight-rule-11
(humans (h_expression angry))
=>
(assert (update-fighting 15))
)

(defrule fight-rule-12
(humans (h_expression finger))
=>
(assert (update-fighting 20))
)

(defrule fight-rule-13
(activity (a_action "is putting legs on"))
(objects (o_description chair))
=>
(assert (update-fighting 20))
(assert (update-normal2 10))
;(printout t "fight !" crlf)
)

(defrule fight-rule-14
(activity (a_action "touching"))
=>
(assert (update-fighting 15))
)

(defrule fight-rule-15
(activity (a_action "hitting"))
=>
(assert (update-fighting 55))
(assert (update-normal2 20))
)

```

```

(defrule fight-rule-16
(activity (a_action "throwing an object at"))
=>
(assert (update-fighting 30))
(assert (update-normal2 15))
)

(defrule fight-rule-17
(activity (a_action "is invading private space of"))
=>
(assert (update-fighting 10))
(assert (update-normal2 5))
)

(defrule fight-rule-18
(activity (a_action "hold"))
(objects (o_description knife))
(objects {o_endframe != null})
=>
(assert (update-fighting2 25))
(assert (update-normal 45))
)

(defrule fight-rule-19
(activity (a_action "hold"))
(objects (o_description beercan))
(objects {o_endframe != null})
=>
(assert (update-fighting2 10))
)

(defrule fight-rule-20
(activity (a_action "hold"))
(objects (o_description beerbottle))
(objects {o_endframe != null})
=>
(assert (update-fighting2 10))
)

(defrule fight-rule-21
(activity (a_action "smoking"))
(objects (o_description cigarette))
(objects {o_endframe != null})
=>
(assert (update-fighting2 15))
)

```



```

; control.clp
(defrule control-rule-1
(activity (a_action "check"))
(objects (o_description ticket))
=>
(assert (update-control 30))
(assert (update-fighting2 20))
;(printout t "control !" crlf)
)

(defrule control-rule-2
(activity (a_action "hold"))
(objects (o_description ticket))
=>
(assert (update-control 10))
;(printout t "control !" crlf)
)

(defrule control-rule-3
(activity (a_action "walk"))
(objects (o_description conductorclothing))
(relation (r_relation wear))
=>
(assert (update-control 10))
;(printout t "control !" crlf)
)

(defrule control-rule-4
(objects (o_description conductorclothing))
(relation (r_relation wear))
=>
(assert (update-control 10))
;(printout t "control !" crlf)
)

(defrule control-rule-5
(humans (h_expression noticket))
=>
(assert (update-control 30))
(assert (update-fighting 10))
)

```

```

; begging.clp
(defrule begging-rule-1
(activity (a_action "hold"))
(objects (o_description money))
(objects (o_description artistclothing))
(relation (r_relation wear))
=>
(assert (update-begging 70))
;(printout t "begging !" crlf)
)

(defrule begging-rule-2
(activity (a_action "hold"))
(objects (o_description money))
(objects (o_description beggarclothing))
(relation (r_relation wear))
=>
(assert (update-begging 70))
;(printout t "begging !" crlf)
)







(defrule begging-rule-2
(activity (a_action "is invading private space of"))
=>
(assert (update-begging 10))
)

```

APPENDIX F






People tab




In the people tab the user can use the following icons to annotate the world model:

	Passenger: this icon is used to all the travelers for a train.
	Artist: this icon stands for someone who sings, act or play music in order to get money from the passengers.
	Beggar: someone who ask for money from passengers.
	Conductor: the one in charge in the train. A conductor checks the tickets.
	Hooligan: a football club fan. Mostly in groups and they are very violent and aggressive.
	Police officer

Expression tab











The expression tab is used to annotate the human facial expression and body language.

	Afraid
	Angry
	Happy
	Sad
	surprised

	Giving the middle finger gesture (body language)
	In fighting gesture (body language)
	No ticket gesture (body language)















Actions tab

The action tab is used to describe an action the chosen human is performing.

	Checking: this icon is used to describe a conductor is checking the tickets.
	Holding: this icon can be used when a human is holding an object.
	Hitting
	Putting legs on: this icon should be used when a human is putting his legs on the seat.
	Shouting
	Sitting
	Walking
	Talking
	Smoking
	Putting: this icon differ from the putting legs on icon, this can be used when putting an object like a book, bag, etc...






Objects tab

The possible objects that can be found in the train compartments are:

	Baby pram
	Beer bottle
	Beer can
	Bag
	Train tickets
	Book
	Cell phone
	Train seat
	Cigarette
	Doll
	Knife
	Money
	Mp3-player
	Paint

	Train table
---	-------------

4.2.5 Relation tab

	Hitting
	Shouting at
	Touching
	Invading private space
	Throwing an object at

Note that hitting is also in the action tab, but there it is used for hitting an object for example hitting a table and here it is used when hitting another passenger for example.

APPENDIX G

```
<?xml version="1.0" encoding="US-ASCII" ?>
<fact-list>
<fact>
  <name>MAIN::initial-fact</name>
</fact>
<fact>
  <name>MAIN::humans</name>
<slot>
  <name>h_id</name>
  <value type="INTEGER">9</value>
</slot>
<slot>
  <name>h_is_visible</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>h_makes_sound</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>h_sound_volume</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>h_description</name>
  <value type="STRING">white shirt</value>
</slot>
<slot>
<slot>
  <name>h_Framenumber</name>
  <value type="INTEGER">0</value>
</slot>
<slot>
  <name>h_x</name>
  <value type="INTEGER">122</value>
</slot>
<slot>
  <name>h_y</name>
  <value type="INTEGER">11</value>
</slot>
<slot>
  <name>h_width</name>
  <value type="INTEGER">56</value>
</slot>
<slot>
  <name>h_height</name>
  <value type="INTEGER">116</value>
</slot>
<slot>
  <name>h_expression</name>
  <value type="SYMBOL">null</value>
```

```

    </slot>
  </fact>
<fact>
<name>MAIN::activity</name>
<slot>
  <name>a_id</name>
  <value type="INTEGER">1</value>
</slot>
<slot>
  <name>a_is_visible</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>a_object</name>
  <value type="INTEGER">9</value>
</slot>
<slot>
  <name>a_action</name>
  <value type="STRING">hold</value>
</slot>
<slot>
  <name>a_subject</name>
  <value type="INTEGER">806</value>
</slot>
<slot>
  <name>a_makes_sound</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>a_sound_volume</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>a_Framenumber</name>
  <value type="INTEGER">0</value>
</slot>
</fact>
<fact>
  <name>MAIN::objects</name>
<slot>
  <name>o_id</name>
  <value type="INTEGER">806</value>
</slot>
<slot>
  <name>o_description</name>
  <value type="SYMBOL">beerbottle</value>
</slot>
<slot>
  <name>o_is_visible</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>o_makes_sound</name>
  <value type="SYMBOL">null</value>

```



```

    </slot>
<slot>
  <name>o_sound_volume</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>o_framenumber</name>
  <value type="INTEGER">0</value>
</slot>
<slot>
  <name>o_endframe</name>
  <value type="SYMBOL">null</value>
</slot>
</fact>
<fact>
  <name>MAIN::relation</name>
<slot>
  <name>r_id</name>
  <value type="INTEGER">1</value>
</slot>
<slot>
  <name>r_object</name>
  <value type="INTEGER">9</value>
</slot>
<slot>
  <name>r_relation</name>
  <value type="SYMBOL">wear</value>
</slot>
<slot>
  <name>r_subject</name>
  <value type="INTEGER">100</value>
</slot>
<slot>
  <name>r_is_visible</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>r_makes_sound</name>
  <value type="SYMBOL">null</value>
</slot>
<slot>
  <name>r_sound_volume</name>
  <value type="SYMBOL">null</value>
</slot>
</fact>
</fact-list>

```


APPENDIX H

User manual

Start the application

Run the program in NetBeans using F6

Watch a video

To watch a video, click on the file menu and choose “Open video”.

Browse and select the desired video file.

Open frames

Click on the file menu and choose “Open frames”.

Browse and select the .txt file. Note that the filename must be named to a folder containing images in the output folder.

Load XML file

Click on the file menu and choose “Load XML file”.

Browse and select the desired XML file. Note that the filename must start with `SAVED_<Name>`

Save XML file

Click on the file menu and choose “Save XML file”.

Browse the desired direction and type in the name and click on OK.

Save Manual levels

Click on the file menu and choose “Save manual levels”

Type the desired name and click on save.

Load Manual levels

Click on the file menu and choose “Load manual levels”

Browse for the file and click open.

Reset

To reset the current session click on the file menu and choose “Reset”

View captured Frames

Choose View – View captured frames

Annotating

Before starting the annotating process frames must be opened.

When this is done, a saved XML file can be loaded to work further on that file or to start a new session.

New session

The annotating can be started by first choosing the desired frame to annotate, usually the annotating start in frame 1.

Second, choose people type:

The choice is between:

Passenger: A normal passenger of a train can be used for male and female.

Hooligan: football supporter that makes troubles.

Beggar: someone who asks for money.

Artist: someone who makes an act (sings) and ask for money.

Conductor: the conductor of the train

Police officer: a police, a cop.

Note that on almost all buttons in the system, there is a mouse over, so e.g. when you move your mouse over the passenger button; you will see “A passenger”

When you click on a people button, you have to enter a small description about the person you choose, and then a rectangle of the location of that person has to be drawn using the mouse.

To draw a rectangle, click and hold the left button of the mouse on the left top corner of the person, move the mouse to the right bottom corner and release the button. The **Undo button** can be used when you draw a wrong rectangle.

The selected person icon will be displayed right to the big frame picture, with a mouse over tip of the description and the frames he is in, figure H.1

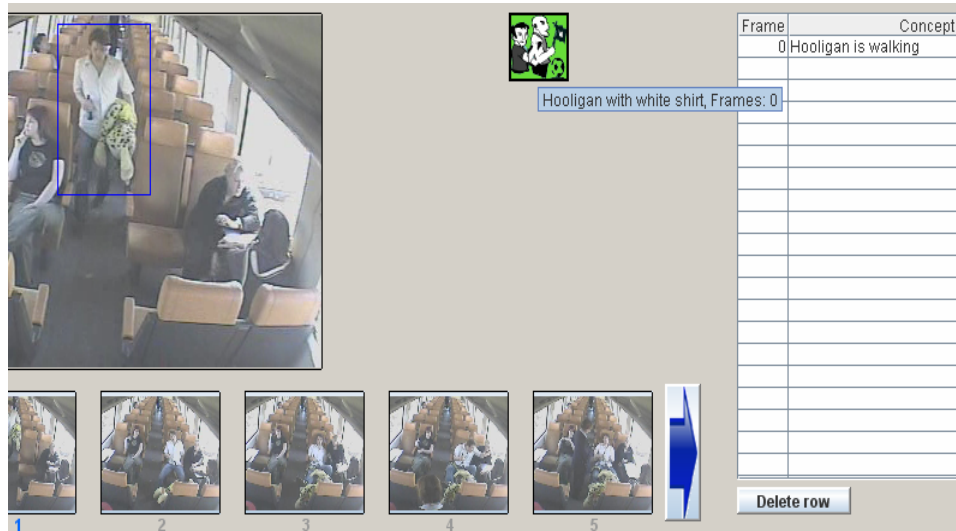


Figure H.1 Example of mouse over tip.

After choosing the people type, a choice can be made between: Expression, Relation and Action.

The syntax is:

<People><Expression>

<People><Relation><People>

<People><Action>

<People><Action><Object>

The program will not allow different choices; everything that is not allowed to be selected is disabled.

Note, when choosing Relation, the second <People> have to be already defined before choosing relation, e.g. first you choose:

- 1- A passenger is sitting.
- 2- A hooligan is invading private space of <passenger you chose in step 1>

When an annotation line is made, the **Add button** must be clicked to add the annotation to the database, the annotation will be shown in the table.

Click on the desired frame to annotate it. Note that you see the current frame number is blue.

The **Left / Right arrows** are used to go to the next 5 frames to the right and go back to the first 5 frames to the left.

When done with annotating the desired frames the **XML** file can be **saved**.

Adding manual levels

To analyze the data a manual aggression levels can be stored and loaded again.

In every frame you choose you can click on the Plus / minus buttons to increase/decrease the level of the progress bars, following that click on **Set Value** button, figure H.2

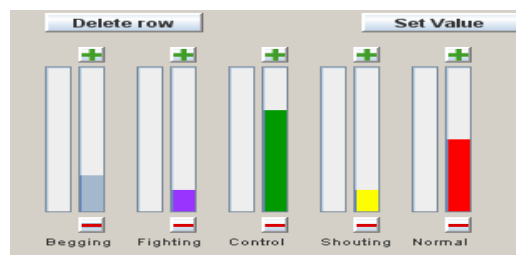


Figure H.2: The manual levels progress bars

After completing the levels it can be **saved**.

The saved file can be **loaded** again when starting a new session.

Loading saved session

After opening the frames, the saved XML file can be loaded.

After loading the XML file you will see all the objects and people that are used in the session. Figure H.3

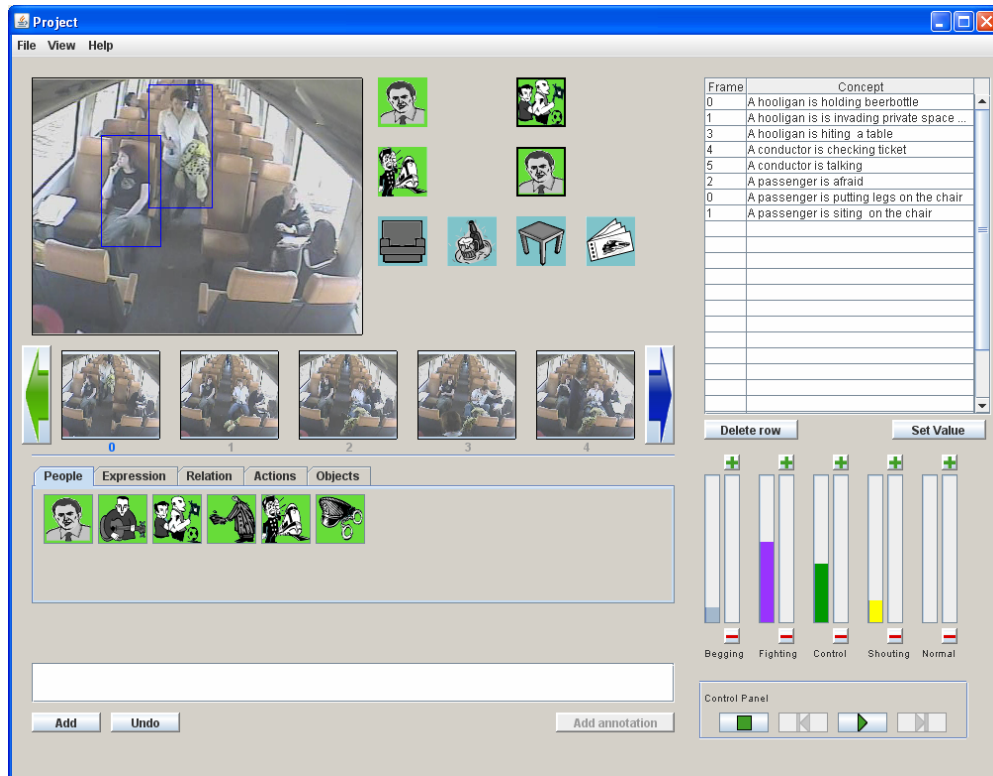


Figure H.3: Loading XML file.

Clicking on different frames will allow you to see what the aggression levels were on that frame.

Play session

After loading a session, you can play the whole session as a scene and see what has been done on every frame. To do that click on the Play button in the control panel, figure H.4

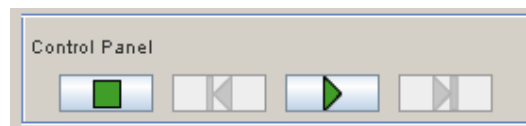


Figure H.4: Control panel play button

When clicking on the play button the session will play after 10 seconds and will start from frame 1 and stop on frame 10, the time between the frames is 10 seconds. During that period the user can click on the **Pause button** figure H.5, to pause the scene.

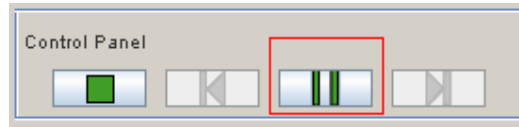


Figure H.5: Pause button

Clicking on the **Stop Button** will stop the play of the scene and go back to frame 0.

After pausing or stopping the scene, the **Forward** and **Rewind buttons** are available, clicking on the forward button will go to the next frame, and rewind button will go back to the previous frame.

Clicking on the button **Add annotation** will allow you to add annotation in the current scene, note that you will have the possibility to choose between all the passengers in the scene.

APPENDIX G

PAPER

VIDEO CONTENT ANALYSIS & AGGRESSION DETECTION SYSTEM FOR A
TRAIN ENVIRONMENT

VIDEO CONTENT ANALYSIS & AGGRESSION DETECTION SYSTEM FOR A TRAIN ENVIRONMENT

Mohannad Ismail

Man-Machine Interaction Group

Faculty of Electrical Engineering, Mathematics, and Computer Science

Delft University of Technology

Mekelweg 4, 2628 CD Delft, the Netherlands

E-mail: md1808@hotmail.com

KEYWORDS

Aggression detection, rule-base system, expert system, train environment, video content analysis, annotating system.

ABSTRACT

In this paper we investigate the behavior of the human in the train. We designed and implemented a system with high usability standards with which users can annotate situations in the train compartments. We are particularly interested in aggression. The input to the annotating process is images that are captured from video data of recorded scenarios of aggressive and non aggressive situation. We implemented a user interface that is connected to a rule-based expert system to handle the incoming data from the annotating process. The output gives an aggression level and an aggression classification.

1. INTRODUCTION

Aggression is a behavior that is intended to threaten or inflict physical injuries on another person or organism; a broader definition may include such categories as verbal attack, discriminatory behavior, and economic exploitation [1]. To increase passengers safely from aggression in trains, the Dutch railways (NS) and the Man-Machine Interaction research group at Delft University of Technology started the Aggression detection project. The aggression detection project focuses on developing a system that detects human

aggression using a multi-modal system that fuses sound and video signals, recorded in train compartments.

We want to investigate/ acquire knowledge / see how an expert detects aggression. To investigate the process of security experts, we created an acquisition tool.

2. AGGRESSION IN TRAIN

The aggression in train compartments can take many different forms. It can be directed towards the interior of the train, the passengers, conductors and train mechanist. The interior can be damaged or painted, and sometimes burned, or written on. Passengers, conductors and train mechanist can be robbed, threatened, intimidated or physically or mentally hurt. The train aggression is causing a decrease in the number of conductors.

The number of intimidating and threatens towards the conductors is growing. In the first 8 months of 2003, the number of aggression cases was more then 15% than a year earlier in the Netherland. The statistics are showing that the aggression problems are getting worse, in 2001 there were only 6.944 incidents and in 2006 that number was 10.500. The aggression is directed in 75% of the cases towards the conductors and within that 75%, is 1 out of 10 incidents include physical violence [2]. About 10% of the aggression is directed towards the passengers. The

Dutch railroad needs at least 10% more conductors and they can't achieve that by not doing anything to the aggression on the trains. NS tried to solve this problem by hiring guards as conductor assistant during rush hours.

The train interior damage is costing the Dutch railroad about 20 million euros on a yearly basis, according to Mr. Schultz van Haegen the state secretary at the Dutch ministry of Water and Traffic. The damage is mainly caused by vandalism and graffiti that are drawn on the interior of the train. The eradication costs of these graffiti cost about 10 million euros. The misuse of the fire extinguishers and the cleaning of the damage afterwards are costing the NS about 5 million [3]. The conclusion is that NS needs an automatic aggression detection system.

3. AGGRESSION DETECTION BY HUMANS

To understand what is needed to detect aggression automatically, it is important to understand how human detect an aggressive situation. Humans are very well able to detect aggression in verbal expressions. The problem here is that speech recognition techniques is still ongoing researches, and there is no technique available that provides enough accuracy to depend on given the large amount of noise occurring in the train. Therefore, we need to identify other aspects that humans use to detect aggression e.g. emotions, shouting and location. Humans detect aggression easily when he/she knows the environment, e.g. when a human sees two people fighting in a karate school, he/she knows this is part of the sport and its probably not a form of aggression, but when he/she sees two people fighting on a train, then this means it is a form of aggression, another example is when he/she sees someone smoking on the street he/she does not think about aggression, and that is different when someone is smoking in the train, this might lead to aggression situation since smoking is not allowed on the train. These examples imply that the environment where the aggressive or non aggressive situation arises is very important. In other

words, the context of the situation is very important.

The human does not only use the environment to detect aggression, the human detect objects such as a knife, but simply detecting a knife is not enough, e.g. a person holding a knife is not enough, the human identify the activity the person holding the knife, the person could be holding a knife to peel an apple which isn't an aggressive situation, but holding a knife when the person is angry is a different situation. In the examples above we used the word activity and we refer it to peeling an apple, this means that the human have to know the activity to detect aggression. Also, the word angry is used which is an emotion.

Another example of how human detect aggression is when a human sees a person holding a gun, then he does not immediately sees that as an aggressive situation, e.g. a police officer with a gun is not an aggressive situation, in fact it is a safe situation since people tend not to do an aggressive acts when a police officer is around. Although a passenger or a hooligan holding a gun is referred to be very aggressive situation. This implies that knowing the people plays an important role in the human aggression detection.

Another important issue is that humans have memory, and they can easily memorize the changes in time e.g. a person was drinking beer ten minutes ago in the train and they saw him shouting at others, later they see that person again and sees him walking towards them, they immediately remember him and they detect a possible aggression.

Thus, the human reasons about aggressive situations using the knowledge of the environment, the people types, objects, the activity, the emotional state of the person doing the act of aggression and finally they use their memory to identify people [4]. A computer system needs the same knowledge about the environment to reason about aggression. For that reason a world model is needed. The context of that world model should be the same set of

information the human uses to detect aggression, which is people types, activity, objects, relations and emotions.

4. AGGRESSION DETECTING SYSTEM

A good way to start the analysis about what is needed in the system is to sketch what a typical annotating session will look like.

The user needs to see the video file he will annotate, this means that the system must be able to open a video file. Since the annotation will be on static images (not moving video) the user must be able to load a set of frames (pictures) that were captured from a video file. The annotating process can be started now. During annotating the user should be able to select different kinds of people types, objects, emotions and relations. Because the train environment is dynamic and objects appear at one time and disappear at other, the user should be able to remove objects.

After that the user should be able to save the work that has been made, and because there is a save function, there must be a load function as well. When loading a file, the user should be able to play the file and watch what was annotated on each frame, and he/she should be able to pause, stop, forward and rewind between the frames.

5 AN ANALYSIS OF THE TRAIN COMPARTMENT

The environment we will be focusing on is the train compartment. In this environment we have specific objects such as seats, pathways, windows, doors and objects that are brought by humans, and we also have different type of humans that can be found in a train. Humans such as normal passengers, conductors, police, hooligans, and beggars play a specific role e.g. traveler, controller, and information provider. Occasionally unwanted behavior arises for instance, aggressive behavior towards conductors or other passengers, demolition of objects. Our goal is to design an aggression detecting system by multi-modal camera's which are able to observe their environment, communicate with the main computer and reason about

the observed data, and conclude an aggression level as an output.

In this project we chose to use a rule-base expert system as a reasoning model, for the reason that we can represent the situation in a train compartment and put the facts in the knowledge base by annotating the situation.

The expert system needs a model for the representation of the situation in the train compartment. Therefore, we have to define which objects can be found that play a role in aggression. The world is dynamic, this means that objects can appear in one certain moment, and disappear in another. As a result the reasoning mechanism has to use a database to store all objects, people and relations in a database, and of course the time where objects and people appear or disappear will be stored as well.

The static context

In order to represent the environment to the expert system, we must identify the objects within the static environment that can play a role in aggression. The static objects are objects that are always in the train compartment. The objects we have found are: train seat and train table.

The dynamic context

The dynamic objects are objects that are moving, in other words, objects that are not always on a train compartment. The dynamic objects in this environment can be split into two categories, the first one is objects that are brought by people onto the train, and the second one is the people who are on the train.

Objects that are brought by passengers: bag, cigarette, beer can or beer bottle, book, baby pram, knife, tickets, cell phone, doll, Mp3-player, money and paint.

People on the train: conductor, passenger, beggar, artist and hooligan.

Activities & behaviors

We analyzed the actions that can be made on a train, and we found: sitting, smoking, talking, walking, hitting, checking tickets,

putting, holding, shouting, and putting legs on the seat. Combinations between these actions and objects that are found, aggression situation can be created, like:

A hooligan is holding a knife, < *very unsafe situation* >
 A hooligan is holding paint, < *damaging the interior* >
 A passenger is smoking a cigarette, < *may cause a fight* >

6. ARCHITECTURE

The system has four main components (figure 1). The GUI allows the user to interact with the system. All the user commands will go through the GUI. The user interface contains facilities for the user to perform all the actions that were discussed earlier. The Validator validates the annotations resulting from user actions. Only valid annotations are allowed to the reasoning component, invalid annotations are rejected with a message. The Validator sends the valid annotations as facts into the reasoning part, where Jess will reason about the aggression and send the output back to the GUI, where the output is displayed. The XML/IO is used for the saving and loading the XML file. It is also used for generating JPEG images of the locations of the people or objects as specified by the annotations. Playing a video file of the scenario to be annotated is also the work of the XML/IO component.

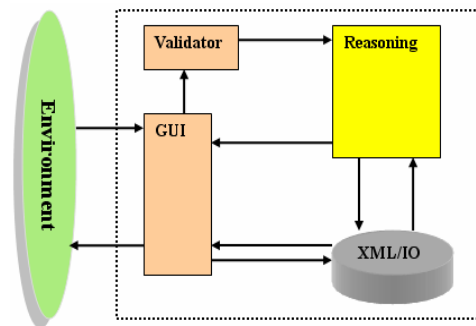


Figure 1: System architecture showing the different components and how they are related

7. REASONING

We defined five scripts for the aggression levels: normal, fight, shout, control and begging:

Normal: The normal level is when everything on the train is normal, that is when there is no shouting, begging or fighting.

Fight: A fight will occur when 2 or more people are arguing.

Shout: When one or more persons are screaming or talking very loudly the Shout level will occur.

Control: When the conductor is checking the tickets this will be the result.

Begging: When a beggar is asking for money, begging will be the result.

The reasoning system works with numbers, the higher the number the most likely it is the correct case (Normal, Fighting, Shouting, Begging or Control). Figure 2 shows how the reasoning system works.

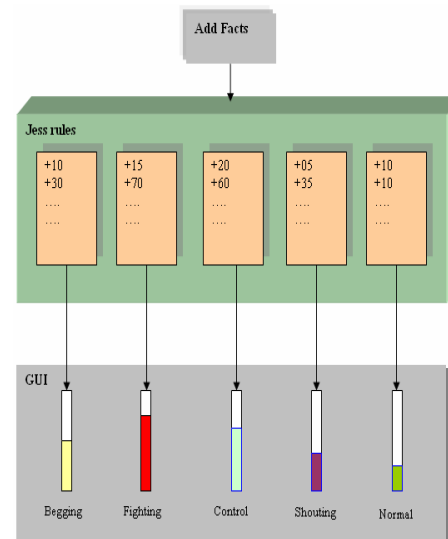


Figure 2: The reasoning system

8. IMPLEMENTATION

The scripts have rules, when the user is annotating he/she will assert facts into the Jess database and from there a result will be given. Those rules are similar to CLIPS rules. There is also a script defined which provides the methods that are used in the five scripts described above.

The below rule will add 60 to the fighting level when “a beggar is holding a knife” is annotated:

```
(defrule fight-rule-100
  (activity (a_action "hold"))
  (objects (o_description
    beggarclothing))
  (relation (r_relation wear))
  (objects (o_description
    knife))
  =>
  (assert (update-fighting
    60))
  )
```

Figure 3 shows the designed user interface. We used icons to present the people types, objects, emotions and relations. The use of icons makes it easy to construct annotations that can be easily validated due to the fact that we limited the options in the annotation process in such way that the annotations are error free of syntax.

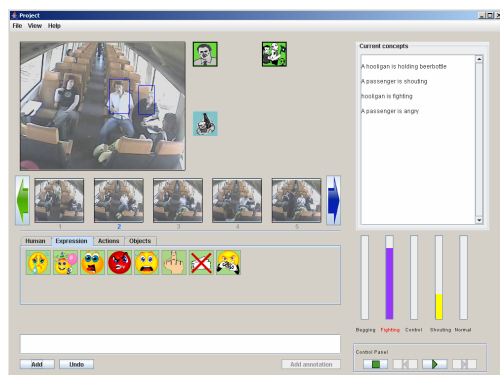


Figure 3: The user interface

9. TESTING & EXPERIMENT RESULTS

The testing is carried out by students. For our experiment we had 5 test subjects, all of the 5 subjects are students from the MMI group. We tested different scenarios. A task list was giving where testers have to complete a certain series of tasks using the system. The first few tasks that were given were relatively easy to do, and everything was explained exactly, but later on the tasks were open and the user can do what he/she thinks is the right action, this

is done so we can measure the learn-ability of the system. After finishing the test, the users have to fill in a debriefing form, where we ask the testers on their opinion regarding the system, and to give a grade about several things like (difficulty, user friendly). We also asked the testers to give us three things they like and dislike about the system. This is done to get a better idea about how the testers think about the system and the usability of it.

The entire test subjects were able to accomplish all the tasks during the test. Five different scenarios of aggressive and non aggressive situation on the train could be annotated using the system. The aggression that was detected by the system was very close to the aggression level that testers gave to the annotated scenarios. The average grade that was given by the test subjects about the overall system was 7.8.

10. CONCLUSIONS & RECOMMENDATIONS

People in trains can show aggressive behavior. A multimodal camera monitoring the train can recognize this behavior by applying knowledge for people on a train and the objects that are on the train, either as interior of the train or brought on the train by passengers. With the large variety of objects, it is hard to formulate rules that apply to all objects or to classify all objects.

The multimodal camera needs many skills for its task of identifying aggression: it should recognize people as they move on the train, identify groups of people, i.e. people that belong together, identify gesture, body language, and recognize speech. All these qualities are necessary to properly identify aggression.

The final goal was to develop a model to annotate a video sequent of images and to detect the level of aggression.

Dynamic Bayesian networks are probably the most appropriate approach, since we will be able to use pattern recognizing to classify aggression, based on either a priori knowledge or on statistical information extracted from the patterns

[5]. But because of its simplicity and availability of data we use a rule based system as a first approach. It would be nice to know whether Dynamic Bayesian networks will be more accurate and more efficient than using the expert system and a rule based approach that is used in this report. It would be clever if we could compare both outputs in one system.

[57] M. Pantic and L.J.M. Rothkrantz, "Expert system for automatic analysis of facial expressions" Image and Vision Computing, vol. 18, no. 11, pp. 881--905, 2000.

REFERENCE

- [47] Sci-Tech Encyclopedia: Aggression, <http://www.answers.com/topic/aggression>, last visited at September 25, 2007.
- [48] Agressie in de trein neemt toe, Algemeen Dagblad, 1 november 2003.
- [49] Spoorvandalisme kost zeker 20 miljoen per jaar, <http://www.home.nl/nieuws/binnenland/artikel/00114122>, Last visited at September 25, 2007.
- [50] Robert A. Baron, Deborah R. Richardson, "*Human aggression*" Second edition, ISBN: 978-0306444586, Plenum Publishers, Nov 1993
- [51] J. Russell and J. Fernandez-Dols, "*The Psychology of Facial Expression*", New York: Cambridge Univ. Press, 1997.
- [52] Sergios Theodoridis, Konstantinos Koutroumbas, "Pattern Recognition, 3rd Edition, ISBN 0123695317, February 2006
- [53] P.Klahr and D.Waterman, "*Expert systems techniques, tools and applications*", Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1986
- [54] S. Fitrianie and L.J.M.Rothkrantz, "*An Icon-Based Communication Tool on a PDA*", Euromedia 2005, volume 11, pages 83-90, ISBN 90-77381-17-1, 2005
- [55] P.Schooneman, "*ISME – Icon based System for Managing Emergencies*", Master Thesis, TU Delft, 2005.
- [56] Sabrina Sestito, Tharam S. Dillon, "*Automated Knowledge Acquisition*", Prentice Hall of Australia Pty Ltd, 1994.