

Designing and implementing a low cost security system



Designing and implementing a low cost security system

Student: Nicky Vermeere

Studentnr: 1540666

Supervisor: H. Karssenberg

Acknowledgements

This paper would not have been possible without the help of many people.

I would like to thank my supervisor, Henk Karssenberg, for giving me the opportunity to do this final project and guiding me through the process.

I also would like to thank my internship company, Verybusy s.r.o and its staff and especially, my manager, Pieter van de Vijver, for always helping me and giving me support, ideas and energy when I needed it.

Special thanks go to Matt Rees and my brother Wesley Vermeere for proofreading my paper and giving advice.

Lastly I would like to thank my parents who always support me and help me in any way they can.

Nicky Vermeere

Management summary

The project assignment was to design and implement a security system that uses low-cost solutions. This security system will eventually be part of a bigger software package. The system needed to still perform basic tasks when the power was cut.

The first step was to research hardware, and research software that was already available. Also necessary security for the system was researched.

Hardware:

Choices were made for using Wi-Fi webcams to capture the images, with a battery operated back-up power supply. A laptop was chosen on which the system would run because it can still function when there is no power supply.

Software:

Different software packages were selected to take over some responsibilities for the system. The software used is the Java Media Framework, to control the video streams. An Apache webserver will be used to host the webpage representation of the system. Code from the Lejos Vision System will be used to detect motion. A Samba server, that already is deployed in the business, will store the recorded images and videos. These files will also be available by making use of an ftp server named proftpd. To send mail the JavaMail API is used.

Security:

The local system, as well as the access to the webcams, will be password protected. The e-mails sent will be encrypted using OpenPGP and the webpage will be encrypted using the HTTPS protocol.

The second step was selecting a software architecture. The Architecture Business Cycle was used to create and select this software architecture. This method guides the designer from an initial list of requirements to an architecture.

The functional requirements for the system were defined as following:

- The system has to connect to multiple (Wi-Fi/IP) cameras to ensure that multiple parts of a space can be monitored.
- The system has to be able to detect movement.
- When an intruder is detected video and images have to be recorded.
- The system must be available from the internet through a webpage
- The images have to be sent through mail and SMS.
- Stored video and images have to be available online through a download link.
- A physical alarm has to be started.
- All events must be logged.
- The system needs a timer to delay the start of the physical alarm with 5 minutes to give the administrator the chance to log in and start or stop the alarm.

The non-functional requirements were defined as following:

- The system has to be low cost.
- The server computer and at least one router and one camera need back-up power. In case the power is cut the system would be still functioning.
- The system needs to be modifiable. In the future the system will be implemented in a bigger system.

After the functional and non-functional requirements were defined. They were made visual by using the “4+1 View”. This “4+1 View” contains different viewpoints of the system. The system needed to be modifiable and needed to connect to multiple cameras. The choice was made to use an Object Oriented architecture. To support the modifiability of the system, also the choice was made to make use of Software Patterns. These patterns are documented solutions to specific and recurring software problems.

The third step was designing the system. All the patterns used have been worked out and code has been written. Also code has been written for reading the video streams and using these video streams to detect movement. Then the implementations of other requirements, as recording images, sending of mail writing the log file and using a timer delay were written. A design was made for the webpage and the user interface was created.

By using the Architecture Business Cycle to select the architecture and to make use of Software patterns the system has become a flexible solution to the project assignment. There are still some improvements possible. The movement detection can be optimized and the handling of light interference can be improved. The created system is very flexible and can be easily integrated in the bigger software package that the business will create.

Contents

Acknowledgements	3
Management summary	4
Preface	7
1. The business and the project	8
1.1. The Business	8
1.2. The Project	8
2. Research of hardware, software and encryption	9
2.1. Research of available developing tools	9
2.2. Research of useable software	9
2.3. Research of required hardware	11
2.4. Research of encryption possibilities	13
3. Selecting the architecture	14
3.1. What is software architecture	14
3.2. The Architecture Business Cycle	15
3.2.1. Creating the business case for the system	15
3.2.2. Understanding the requirements	15
3.2.3. The 4+1 View	17
3.2.4. Creating and selecting the architecture	25
4. Designing and implementing of the system	26
4.1. Software Patterns	26
4.2. Reading the video streams	33
4.3. Motion detection	35
4.4. Recording images and video	37
4.5. The webpage	39
4.6. Sending of mail	40
4.7. The log file	41
4.8. The timer	41
4.9. The user interface	42
5. Conclusion, recommendations and evaluation	43
5.1. Conclusion	43
5.2. Recommendations	43
5.3. Evaluation	43
Bibliography	44
Appendices	45
Appendix A: The Logical View - Sequence Diagrams	
Appendix B: USE case diagrams	
Appendix C: User interfaces	

Preface

The subject of this paper is creating and developing a security system for the Linux O.S. It describes the steps from system requirements to an architecture to the actual system. The system will eventually become part of a bigger package. The all-inclusive software package will include a cashier system and a bookkeeping system and will be integrated with Google docs. This package can be sold to other companies on a low cost basis.

The security system to build will use casual (Wi-Fi) web cameras, motion detection software, Wi-Fi routers and a mobile device. The hardware used has to be purchased on a low cost basis so an affordable security package can be developed and put on the market.

The paper consists of five parts. In part one the business and the system will be described globally. Part two describes what kind of software is available to develop and support the system, what hardware is required and how the system can be encrypted. Part three explains how and what software architecture is chosen and why this software architecture fits the system. Part four will go deeper in the design and implementation of the system. Finally, part five contains the conclusion, recommendations and evaluation.

1. The business and the project

1.1. The Business

Verybusy s.r.o. is a company located in the city of Brno, Czech Republic. The owner, Pieter van de Vijver, moved from the Netherlands to the Czech Republic four years ago and finished his bachelor studies at a Czech company. After receiving his bachelor's diploma, he stayed in the Czech Republic. As a former international student he noticed some things missing, like a decent launderette, in the city of Brno. On 16.04.2007 he founded Verybusy s.r.o. and immediately launched CLUBWASH::Brno, a multilingual laundry-internet café.

Since then the shop has expanded to become an international cultural center for students. Verybusy s.r.o. is in regular contact with the Brno universities and has started to expand its IT development. At the moment, the first projects have already been initiated, including multilingual website development and a security system. In the future, the business wants to create an all-inclusive software package for startup companies. It will contain a (cash) registry system, a bookkeeping system and various other systems. The security system will become a part of this package.

Next to these IT projects, Verybusy s.r.o. is also active in Roma support projects and cooperates with Masaryk University starting up a student radio station. Gypsy children get regular music lessons and are learning English by playing music and singing along. Twice a week there are pub quizzes held, one for all people and one specially made for the international students. Every week there are poker tournaments and there are regular theater performances and concerts, as well as seminars and exhibitions.

1.2. The Project

The company wants to develop a security system that uses low-cost solutions. The building it will secure is located in the center of Brno. It has one main entrance that leads to a front room of six by six meters. One camera needs to monitor this area. Then there is a hallway that leads to the back hall. The hallway also needs to be monitored with one camera. The main hall is about fifteen meters long and seven meters wide and needs to be monitored with one camera. The system will be in a locked office space behind the hall.

When an intruder is detected, the alarm should go off (alarm bell/light) and an online system would log this action, record pictures and send a message including these pictures to the mobile phone of the system administrators. The main system contains the software that checks the environment for intruders and it will be available online to the system administrators. Access has to be provided through a webpage that has a secured connection. In case the main system is cut off from power, a back-up generator takes over the power supply.

The student will have the task of creating and designing the system. The student has the function of software engineer and needs to document the steps taken to get to the design and implementation of the software system.

2. Research of hardware, software and encryption

2.1. Research of available developing tools

To build the security program, there will be some tools needed to develop the system.

The program will be written in Java. Java code can be written in any text editor, but for a more flexible solution it will be helpful to use an Integrated Development Environment (IDE) like Eclipse. This tool is available for Linux and can be used freely.

To create the diagrams in this paper, the program Software Ideas Modeler is used. This program has many options for creating different software diagrams. It supports drawing use-cases, sequence diagrams, component diagrams and many others.

To write this paper, the Microsoft Word program is used.

2.2. Research of useable software

The system will consist of different parts. There will be the detecting of movement, storing images, sending e-mails and making the whole system available only to system administrators. It is useful to look into already available software to take over some parts of the system. Using software that is already available can make the system perform faster and more securely, as long as it holds up to the quality standards necessary for the system.

As mentioned above, the system will be programmed in the Java language, so the Java Development Kit is needed. This package contains the Java compiler and can interpret the java code that will be written.

The cameras connected to the system need to be manipulated. The Java Development Kit has some ways to deal with this but is not flexible and has a lot of shortcomings. For handling the different camera and audio streams it is necessary to use a different solution. The Java Media Framework is a package from the creators of Java (Sun Microsystems) and is specifically designed for handling media with Java. Although it is not considered a perfect solution¹ and hasn't been updated since May 2003, it is still very usable. The Java Media Framework has multiple methods to handle video, audio and images and can be easily integrated in the project.

An alternative for the Java Media Framework is an open source project called "Freedom for media in Java"². This project is compatible with the Java Media Framework and has expanded functionality. However, this project also has not been updated since 2007.

In this project the choice has been made to use the Java Media Framework. The JMF has all functionality needed for the system. Additionally, there is a lot of documentation and there are a lot of sample programs available that make it easier to tackle problems.

The code from the "Motion"³ project can be used to integrate movement detection in the system. It makes use of webcam images and tracks if there is any movement in the pictures. The program is only available for the Linux environment. It is a command-line-based tool. It is possible to "feed" the streams to the "Motion" program and get results returned to the security system. The program can detect the movement and return the detected signals to the main program. Although it is a very promising program, it is too complicated and too inflexible for this project at this time.

¹ http://www.oreillynet.com/mac/blog/2005/12/jmf_a_mistake_asking_to_be_rem.html

² <http://fmj-sf.net/index.php>

³ <http://www.lavrsen.dk/foswiki/bin/view/Motion/WebHome>

The Lejos Vision system is a system for the Lego MindStorms Robotics Invention System. "It provides motion detection, color detection, light detection and interfacing with robots that use the Lego Mindstorms RCX brick, to allow Mindstorms robots to respond to what they see".⁴ A part of this project code contains motion detection. This code is very clear and will be implemented in this system.

When an intruder is detected, a video will be recorded locally and pictures will be sent to the administrator. The files stored locally will be accessed through a server running on the local computer. The server of choice in this case is a Samba server that is already deployed in the business. The Samba server has options for file sharing and has authentication and authorization capabilities. Multiple users can be created and the stored content can be accessed within the local network. To access the content via FTP, the "proftpd" program will be used. This program is also already deployed in Verybusy s.r.o. This program is easy to configure and has security possibilities built in. Folders can be shared and secured and made available via the FTP protocol.

The online storage of files will consist of sending e-mails with attachments. In these e-mails there will be data of the event, images and direct links to the recorded video and access to live video. Because part of the data is stored locally the system itself needs to be secure, therefore, it will be in a locked environment.

The system needs to be available via a webpage, so a webserver is necessary. One of the most common and most used webserver is the Apache webserver. This server is also available for the Linux Operating System.

When an intruder is detected, the system needs to send an e-mail to the administrator with the images. The JavaMail package from Oracle is a platform and protocol-independent framework that has methods to send the necessary e-mails.

To encrypt the e-mail, the Bouncy Castle Crypto package provides an API for Java to decrypt and encrypt messages.

⁴ <http://homepage.ntlworld.com/lawrie.griffiths/vision.html>

2.3. Research of required hardware

The system needs to be able to handle the camera streams and to manipulate the streams (compare and calculate changes). It also needs to be able to send messages and receive incoming calls. For all these qualities, it is necessary to have good and near flawless functioning hardware.

Bandwidth Qualities

The system will handle multiple webcams. One USB webcam will be used to check the direct environment of the computer running the system. All the other webcams used are network cameras. As a result the system doesn't need powerful USB hubs. To accommodate the network cameras, the system needs to be equipped with a decent network card and network connection. To connect to the network, the system should be able to connect to a router by cable or Wi-Fi. A standard 100Mbit card would be sufficient for this system. The routers can be basic routers with wireless possibilities.

Webcam Qualities

To detect intruders the system will make use of low-cost webcams. The webcams need to have a resolution that gives a clear view of the area when it is fully lighted. For streaming the resolution will be reduced in favor of using the bandwidth as efficient as possible.

The standard resolution for the local system will be 640 x 480 and will have a frame rate of 15 fps (frames per second). For online viewing the resolution will be reduced to 320 x 240 and will have a frame rate of 15 fps. The webcams need to be accessible even after the power is down, so they will need to have a back-up power supply and back-up network access. Most Wi-Fi webcams make use of a 12V power supply, so a battery that delivers this power can be connected as a back-up power supply, similar to smoke detectors or emergency lighting.

The network cameras used should be able to connect via LAN and Wi-Fi. When a connection with a router gets lost, the cameras should be able to find the next router in line and connect to that router. Most of the network cameras available at this time have that option.

Processor Qualities

The streams will be read into the program and a picture will be taken. Then every picture from the webcams will be compared with the previous picture taken. This calculation asks for some processing power. A standard 1 GHz computer will be sufficient to handle these calculations.

Network Qualities

The security program will run on a dedicated server computer that will have a connection with the Local Network and the internet. When an intruder cuts the power the system would still need to record the events and have a connection with the "outside world". Therefore a laptop is the most logical choice. When there is no power, the internet connection will also be down. A solution to this problem is to use a USB-stick with a dial up connection to the mobile internet. This will make sure the system can still fulfill its basic tasks. The laptop battery makes sure the system is still up and running for some time and the system can still send a message to the "outside world" to notify that the power is down and record possible intruders.

Infrastructure Qualities

The system will be placed on the premises and communicate with all the devices via a Local Area Network (LAN). Multiple routers will be used for connections between the server and network cameras. When the routers don't get any power it will be impossible to connect to the webcams. To deal with this a dedicated router will be placed on a central location so it covers multiple webcams. This router can take over the connections in case of a power failure. This router will also have its own back-up battery operated power connection.

Storing Qualities

When an intruder is detected the system stores video and image files to the server computer. For this reason the server computer needs to have a storing capacity to handle this. Every recorded event will record until 5 minutes after the event is detected. The amount of available disk space necessary will be a maximum of 20 Gb. Almost all computers have this space.

Mail system

To notify the system administrator when there is an intruder, the system needs to send a message. The most convenient way to do this is by sending an e-mail. In this e-mail there will be information about the event and pictures of the event. Every five minutes a new e-mail will be sent with new pictures. This will continue as long as there is activity in the building. The security program can implement code from the JavaMail software package described in the previous chapter. This package is capable of sending e-mails.

It is also helpful to send a message to the mobile phone of the administrator so he or she is informed quickly and can take the appropriate actions necessary. This can be done by using a service to send an SMS or MMS with pictures.

Alarm bell

When an intruder is detected, a physical alarm has to be triggered. A simple bell in a central location will be placed in the business.

2.4. Research of encryption possibilities

Risks of failing security:

One of the risks is that the video streams can be recorded and distributed, this is a big privacy problem. People that don't have anything to do with the business should not have access to images and video monitoring the business. When they have original recordings they can, for instance, use these videos to override and replace the network camera streams. The system will never be able to detect that there are intruders, because as far as it is concerned, nothing has changed.

When people can login to the internet part of the program, it will be very easy to turn off the system and stop security completely.

What security will be used:

The local system will be password protected. The password will also be stored locally. To ensure the password is not readable for users that are not allowed to start the system the password will be hashed.

The network cameras are also password protected. Most network cameras have a web interface where all the security features can be set up. In this case, one of the cameras used is an Air Live WL-1200CAM that has a web interface where it is possible to enter a password for access to the camera.



Fig 2.1 Password page Wi-Fi camera

The e-mails sent when an intruder is detected will be encrypted using OpenPGP (Open Pretty Good Privacy). This technique makes use of public key encryption. A private and a public key will be generated. Data encrypted with the public key can only be decrypted by the private key. The Bouncy Castle Crypto package provides an application programming interface for Java.

When the administrator wants to access the system from outside the business, a webpage will be used. This webpage will be encrypted with the HTTPS protocol (Hypertext Transfer Protocol Secure). This protocol is a combination of the standard networking protocol for the World Wide Web and the SSL/TLS (Secure Sockets Layer/Transport Layer Security). This protocol creates a secure channel over an insecure network (the internet). As mentioned in the part about useable software, an Apache server will be used to host the web services. SSL makes use of public key encryption. Apache also makes use of certificates; when these certificates are signed by a trusted Certificate Authority (CA) it ensures that the certificate holder really is who he claims to be. In this case there will be a certificate used, but it will not be signed by a CA.

3. Selecting the architecture

3.1. What is software architecture

The IEEE⁵ Architecture Working Group (P1471)⁶, the Recommended Practice for Architectural Description, has established the following definition of architecture and related terms:

“Architecture is the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.”

An architect thinks of the principles a system has to be built on. What are the fundamental rules the system has to comply to? These fundamentals create the basis for all the following steps. Although the word fundamental is a clear word, it doesn't mean anything to a system as long as you don't know to whom or what it is fundamental. A locksmith and a burglar are essentially doing the same thing: they pick locks and expect to get “paid” for their work, but only of them has the fundamental rule to do it the legal way. So fundamental rules have to apply to the system and to the environment where the system is going to be placed in.

A software architecture reveals how software should be structured. It can be referred to how a building is structured. First you put down the foundation, and on that foundation you build the rest of the house. Every building block of the system will be structured based on the selected architecture.

Selecting an architecture is not always solely based on requirements. When you have two different architects and give them the same requirements, chances are they won't end up with the exact same architecture. There can be many factors outside the system that influence the architecture. An organization can have a big influence on the architecture. When a company decides to do a big project, they may want to hire specialized partners that will make part of the system's functionality. This has an influence on the architecture. It is much more likely that the architecture that will be chosen has the capabilities to divide the system in multiple interacting parts. The outsourced parts will be fit in the system, and all of this would only be possible because the architecture chosen supports this.

An architecture is helpful in communicating with the stakeholders of the project. Because an architecture is an abstract view of the system, all stakeholders should be able to comprehend it. It should not be necessary to have a specialized background to understand the architecture.

⁵ Institute of Electrical and Electronics Engineers

⁶ <http://www.iso-architecture.org/ieee-1471/>

3.2.The Architecture Business Cycle

The Architecture Business Cycle is a method for creating an architecture and developing a system. It is divided in several structured steps. Each of these steps will be described below and projected on the security system. The Architecture Business Cycle is used because it is a structural way of getting from an initial list of requirements to an actual architecture by using diagrams and feedback from the stakeholders.

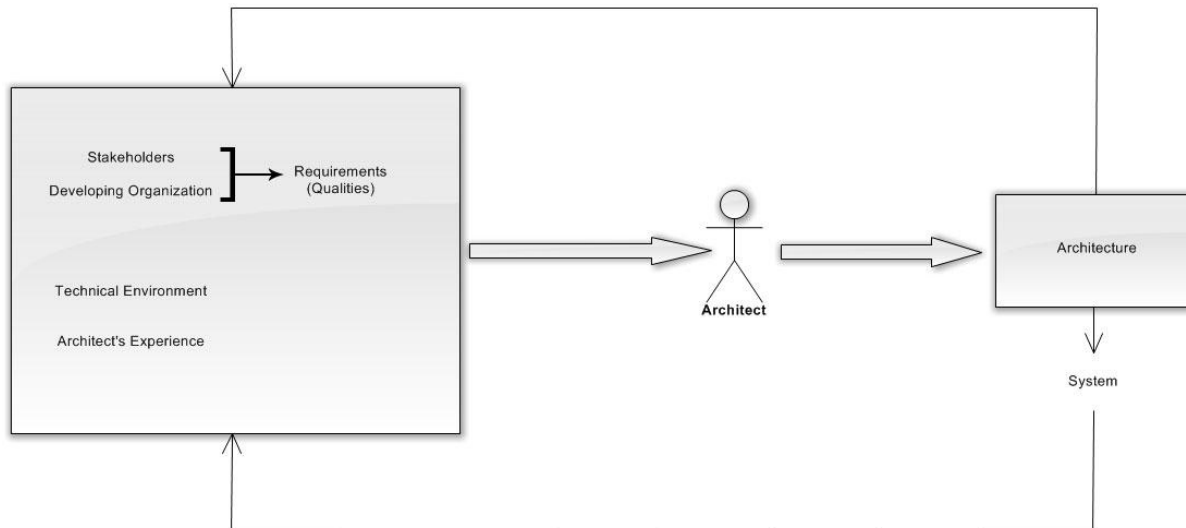


Fig 3.1 the Architecture Business Cycle

Fig 3.1 describes the Architecture Business Cycle. The developing organization and stakeholders (the Project Group) decides on the requirements and gives these requirements to the architect. The architect also looks at the technical environment, in this case what is today's technology like and does it need to be used, and uses his own experience. The architect will then choose an architecture for the system. When this architecture is selected the architect has a meeting with the Project Group again. When everybody is satisfied, the architecture can be made into a system. The system is created and presented to the Project Group again. This procedure follows the same way as developing an architecture. When the system doesn't satisfy everybody, it is possible to re-adjust the architecture and from that new architecture create the system.

In this project the stakeholder and developing organization are the business owner of Verybusy S.R.O. Therefore there only is contact between the architect and the business owner.

3.2.1. Creating the business case for the system

In this step the business goals are defined. The architect, developing organization and shareholders determine the boundaries of the project. How much should the project cost, does it have to interact with other systems in the business and does it need the possibility to expand?

After meetings with the shareholders, in this case the owner of the business, the system has to run on the Open Source Operating System, Ubuntu (Linux) to keep costs low. The prices for the webcams have to be maximum 100 euros. This will keep the price of the system with 3 cameras under 500 euros. The system has to be able to connect to a server system and to an e-mailing system. In the future, the business wants to use this program to be part of a bigger package, so it is necessary that the system is easy modifiable.

3.2.2. Understanding the requirements

The functional and non-functional requirements for the system are defined and made visual by using the "4+1" view.⁷

⁷ <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>

Below are the functional and non-functional requirements for the system. Every requirement has a reference name between brackets. This way they can be tracked in the next part dealing with the system design.

The functional requirements for the system:

[F-REQ 1]: The system has to connect to multiple (Wi-Fi/IP) cameras to ensure that multiple parts of a space can be monitored.

[F-REQ 2]: The system has to be able to detect movement.

[F-REQ 3]: When an intruder is detected video and images have to be recorded.

[F-REQ 4]: The system must be available from the internet through a webpage

[F-REQ 5]: The images have to be sent through mail and SMS.

[F-REQ 6]: Stored video and images have to be available online through a download link.

[F-REQ 7]: A physical alarm has to be started.

[F-REQ 8]: All events must be logged.

[F-REQ 9]: The system needs a timer to delay the start of the physical alarm with 5 minutes to give the administrator the chance to log in and start or stop the alarm.

The non-functional requirements:

[NF-REQ 1]: Because the system has to be low cost, an open source system, like Linux, is the best available option.

[NF-REQ 2]: The server computer and at least one router and one camera need back-up power. In case the power is cut the system would be still functioning.

[NF-REQ 3]: The system needs to be modifiable. In the future the system will be implemented in a bigger system.

3.2.3. The 4+1 View

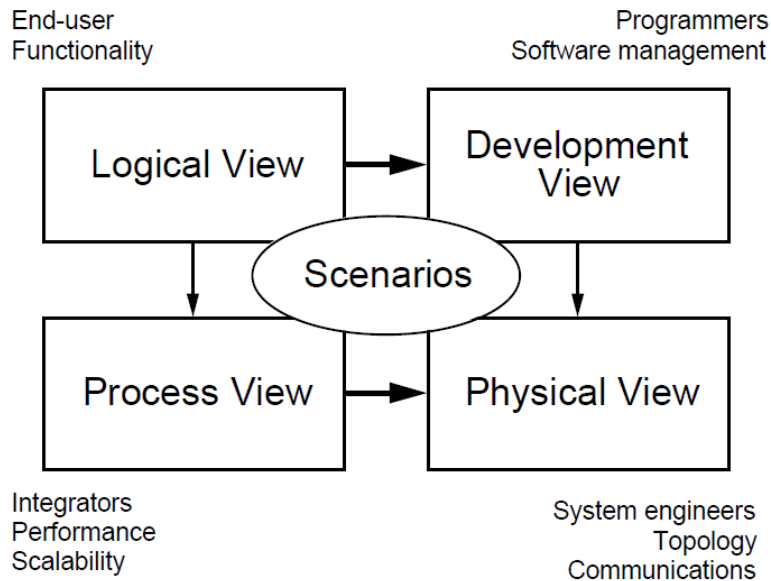


Fig 3.2 the "4+1" View

Figure 3.2 shows the "4+1" View. This view consists of the *Logical* View that is dealing with the functionality that the system provides to users and is visually represented as a Sequence Diagram.

The *Development* View that focuses on the organization of the actual software modules in the software-development environment visually represented by *Component diagrams*.

The *Process* View that focuses on the behavior during runtime visually represented by an *Activity Diagram*.

The *Physical* View that shows the topology of the different components on the physical layer visually represented by *Deployment diagrams*.

Finally the "+1" is the *Scenario* View that shows how the elements of the four views working together seamlessly, visually represented by USE-Cases.

The logical view

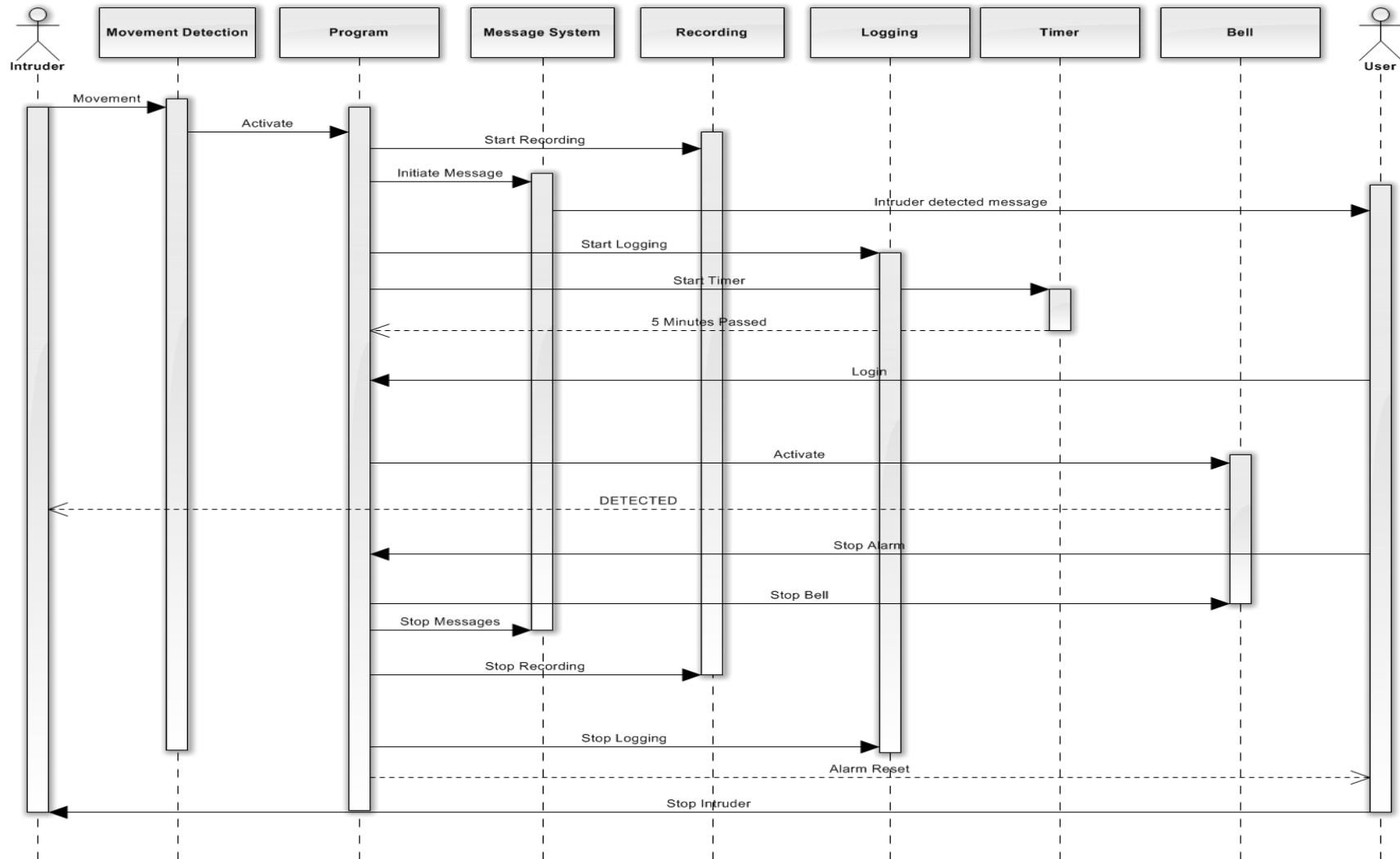


Fig 3.3 Sequence diagram detected intruder

Fig 3.3 shows the logical view through a sequence diagram in case an intruder is detected

When there is movement the Movement Detection module will inform the Program module. The Program module triggers the recording of images, activates the Message System, starts the Logging of the event and then the Timer is started. The message system will send a message every 5 minutes as long as the alarm is active.

The Timer module starts, and after five minutes sends a message to the Program module that five minutes have passed. The program module will then trigger the physical alarm. This delay is part of the system so it is possible for the administrator to do a visual check first. It can happen that an employee forgets to turn the system off and it doesn't need to ring the bell when there is no actual intruder. If this is not the case, the bell will start and the intruder becomes aware that he is detected.

The administrator can login in the program by accessing the program through a web interface. He has the option to view and download the recorded images and to watch the live video streams. The administrator can also start or stop the alarm. When he chooses to end the alert, the program module will stop the physical alarm. Also the recording, the logging and the message system will be stopped. The program will return a message to the user with the information that the alarm has been reset.

Eventually the last step is the administrator stopping the intruder.

More logical views are available in Appendix A describing the sequence for adding and deleting a camera and for starting the system.

The Development View

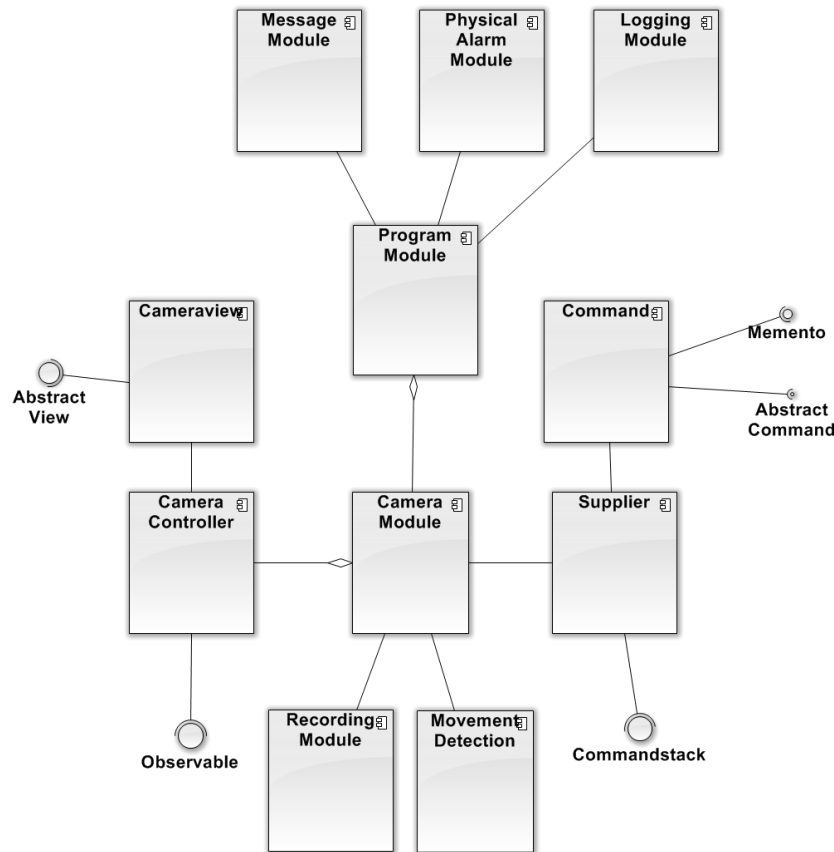


Fig. 3.4 Component Diagram

Fig 3.4 shows the development view through a component diagram.

The Program module is the entry point of the program. The Message module, which handles the sending of messages to the system administrator, is connected to it. The logging module keeps track of all the events and logs them in a file.

The bottom part of this diagram is about how the cameras are handled. The Camera module contains the business logic for a camera. It reads the streams and makes them available for other components. The Movement Detection component receives the stream from the Camera module and sends a checked version back. When anything changes it will outline the change and send the stream back to the Camera module. The Recording module handles the recording and storing of the video and is using data from the Camera module.

The Cameraview implements all the methods from the Abstract View component, which contains basic information that applies to all views. Every view is connected to a Camera Controller component. This Camera Controller gets the data necessary from the Camera Model and sends it to the attached view. When the security system is active it will not get the basic stream from the Camera Model, but it will get the stream checked by the Movement Detection component.

The Command component is an instance of the Abstract Command interface and extends the Memento component. The Memento component is used to store state information. The Command handler implements all the methods from the CommandStack. Commands carried out on a camera will end up on the CommandStack and can be undone. Commands will be divided in different categories. The "no change" commands are commands that don't change the internal data. The "undoable" commands are commands that change data. These commands will be put on the CommandStack and are reversible. The "no reverse" commands are commands that change the internal data completely and therefore are irreversible.

The Process View

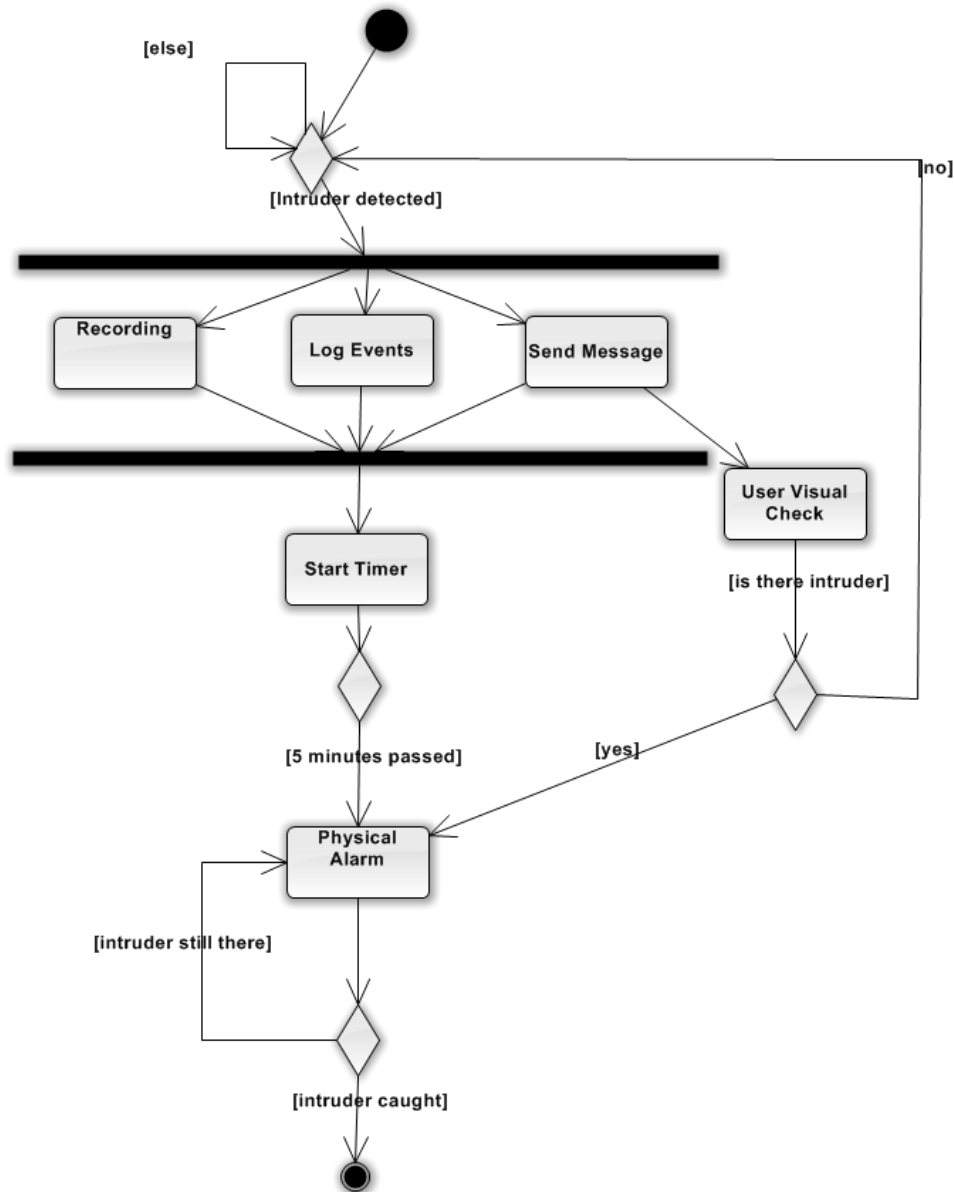


Fig. 3.5 Activity Diagram Intruder

Fig 3.5 shows the process view represented by an activity diagram

Starting at the initial state there is the choice if there is an intruder. In case there is a detected intruder the recording and logging will start and a message will be sent. If there is no detected intruder the next step is checking again if there is an intruder.

Then a timer starts. This timer gives the administrator five minutes to respond to the event. When there is no response within those five minutes the physical alarm will start. If the administrator is in time he can assess if it is necessary to start or stop the alarm. When there is no reason to start the alarm he can reset the system to its original state.

When the intruder is caught, the final state has been reached.

The Physical View

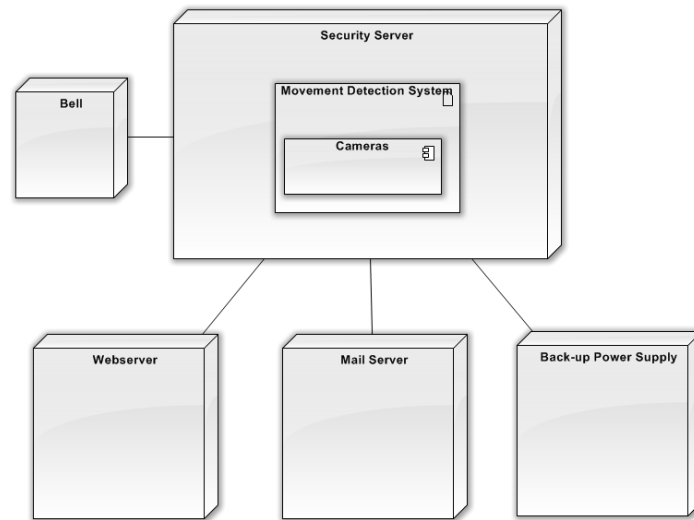


Fig 3.6 Deployment Diagram

Fig 3.6 shows the Physical View visualized by a deployment diagram.

The security server is running the movement detection system. The movement detection system makes use of the cameras. The security server has access to a webserver on the local computer to host the webpage that users can use to manipulate the system from anywhere. The security server also has access to the mail server that it uses to send the messages to the users.

There is also a back-up power supply connected in case there is a power failure.

Overview of the system

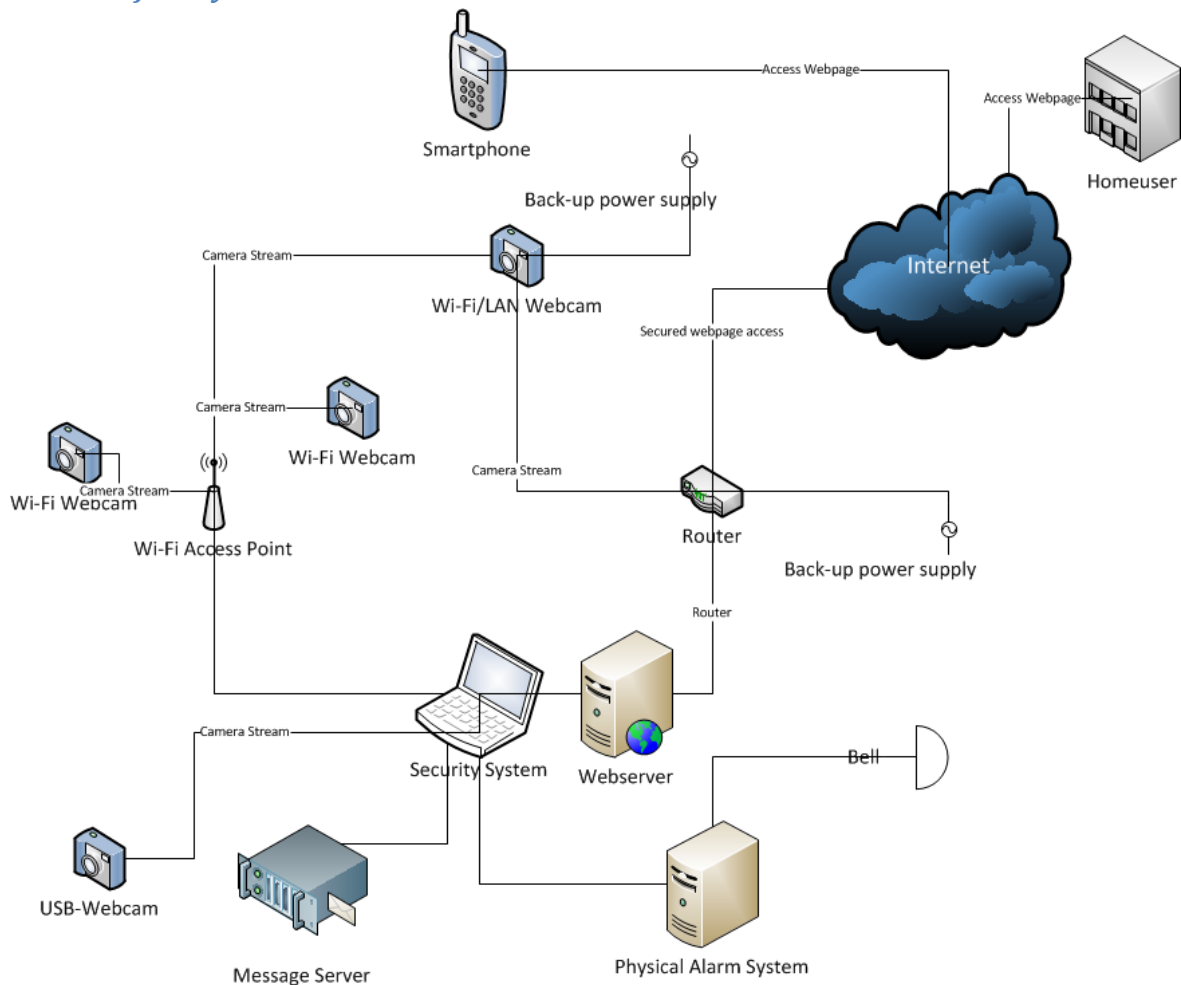


Fig 3.7 System Overview

Fig 3.7 shows an overview of the system.

The different Wi-Fi webcams have a connection with a Wi-Fi access point. The security system also has a connection point so it can receive the camera streams. There is one USB webcam connected to the security system. This webcam will be monitoring the room where the system is located. One of the cameras is connected with a cable to a router. In case of a power failure both of these devices will receive back-up power from a battery. This to make sure the security system can still fulfill its basic tasks. The security system itself will be installed on a laptop. This makes sure the system can handle a power failure for a period of time.

There is a connection between the security system and a mail server to send messages via e-mail and SMS. Also it is connected to the physical alarm that will ring the bell.

The webserver receives incoming connections from the internet. It will provide a user interface and a way to control the security system.

Scenarios

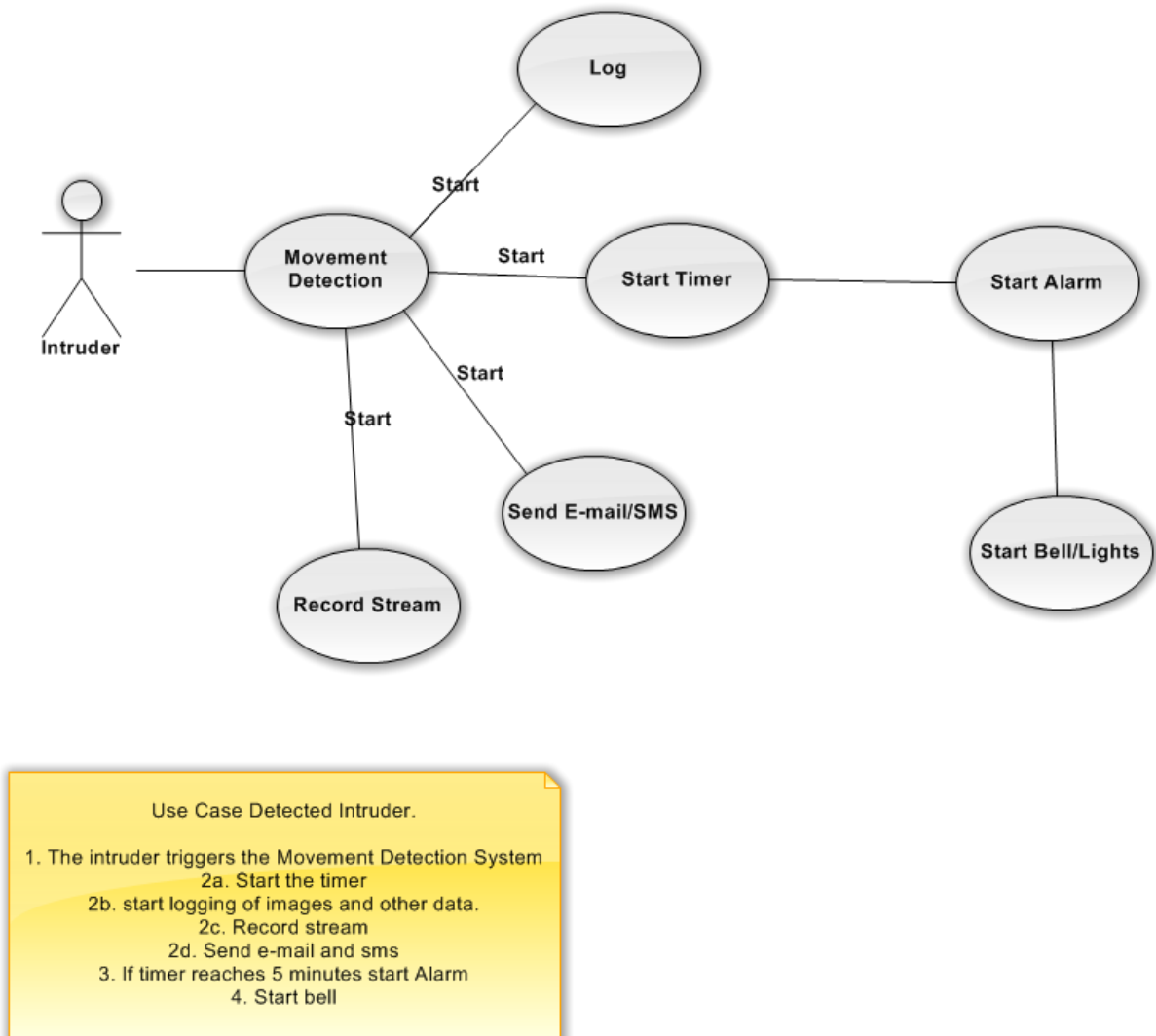


Fig 3.8 Use case of detected intruder

Fig 3.8 shows a use-case with the behavior of the system when there is an intruder detected. The movement detection signals the intruder and starts the logging of events, initiates the sending of e-mails and SMS and starts the timer. When the timer reaches five minutes it will start the alarm and the bell will go on.

More use cases are described in Appendix B

3.2.4. Creating and selecting the architecture

As mentioned before, an architecture is partially chosen by how well it fits the requirements, but also by the technical environment and experience of the architect. In this case the architect has experience with Object Oriented architectures. Each aspect of the system can be divided in modules. For instance, the cameras can be seen as an object and the message system can be seen as an object.

Because the system has to be modifiable the object-oriented architecture is a good way of dealing with these problems. It ensures that modules can be expanded. Taking them out of context and making them more abstract ensures that they can be used in different ways and different contexts. For instance when a business wants to expand the security program and they want to show the video of different cameras on a screen that is visible to everybody it is easy to just expand the program and make use of the camera objects. In this case it is not necessary to re-write the whole program.

In Object Oriented Architectures, there are always problems that are re-appearing. Because of this, a lot of problems have already been solved by other software engineers. Basic resolutions to these problems have been documented. These documented solutions are known as Software Patterns. The system will make use of these patterns. The patterns will be described and applied to the system in the next chapter that is dealing with the design of the system.

4. Designing and implementing of the system

4.1. Software Patterns

Software patterns are documented solutions to specific and recurring software problems. Every pattern consists of three different parts. The relationship between a certain context, the problem it has to tackle and the resolution to this problem. A fourth part is added that explains how the patterns are used in the security system. **[NF-REQ 3]**

The Model View Controller Pattern

Context: Interactive application with a flexible human-interface. In the system there will be modules that can be viewed in different ways.

Problem: The system has the requirement to be modifiable. It might be necessary in the future to expand the system with added functionality or changing the way the data is shown to the user. If all information is in one module changing or adjusting will take a long time and will cost a lot of extra resources.

Solution: The main functionality (the model) consists of the core data and functionality. The view displays the information to the user and gets the data from the model. Each view has an associated controller component that regulates the communication between model and view. This solution allows multiple views to make use of the model. When there is a need to change a view it is not necessary to change the whole system, but only make changes in the view component.

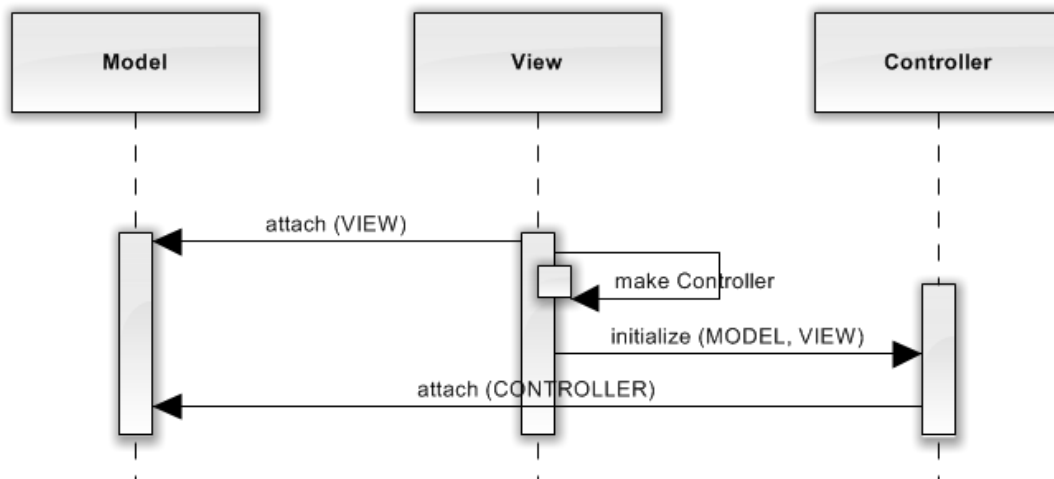


Fig. 4.1 the Model View Controller

How it is used in the security system:

In this case the *model* is CameraModel, the *view* is CameraView and the *controller* is CameraViewController.

The CameraModel

The variables are "private", because the only class that needs to have access is the class itself. The constructor of this class needs to be invoked with a String (the title of the camera). Then a view for the CameraModel is created invoked with the CameraModel (this) and the MainView, which acts as a parent of all views. It is possible to create multiple instances of the CameraModel. This means multiple cameras can be added to the system.

[F-REQ 1]

```
public class CameraModel{
    private String title;
    private CameraView theView;

    public CameraModel(String title) {
        this.title = title;
        theView = new CameraView(this, MainView.getInstance());
    }
}
```

The CameraView

The CameraView implements all the methods from the AbstractView. The variables are "private", because the only class that needs access to these is the class itself.

```
public class CameraView implements AbstractView {
    private CameraViewController controller = null;
    private MainView mainView = null;
    private CameraModel model = null;
    private ViewHandler viewHandler = null;
    ...
}
```

The constructor of this class needs to be invoked with the model it is going to use and the view it has as a parent view. The variables are set.

```
public CameraView(CameraModel model, MainView mainView){
    this.model = (CameraModel) model;
    this.mainView = mainView;
}
```

The CameraView has a method to make a controller that will be attached to this view. First it will create the CameraViewController and then initialize it with the model parameter and the view (this) parameter.

```
public void makeController() {
    controller = new CameraViewController();
    controller.initialize(model, this);
}
```

The CameraViewController

The CameraViewController class implements all the methods from the AbstractController, which will be explained in the Command Processor pattern. Also here the variables are set "private" because the only class that needs access to them is the class itself.

```
public class CameraViewController implements AbstractController {
    private CameraModel cameraModel=null;
    private CameraView view = null;
    ...
}
```

The initialize method invoked by the CameraView in the previous section attaches the model and view to the variables in this class and it registers itself with the CameraModel

```
public void initialize(Observable model, AbstractView v){
    cameraModel = (CameraModel) model;
    cameraModel.registerObserver(this);
    view = (CameraView) v;
}
```

The Viewhandler Pattern

Context: The system provides multiple views that will be used at the same time.

Problem: When multiple views are used (one camera view, multi-camera view, show options) it is difficult to keep track of the different views. Data can be represented in different ways. The different views should not be depending on each other and it should be possible to add more views on demand.

Solution: The Viewhandler component manages all views. Every view will be added to a list and the Viewhandler takes over responsibility for them.

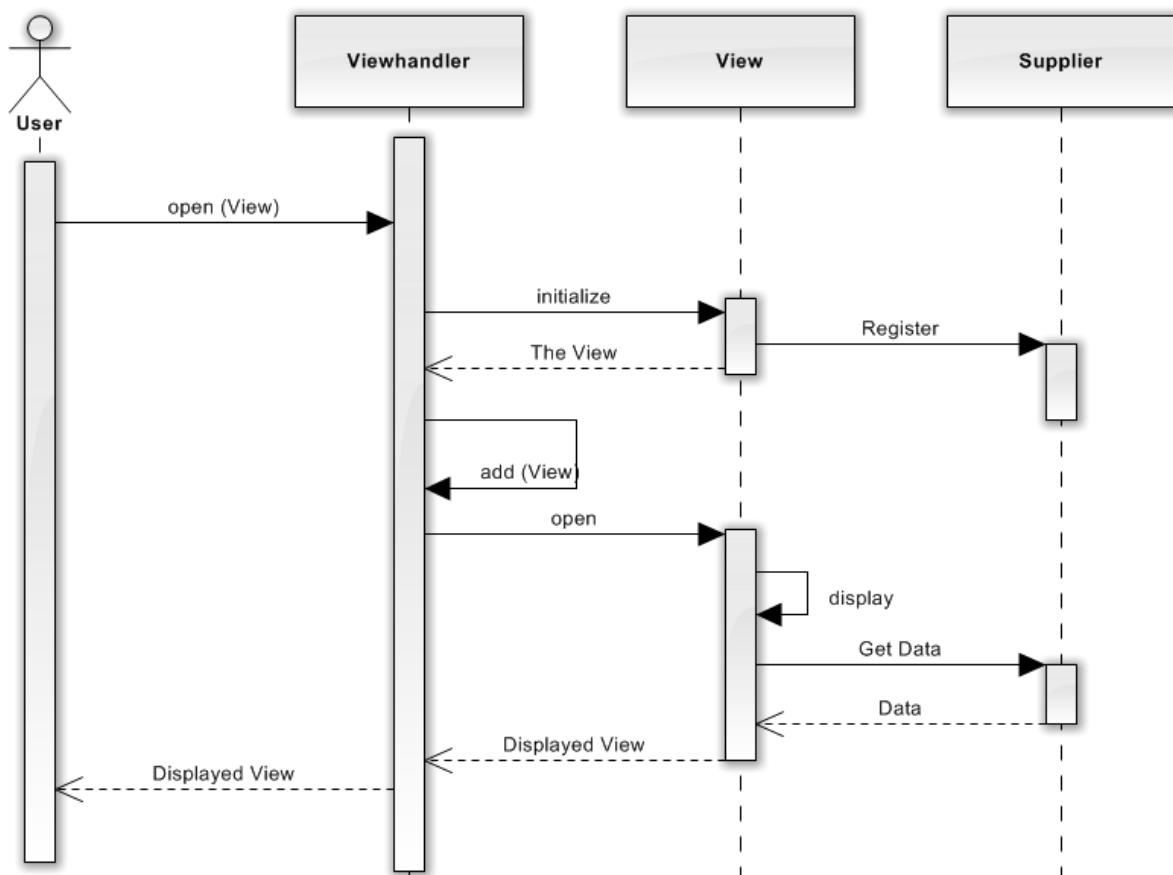


Fig 4.2 the Viewhandler

How it is used in the system:

In this case the Viewhandler is Viewhandler, the View is CameraView and the Supplier is the CameraModel.

The variables again are private. The vhInstance and isInstanced are static, because they are part of this class and shouldn't be part of instances of this class. The constructor of this class creates an ArrayList that only accepts AbstractViews.

```
public class ViewHandler {
    private static ViewHandler vhInstance;
    private static boolean isInstanced = false;
    private ArrayList<AbstractView> viewList = null;

    private ViewHandler() {
        viewList = new ArrayList<AbstractView>();
    }
}
```

The Viewhandler has a method **openView()**, described later, that handles every added view. In the constructor of the CameraModel two lines are added to make use of the ViewHandler. The instance of the Viewhandler is loaded in a class variable and then the method **openView()** is called with theView as a parameter.

```
public CameraModel(String title) {
    this.title = title;
    viewHandler = ViewHandler.getInstance();
    theView = new CameraView(this, MainView.getInstance());
    viewHandler.openView(theView);
}
```

The openView method of the ViewHandler calls different methods of the view it received as input and adds the view to the ArrayList of views.

```
public void openView(AbstractView view) {
    view.initialize();
    view.makeController();
    view.doLayout();
    view.open();
    viewList.add(view);
}
```

The Publisher-Subscriber Pattern (Observer)

Context: Multiple components are dependent on the same data. The components need to be notified when changes occur.

Problem: When data changes, usually multiple components are affected. In such a case all dependent modules should be notified.

Solution: One dedicated component takes the role of the publisher. All modules that are dependent on changes of the publisher are its Observers. Any time the state of the Publisher changes it notifies all its Observers.

How it is used in the system:

There is an interface Observer that only has a method **update()**. This method ensures that all connected Observable objects are notified.

The Observable class is an abstract class that is extendible. The variables are protected so they can only be accessed from the same package and by a subclass of this class in another package. The constructor method creates a ArrayList of Observers.

```
public abstract class Observable {

    protected ArrayList<Observer> observerList = null;
    protected boolean stateChanged = false;

    public Observable() {
        observerList = new ArrayList<Observer>();
    }
}
```

Every observer created will be stored in this list with help of the **registerObserver()** method.

```
public void registerObserver(Observer theObserver) {
    observerList.add(theObserver);
}
```

The Command Processor Pattern

Context: There is a need for more functionality that expands the basic functionality and possibilities to undo changes.

Problem: Cameras need to be added, images may need to be enhanced like adjusting the brightness or colors, and there should be specific commands to deal with this. These commands should not change the existing code. When changes are made, it should also be possible to reverse some of these commands.

Solution: Abstract Command objects will be made wherein the functionality of the command is described and how to “do” and “undo” the command. These commands will apply their functionality to the data they get from the supplier. The Command Processor will put these commands in a stack so it can keep track of the commands and undo them if necessary.

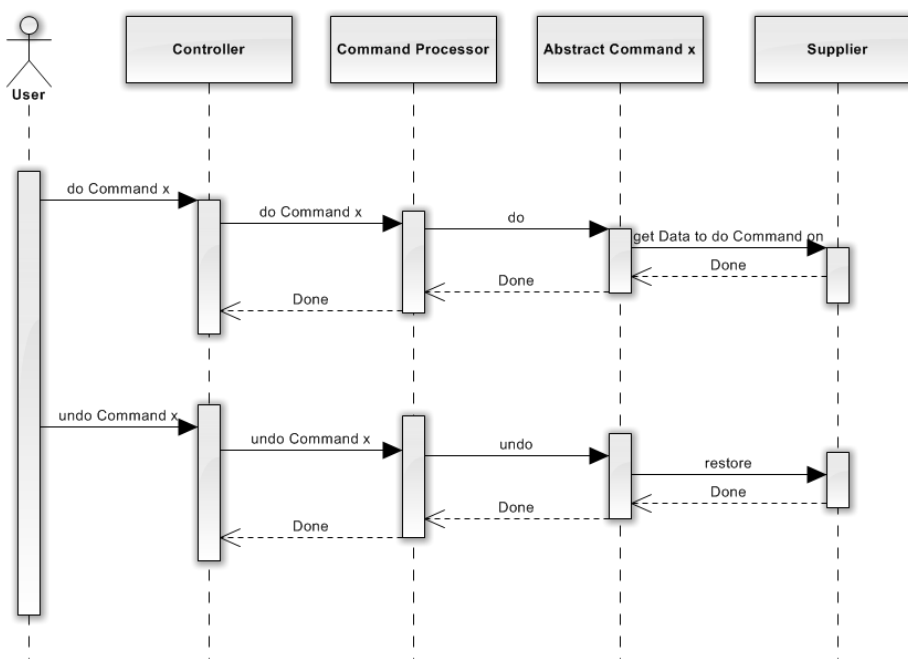


Fig. 4.3 the Command Processor

How it is used in the system:

In this case the Controller is the MainViewController, the Command Processor is an implementation of the CommandProcessor class in the ProgramModel. The Supplier is an implementation of the Supplier class. The

Supplier fetches the data that needs to be adjusted. In this case, the data of the ProgramModel will be adjusted. The adding of a new camera to the main program will be explained.

The class NewCameraCommand extends the AbstractCommand class. The supplier variable is the part that the command has effect on, the ProgramModel. The model variable will contain the camera to be added to the supplier. The *instances* variable counts how many new cameras have been added.

```
public class NewCameraCommand extends AbstractCommand {
    private ProgramModel supplier = null;
    private CameraModel model = null;
    private static int instances = 0;
```

The AbstractCommand class defines the sort of command. If it will have no effect it will be a *NOCHANGE* command. When the changes can be undone it will be a *UNDOABLE* command and when it is a command that alters the data in a way that undoing is not possible it will be a *NOREVERSE* command. The NewCameraCommand is a *NOCHANGE* command.

```
public abstract class AbstractCommand {
    public static int NOCHANGE = 0;
    public static int UNDOABLE = 1;
    public static int NOREVERSE = 2;
    ...
```

The ProgramModel implements the CommandProcessor and creates a Stack list of AbstractCommand

```
public class ProgramModel implements CommandProcessor{
    private Stack<AbstractCommand> activeCommandStack = null;
    ...
```

The ProgramModel implemented the methods *executeCommand()* and *undoCommand()* from the CommandProcessor. The method is invoked with the command (in this case the NewCameraCommand). It checks the type of the command, a *NOCHANGE* command. If it was an *UNDOABLE* command it would be pushed on the command stack and all observers would be notified of the changes. The *undoCommand()* checks if the commandstack is not empty and if so it will undo the command and pop (remove) it from the stack and notify all observers.

```
public void executeCommand(AbstractCommand theCommand) {
    theCommand.doCommand();
    if (theCommand.getType() == AbstractCommand.UNDOABLE) {
        activeCommandStack.push(theCommand);
        notifyObservers();
    }
    if (theCommand.getType() == AbstractCommand.NOREVERSE) {
        activeCommandStack.clear();
        notifyObservers();
    }
}

public void undoCommand() {
    if(!activeCommandStack.isEmpty()){
        AbstractCommand theCommand = activeCommandStack.pop();
        theCommand.undoCommand();
        notifyObservers();
    }
}
```

The Memento Pattern

Context: Commands are carried out and there needs to be a possibility to restore the previous state.

Problem: Every time a command to change a part of the system is carried out, there needs to be a way to store the state to return to this state whenever an undo command is carried out.

Solution: The memento will store the state of the object and makes it available in case the changed object has to be restored in this state.

How it is used in the system:

The CameraModel has a named inner class that is called CameraModelMemento that extends the AbstractMemento. It is an inner class because it has a composite relationship with the CameraModel. The class consists of the variables that will be set and the implementation of the methods in the AbstractMemento to **setState()** and to **getState()**.

```
private class CameraModelMemento extends AbstractMemento {  
    // Variables (state)  
    public void setState() {  
    }  
    public void getState() {  
    }  
}
```

When a command is carried out the **createMemento()** method is called. It creates the new memento and sets the necessary state.

```
public AbstractMemento createMemento() {  
    CameraModelMemento cameraModelMemento = new CameraModelMemento();  
    cameraModelMemento.setState();  
    return cameraModelMemento;  
}
```


4.2. Reading the video streams

The system is going to make use of the Java Media Framework. This framework has methods to handle video. It will be used to read the video stream input. This part will explain how the reading of the streams works. The method `loadCamera(String camera)` in the `CameraModel` class will create the stream for a camera.

First a Processor needs to be created. A Processor needs to have complete control over a video stream.

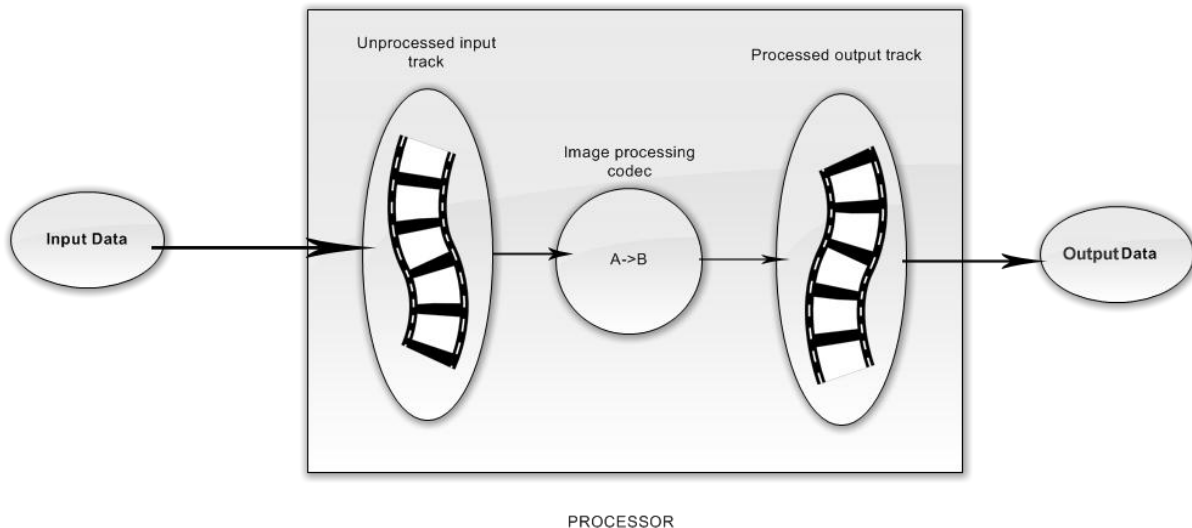


Fig 4.4 Video Processor

Fig 4.4 shows how an unprocessed video track is transformed to a processed output track that can be used by the system.

The `CameraModel` will create a `DataSource` for a camera. This can be seen as the medium containing the stream data. A variable, named `device`, will be used to store the info from the capture device. The `MediaLocator` variable `ml` stores the `Locator` of the device. When a valid `MediaLocator` is available, the `DataSource` will be created.

```
CaptureDeviceInfo device =
CaptureDeviceManager.getDevice(cameraDevice);

MediaLocator ml = device.getLocator();
DataSource tds = null;
try{
    tds = Manager.createDataSource(ml);
} catch (Exception e) {
    System.err.println("Failed to create a datasource");
    System.exit(1);
}
```

When a DataSource is created, the processor for the media can be created. This will happen in the **openStream()** method of the CameraModel. The method needs to be called with the DataSource as input. The variable *p* used in this method is a static Processor variable in the CameraModel class. The variable *tds* is the DataSource. After creating the processor, a ControllerListener is added. This listener is used for starting, pausing, and stopping the streaming media.

```
try {
    p = Manager.createProcessor(tds);
} catch (Exception e) {
    System.err.println("Failed to create a processor from the given
        datasource: " + e);
}
p.addControllerListener(this);
```

A processor goes through a series of states before it can use the input media. The initial state is an unrealized state. It needs to be configured to get into a realized state. In the configuring state the processor connects to the DataSource. This is done by calling the configure function from the Manager class.

```
p.configure();
if (!waitForState(p.Configured)) {
    System.err.println("Failed to configure the processor.");
}
```

The processor is now fully configured and ready to be used.

DataSources present media in tracks. Every track consists of a type of media. For example an MPEG movie can contain an audio and a video track. To handle these tracks a TrackControl is necessary. In this part a TrackControl array is made and the video track controls are loaded in a variable *videoTrack*.

```
TrackControl tc[] = p.getTrackControls();
if (tc == null) {
    System.err.println("Failed to obtain track controls from the
        processor.");
}
TrackControl videoTrack = null;
for (int i = 0; i < tc.length; i++) {
    if (tc[i].getFormat() instanceof VideoFormat) {
        videoTrack = tc[i];
        break;
    }
}
```

Now there is a processor created that handles the input data. Next a codec is needed to process the data. A codec array is created in case multiple effects need to be used. These effects will be added to the videoTrack. In this case the motionDetectionEffect is added. Then the codec array is set to the videoTrack track control. The motionDetectionEffect will be explained in the next part, the motion detection of the system.

```
try {
    Codec codec[] = { motionDetectionEffect };
    videoTrack.setCodecChain(codec);
} catch (UnsupportedPluginException e) {
    System.err.println("The processor does not support effects.");
}
```

4.3. Motion detection

[F-REQ 2]

To detect motion the program will first take a screenshot of the first frame it receives. This screenshot will be stored and the following frame will be compared to this screenshot. Whenever a pixel has changed considerably, determined by a threshold variable, a internal black-white-red image is marked at the location where this happened. Then the checked region will be checked for bigger clusters of changed pixels. If the count of these clusters is bigger then a certain threshold it will be considered as detected motion. The first reference frame will be deleted and replaced by the second frame. Then the third frame will be compared with the second frame and so on.

In the previous part the motionDetectionEffect was added to the videoTrack track control. This motionDetectionEffect is an instance of the MotionDetectionEffect class. The MotionDetectionEffect class extends the Codec class and therefore inherits the methods of this JMF class. The **process()** method, inherited from the Codec class is called when the Processor, described in reading the video streams, is started with the following command `p.start()`

The different variables to handle the images are declared in the **process()** method. Behind the declarations are comments explaining for what they are used.

```
int outputDataLength =
    ((VideoFormat)outputFormat).getMaxDataLength(); //Length of output
int width = sizeIn.width; //get the Frame width
int height = sizeIn.height; //get the Frame height
int r,g,b; // values to store the red, green and blue values
int ip, op; // values to store intensity
byte result; //store the intensity normalization
int avg = 0; //average intensity
int refDataInt = 0; //reference data
int inDataInt = 0; //input data
int correction; //difference between the image and reference image
```

If there is no reference image, it will be created. The data in this variable will be to compare all following images to. The refData is declared as a byte array and the `System.arraycopy()` copies the incoming data in the `refData`. The `bwData` is used later during the movement detection, it will contain a black and white image of the frame.

```
if (refData == null) {
    refData = new byte[outputDataLength];
    bwData = new byte[outputDataLength];
    System.arraycopy (inData, 0, refData, 0, inData.length);
    ...
}
```

The average intensity of the image is calculated. Every pixel's intensity is added to the avg variable. The `inData` is a byte array that is being converted to an int array. When some of the bytes are 128 or above they will be interpreted as being negative. When they are cast to integers they will still be negative. To make sure that the byte value is between 0 and 255 the `& 0xFF` is added.

```
for (ip = 0; ip < outputDataLength; ip++) {
    avg += (int) (inData[ip] & 0xFF);
}
avg_img_intensity = avg / outputDataLength;
...
```

Now the reference frame is compared with the new frame. A new variable *pixStrideIn* is introduced. When a video image is stored in memory, the memory buffer contains extra padding bytes after each row of pixels. The stride is the image width plus the extra padding.

The for loop starts with 0 and will run until the output data length divided by the *pixStrideIn* variable. Every pixel from the *inData* variable that is delivering the new frame will be compared with the reference data and stored in the variables *r*, *g* and *b*.

```
for (int ii=0; ii< outputDataLength/pixStrideIn; ii++) {
    refDataInt = (int) refData[ip] & 0xFF;
    inDataInt = (int) inData[ip++] & 0xFF;
    r = (refDataInt > inDataInt) ? refDataInt - inDataInt :
inDataInt - refDataInt;

    refDataInt = (int) refData[ip] & 0xFF;
    inDataInt = (int) inData[ip++] & 0xFF;
    g = (refDataInt > inDataInt) ? refDataInt - inDataInt :
inDataInt - refDataInt;

    refDataInt = (int) refData[ip] & 0xFF;
    inDataInt = (int) inData[ip++] & 0xFF;
    b = (refDataInt > inDataInt) ? refDataInt - inDataInt :
inDataInt - refDataInt;
```

The *result* variable will be filled with a result of every pixel calculated. This *result* is calculated by a standard formula⁸ for "lightness" or "brightness".

```
result = (byte) (java.lang.Math.sqrt((double) ( (r*r) + (g*g) + (b*b)
) / 3.0));
```

When the *result* is calculated it will be compared with the threshold variable. Because small changes will always occur, the threshold variable is used to determine how big the change is. If the result exceeds the threshold, motion is detected and the change will get the white color.

```
if (result > (byte)threshold) {
    bwData[op++] = (byte)255;
    bwData[op++] = (byte)255;
    bwData[op++] = (byte)255;
} else {
    bwData[op++] = (byte)result;
    bwData[op++] = (byte)result;
    bwData[op++] = (byte)result;
}
```

⁸ http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html#RTFTtoC36

4.4. Recording images and video

To record images, the MotionDetectionEffect class has inherited the method **process()** and the method **SaveFrame()** from the JMF Codec class. The **process()** method is called when the Processor described in reading the video streams is started (**p.start()**) [F-REQ 3]

The process method calls the **SaveFrame()** method with the Buffer in variable.

```
public int process(Buffer in, Buffer out) {
    SaveFrame (in)
    ...
}
```

The **SaveFrame()** method first converts the Buffer to a frame.

```
void SaveFrame(Buffer frame){
    BufferToImage buff2Image = new BufferToImage
    ((VideoFormat) frame.getFormat());
    ...
}
```

The method then creates a Java Image.

```
Image currFrame = buff2Image.createImage(frame);
```

The variable *f* creates a new file with the *now* variable that contains the date and time at that moment and adds the jpg extension. A new file is created and a BufferedImage is created. In this BufferedImage the *currFrame* is added. Then a new OutputStream *os* is created that is filled with the file. The OutputStream is then added to a JPEG encoder. Finally the *outImage* is encoded.

```
File f = new File (now + ".jpg");
f.createNewFile();
BufferedImage outImage(320,240, BufferedImage.TYPE_3BYTE_BGR);
outImage.createGraphics().drawImage(currFrame,0,0,null);

OutputStream os = new FileOutputStream(f);

com.sun.image.codec.jpeg.JPEGImageEncoder encoder =
com.sun.image.codec.jpeg.JPEGCodec.createJPEGEncoder(os);
encoder.encode(outImage);
```

To record video, a Record class is being used.[F-REQ 3] This class will handle the recording of all streams. The recording is called from the CameraModel class.

The DataSource of the camera is fetched, and a copy of the DataSource is created.

```
DataSource ds = ((SourceCloneable) getDataSource()).createClone();
```

Then a formats variable is created in which the video formats are stored, followed by creating an outputType, that is set to what kind of output the file should be. In this example the Quicktime format is used.

```
Format formats[] = new Format[1];
formats[0] = new VideoFormat(VideoFormat.CINEPAK);
FileTypeDescriptor outputType =
    new FileTypeDescriptor(FileTypeDescriptor.QUICKTIME);
```

A Processor needs to be created to handle the DataSource. The creating of the processor is similar to the creating of a processor in the part dealing with reading the video streams. Then a controller listener is added to the Processor, used for starting and stopping the stream.

```
p = Manager.createRealizedProcessor(new ProcessorModel(ds, formats,
outputType));
p.addControllerListener(this);
```

A new DataSource is created. The output data from the Processor will be added to this variable.

```
DataSource source = p.getDataOutput();
```

The data is ready to be written to a file. A MediaLocator is created with the filename as an argument. The *now* variable represents the current time and date. A *filewriter* is created and opened so it is possible to write to it.

```
MediaLocator dest = new MediaLocator("file://" + now + ".mov");
DataSink filewriter = null;
try {
    filewriter = Manager.createDataSink(source, dest);
    filewriter.open();
} catch (Exception e) {
    System.out.println("Failed to create file writer");
    System.exit(-1);
}
```

Then the *filewriter* is started.

```
try {
    filewriter.start();
} catch (IOException e) {
    System.exit(-1);
}
```

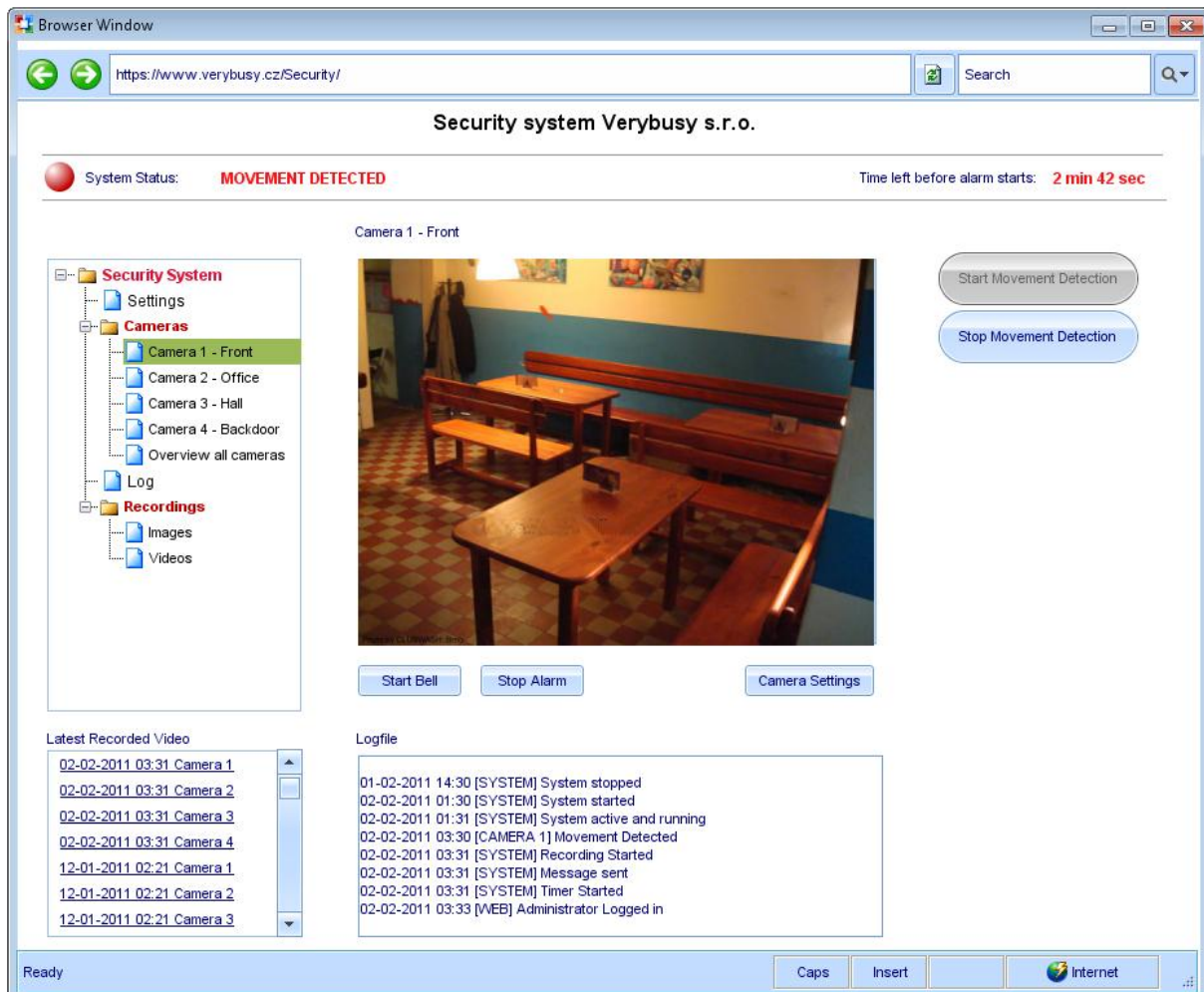
The final step is starting the Processor. When the Processor is started, the video will be recorded and stored.

```
p.start();
```

4.5. The webpage

The webpage will be available on the internet. **[F-REQ 4]**

The system administrator can log in and open the webpage. If movement has been detected the system will show this in the top part. In the right top part there will be indicated how much time is left before the bell will be turned on.



The left part has the menu structure and the options to access the settings of the system. Also it is possible to view every camera, or to select an overview of all the cameras. The log is available and all the recorded streams and images are available.

In the bottom left part there are direct links to the recorded video. **[F-REQ 6]** In the bottom middle part the log file is shown.

When there is a camera selected (in this case camera 1), the picture is shown and under the picture there will be the option to start the bell if there is a need for it, and there is an option to stop the alarm. The final option is to open the camera settings.

On the right side there will be the options to start the movement detection, in this case it is already running, and because of that the button is not accessible. The other option is to stop the movement detection.

4.6. Sending of mail

The sending of mail can be done by using the JavaMail API **[F-REQ 5]**

The method ***fetchConfig()*** takes care of loading values, like the mail server, from a document. The variable ***fMailServerConfig*** then loads this input. This variable is used for sending mail. A document is used because it is easier to adjust the properties when necessary.

```
private static void fetchConfig() {
    InputStream input = null;
    input = new FileInputStream( "./config/mailserver.txt" );
    fMailServerConfig.load( input );
}
```

The ProgramModel class has the method ***sendEmail()*** that will be used to send a message to the administrator when an intruder is detected. ***theMessage*** will contain the actual message in the body of the e-mail.

```
sendEmail("security@verybusy.cz", "administrator@verybusy.cz",
    "Intruder Detected", theMessage);
```

The method ***sendEmail()*** is called with multiple variables. It is called with the e-mail address of the sender, the e-mail address of the receiver, the subject of the mail and the message text. First a *session* will be created. The default instance of the Session is loaded with the properties, like the SMTP server.

Then a *message* is created and loaded with the *session* variable. The message variables, like recipient, subject and body text are set and finally the message is sent with a method of the Transport class, a standard JavaMail API. The ***send()*** method of this class will be used.

```
public void sendEmail(
    String aFromEmailAddr, String aToEmailAddr,
    String aSubject, String aBody){
    Session session = Session.getDefaultInstance(
fMailServerConfig);
    MimeMessage message = new MimeMessage( session );
    try {
        message.addRecipient(
            Message.RecipientType.TO, new InternetAddress(aToEmailAddr)
        );
        message.setSubject( aSubject );
        message.setText( aBody );
        Transport.send( message );
    }
    catch (MessagingException ex){
        System.err.println("Cannot send email. " + ex);
    }
}
```


4.7. The log file

[F-REQ 8]

The log file will keep track of all events happening. The ProgramModel will handle all the messages and append them to the log file by calling the **writeToLog**(String *message*) method. The method creates a Date instance called *now* and a *currentTime* variable that will contain current date and time.

Then a FileWriter, *aWriter*, is created. This FileWriter is created with the location of the log file (*logFile*). Then the *currentTime* is added to a line followed by the *message* and finally a line separator is added to make sure the next message will start at a new line. Then *aWriter* is flushed to make sure everything written is committed to the hard drive and finally the FileWriter is closed.

```
try {
    Date now = new Date();
    String currentTime = now.toString();
    FileWriter aWriter = new FileWriter(logFile, true);
    aWriter.write(currentTime + " " + message
        + System.getProperty("line.separator"));
    aWriter.flush();
    aWriter.close();
}
```

4.8. The timer

[F-REQ 9]

The timer is a method that gets called when an event is detected. The **initTimer**() sets the timer and makes use of the *timer* variable declared in the ProgramModel. If the timer reaches the five minutes it will start the physical alarm. The **startTimer**() method restarts the timer. The **restart**() method makes sure the timer always counts the whole five minutes. The *timerRunning* boolean variable is set to true. If the ProgramModel wants to start the timer and it is already running it doesn't need to restart it. The **stopTimer**() stops the timer from counting and sets the *timerRunning* variable to false.

```
public void initTimer() {
    timer = new Timer(300000, new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            startAlarm();
        }
    })
}

public void startTimer() {
    timer.restart();
    timerRunning = true;
}

public void stopTimer() {
    timer.stop();
    timerRunning = false;
}
```

4.9. The user interface

The user interface is the visual part of the program. In this part there will be a brief description about how the interface will be build up.

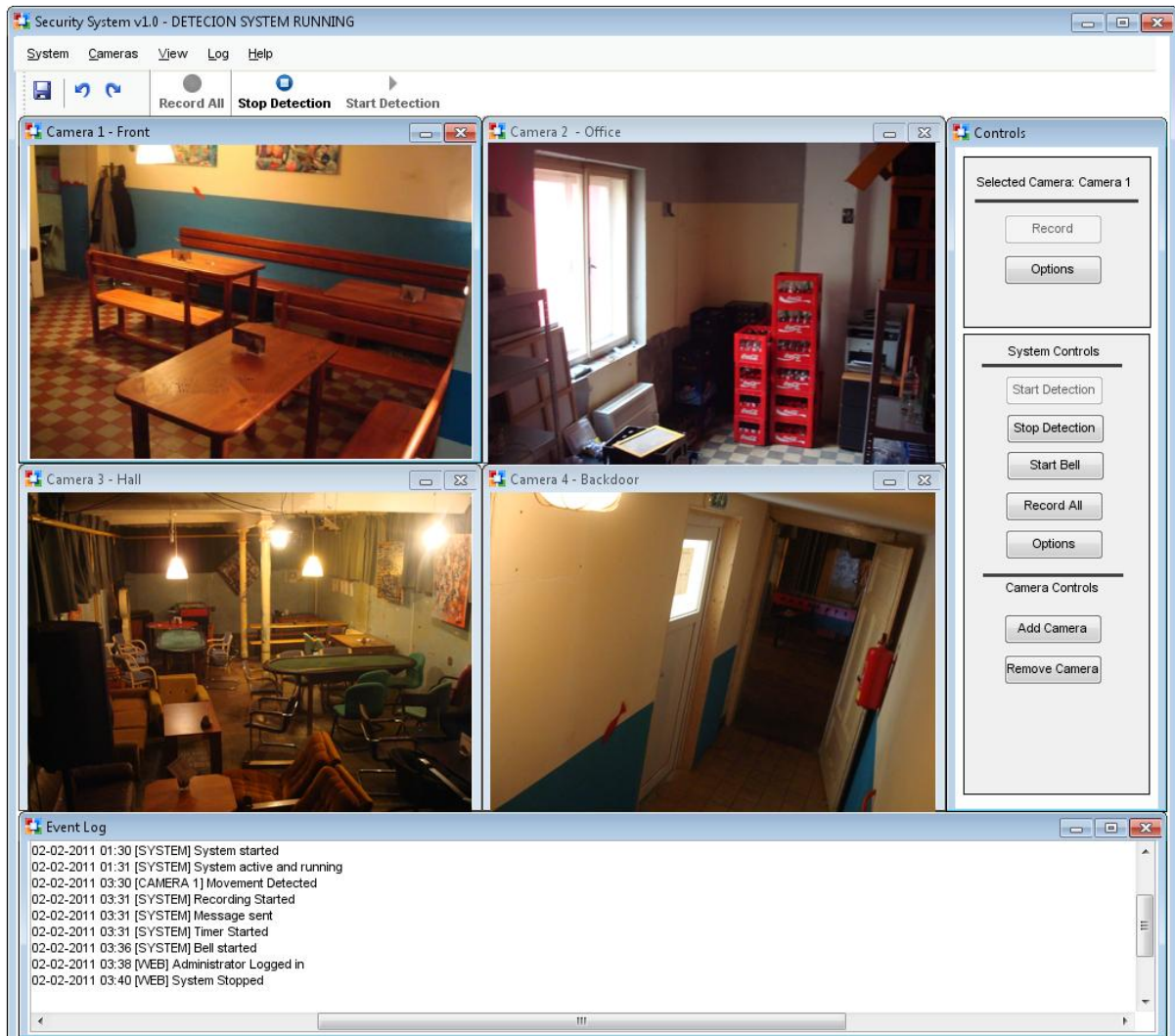


Fig 4.6 the main user interface

Fig 4.6 shows what the main user interface will look like. There is a menu-bar with menus for the system, the cameras, the view, the log and help. Under that menu-bar is a quick bar where there are several commands that will be used more frequently.

There will be four frames that will show the camera feeds. The title consists of a camera name and the place where the camera is located.

On the right are the main controls. When the user selects a camera frame the top part will consist of the options possible for that camera. The bottom part of controls consists of actions that can be performed concerning the system. In this part it is possible to start and stop detection, add or remove cameras or go to the options, to set-up the system.

The bottom part will consist of the event log.

More user interfaces, like options for the system or options for a camera are described in Appendix C.

5. Conclusion, recommendations and evaluation

5.1.Conclusion

The Architecture Business Cycle is a good guide for creating an architecture. By communicating with the stakeholders and writing and re-writing, the system already becomes visible and instead of talking about an abstract system the first design choices are already made. Making choices at an early stage makes the creating of the system easier and faster. At a later stage it is not necessary to rewrite parts because it is already clear what parts will be used and how they interact with each other.

In the future this system will be used as part of a bigger package. Because a choice has been made to use Object Oriented architecture with the use of software patterns embedding this system in a bigger package is made much easier. Also the system can be adjusted much easier and expanded with functionality without completely re-writing the code, but simply by letting the new functions use the already designed code.

The reason for not using an already available and complete software package to handle the motion detection is because this system will be part of a bigger software system. The business doesn't want to be dependent of other available software solutions. If another system would be used it would be much more work to fit it in the package.

5.2.Recommendations

The motion detection is not yet optimized. One of the things to do that will improve the system is to create a faster movement detection algorithm. At this time the streams are in a byte variable. This takes the program longer to calculate. One of the ways to improve that is using an integer variable. The system is able to calculate much faster what will make the overall system faster.

When there is a shadow cast or a car driving by that produces a lot of light the system will respond as movement being detected. The solution at this time is to make sure there is no possibility for this to happen by blinding the windows. An improvement would be when the system is able to determine the difference between light changes and actual movement.

The systems response to an intruder is to send a message to the administrator and to ring a physical alarm. To heighten the security it would be a good thing to have a connection with a security company that has access to the system and can respond to the event.

5.3.Evaluation

The project was very clear and boundaries were defined at an early stage. Because of this and by using a structural approach the entire project was very well manageable. One of the difficulties was finding a suitable API to work with the video streams. The Java Media Framework was a decent solution, but the fact that it was last updated in 2002 makes clear that it is not an up-to-date package. I found it exciting to create and select the architecture, because at that stage the system started to become visible for the first time. The creating of the code is not my strongest side and this part was the most difficult part from my point of view. In general I think the system can be a great asset to the business and can be used in the bigger software package.

Bibliography

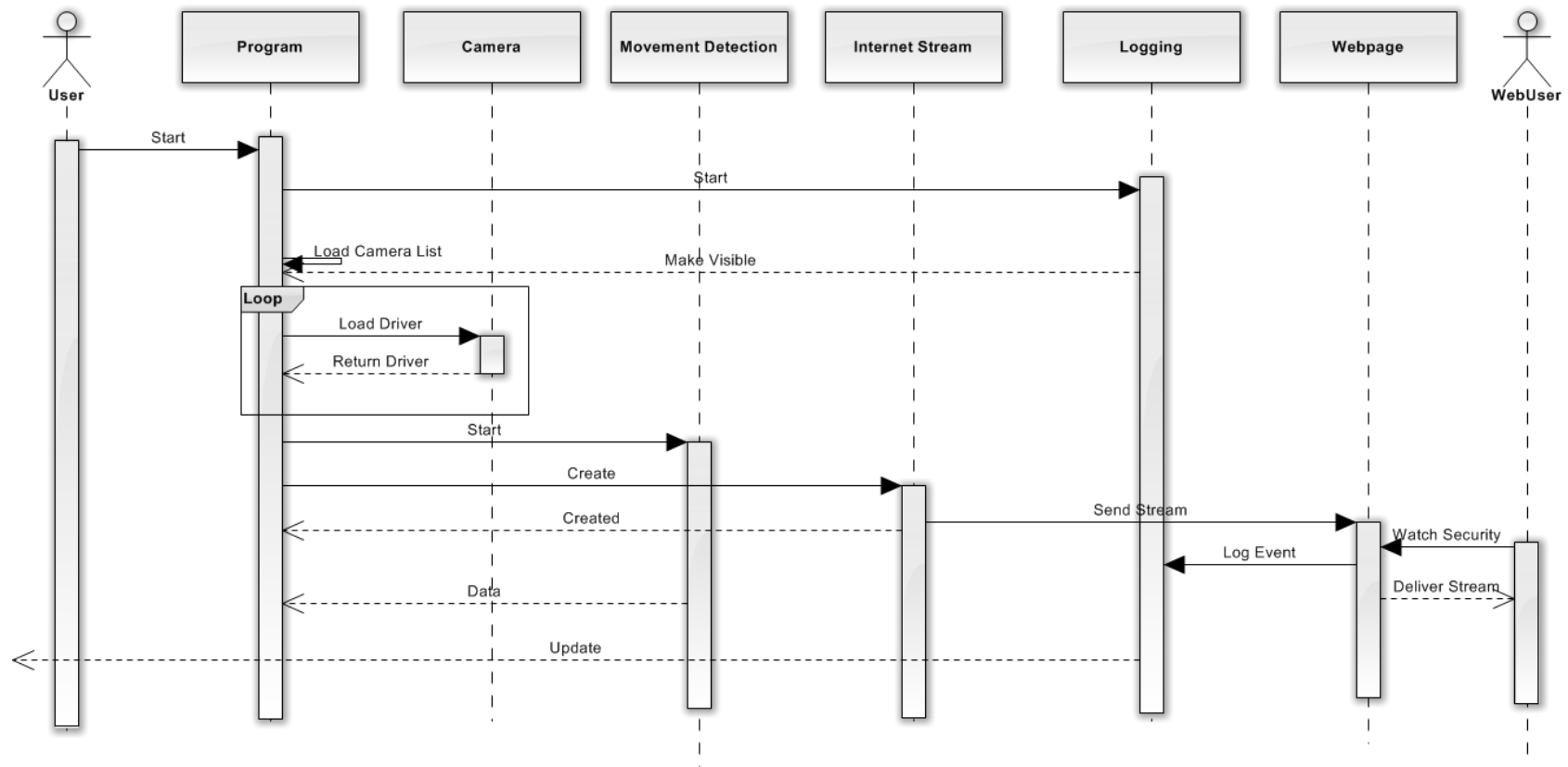
- ANSI/IEEE Standard 1471 :: ISO/IEC 42010. (2001, August 7). Retrieved December 15, 2010, from ISO-Architecture: <http://www.iso-architecture.org/ieee-1471/>
- Adamson, C. (2005, December 12). *Oreillynet.com*. Retrieved October 20, 2010, from MacDevCenter: http://www.oreillynet.com/mac/blog/2005/12/jmf_a_mistake_asking_to_be_rem.html
- Bass, L., Clements, P., & Kazman, R. (2003). *Software Architecture in Practice*. Addison Wesley.
- Bishop, M. (2002). *Computer Security: Art and Science*. Addison Wesley Professional.
- Buschmann, F., Henney, K., & Smith, D. C. (2007). *Pattern-Oriented Software Architecture*. John Wiley & Sons Ltd.
- FMJ Team. (n.d.). *Freedom for media in java*. Retrieved October 21, 2010, from Freedom for media in java: <http://fmj-sf.net/index.php>
- Griffiths, L. (n.d.). *Lejos Vision System*. Retrieved January 15, 2011, from Lejos Vision System: <http://homepage.ntlworld.com/lawrie.griffiths/vision.html>
- Kruchten, P. (1995, November). *4+1 view architecture*. Retrieved November 13, 2010, from University of British Columbia: <http://www.cs.ubc.ca/~gregor/teaching/papers/4+1view-architecture.pdf>
- Motion Project Group. (n.d.). *Motion*. Retrieved December 22, 2011, from Motion.
- Poynton, C. (2006, November 28). *Color FAQ - Frequently Asked Questions Color*. Retrieved January 20, 2011, from Poynton.com: http://www.poynton.com/notes/colour_and_gamma/ColorFAQ.html#RTFTtoC36

Appendices

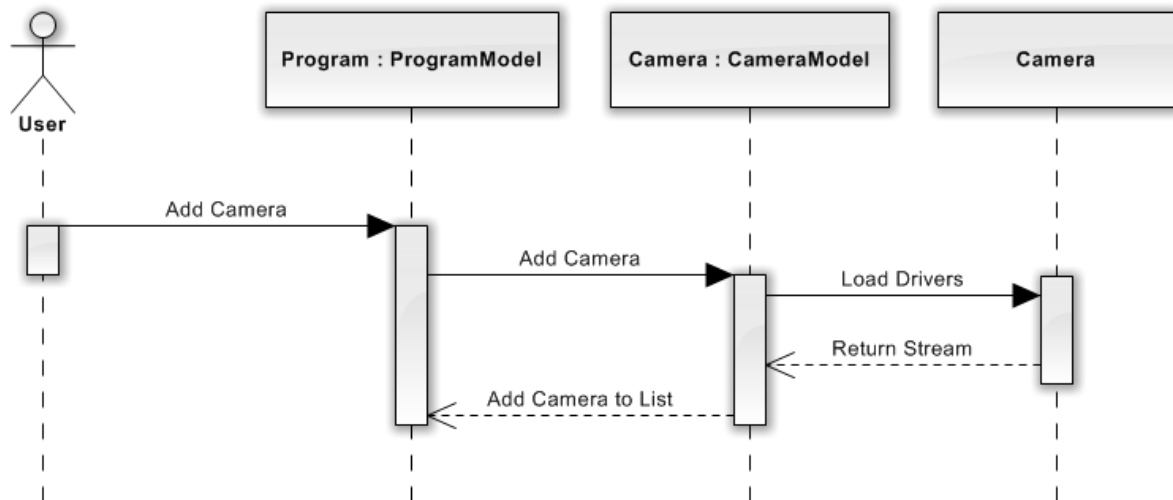
Appendix A

The Logical View - Sequence Diagrams

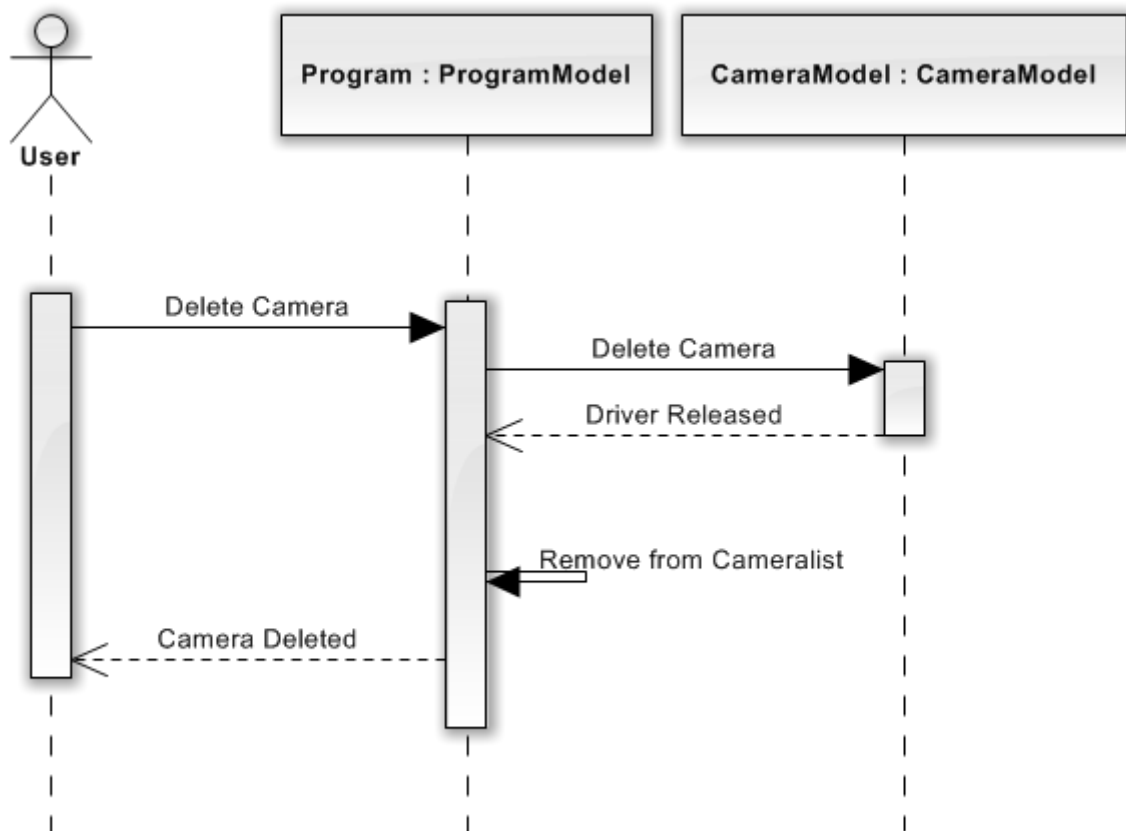
Sequence diagram start security system



Sequence diagram add camera

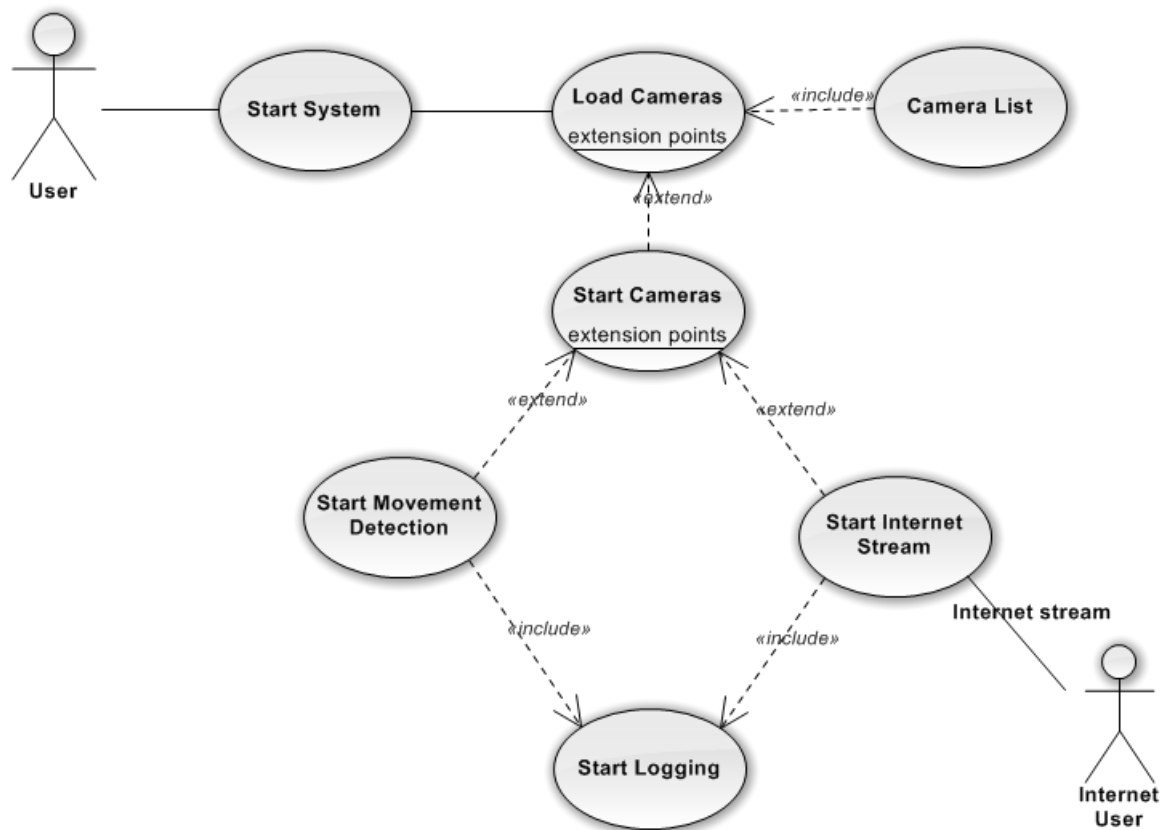


Sequence diagram delete camera



Appendix B

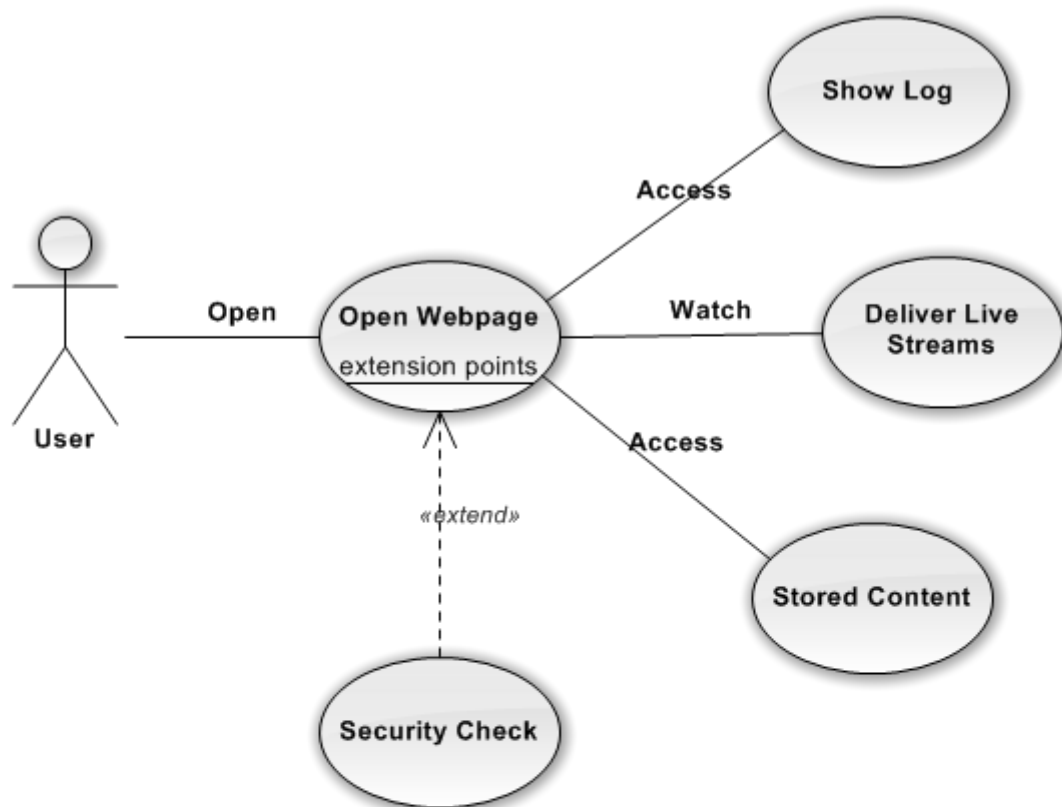
Use Case Scenarios



Use Case Start Security System

1. User starts the system.
2. Load cameras in system
3. Start the cameras
4. Start Movement detection
5. Start internet stream
6. Start logging

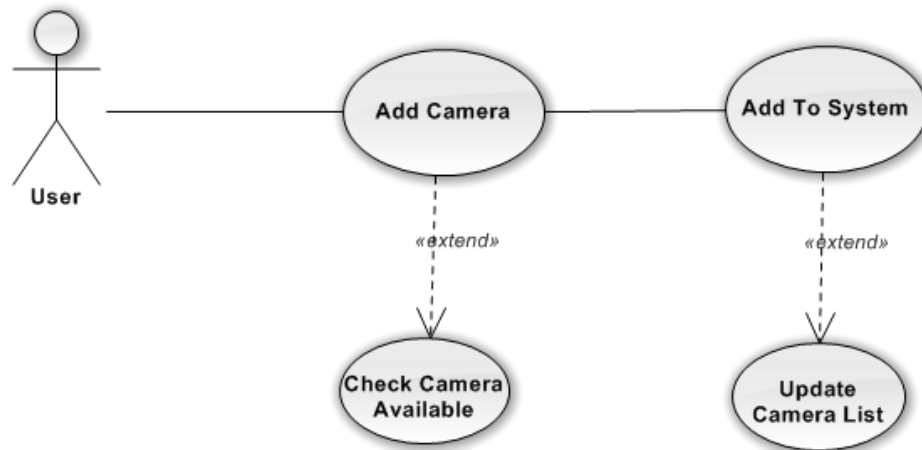
At step 2 load cameras from saved config file.
At step 5 check if there is an internet connection



Use Case Enter System From Internet

1. User opens the webpage.
2. Deliver the webcam streams
3. Deliver the log
4. Deliver the stored content

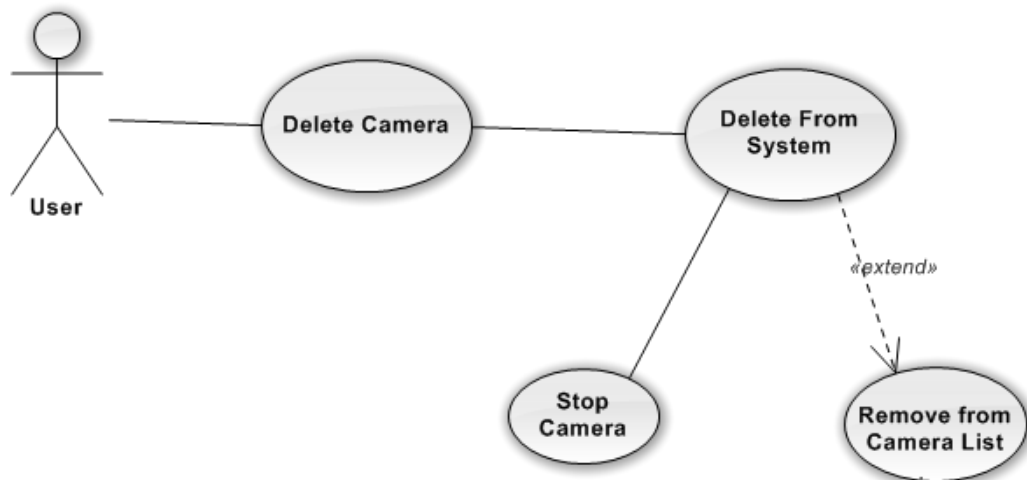
At step 1 the system checks the security. If failed it will log the event and go back to the start screen.



Use Case Add a Camera:

1. User selects and adds a camera
2. Camera is added to the system
3. Camera is added to the list of camera's

At step 1 check if camera is available



Use Case Delete Camera

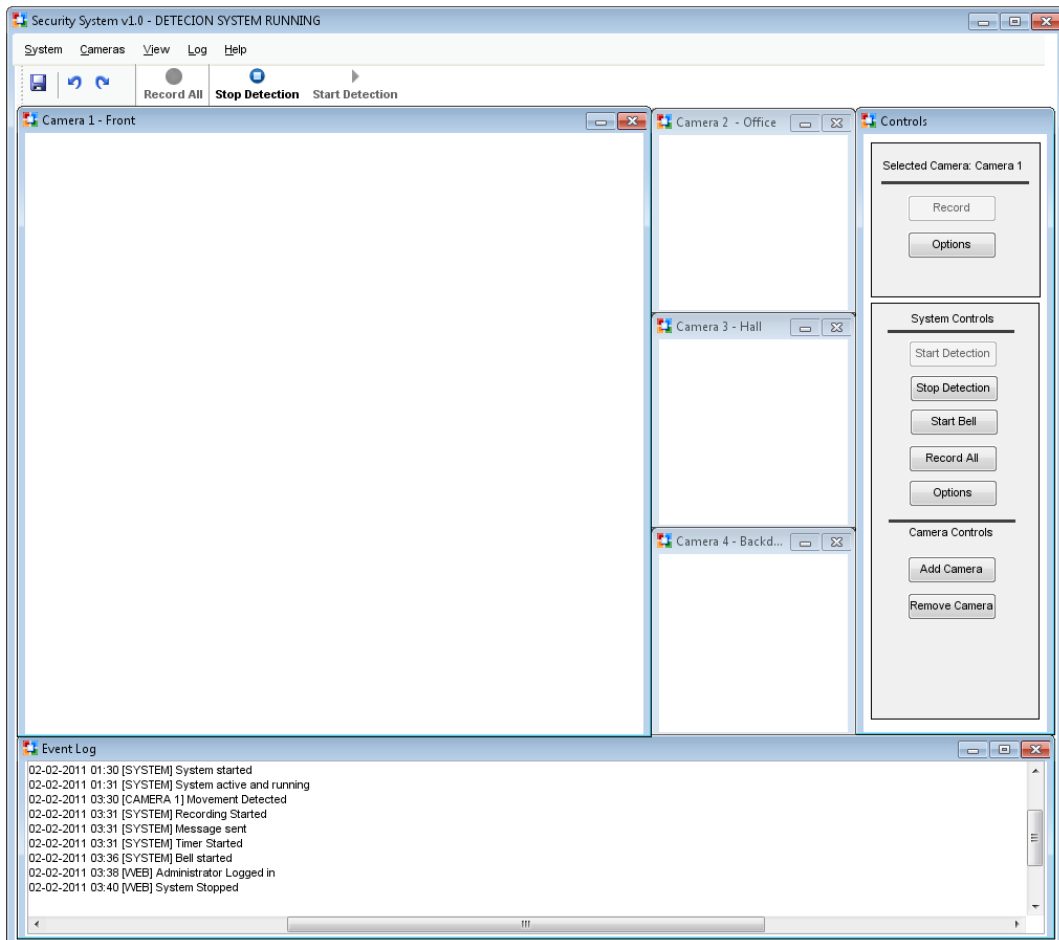
1. User deletest camera
2. Stop camera
3. Delete camera from system
4. Delete camera from camera list

Appendix C

User Interfaces

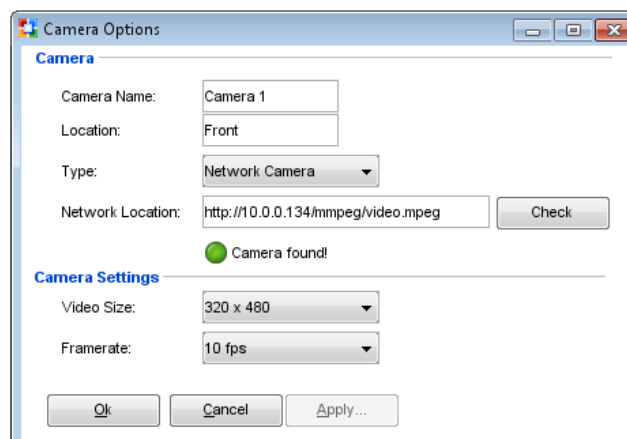
Double click on camera frame

This user interface describes what happens when the user double clicks on a camera frame. The camera frame will resize to double size and the other cameras will get reduced in size.



Camera options user interface:

When there is a click on the options button in the main view the options for a camera will become visible.

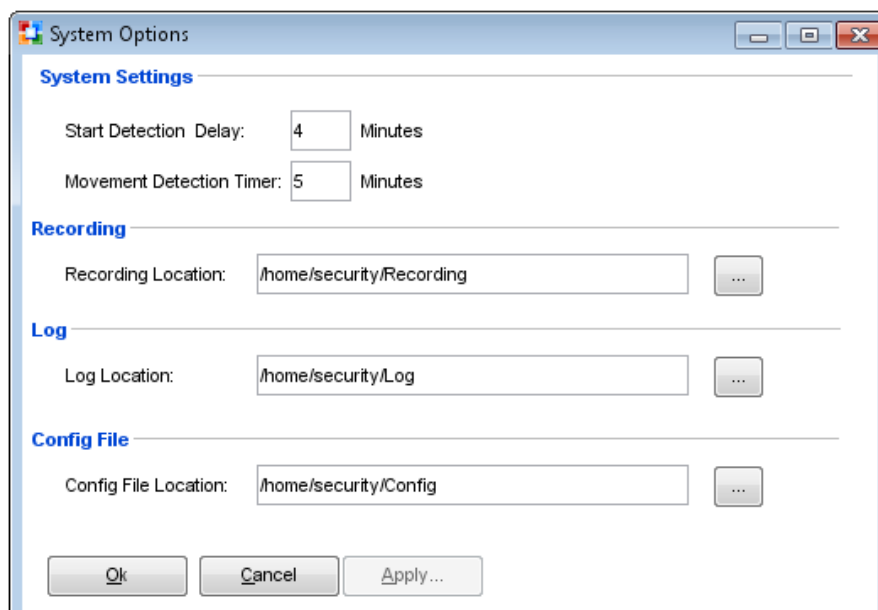


It is possible to enter a camera name and a location. The user can select if it is a network camera or a local camera. If it is a network camera it is necessary to put in the network address. The check button checks if the network camera can be found.

In the camera settings part the user can select the video size and the frame rate of the video.

System options user interface:

This user interface describes the system options.



The screenshot shows a window titled "System Options" with a standard Windows-style title bar (minimize, maximize, close buttons). The window is divided into four sections by horizontal lines:

- System Settings:** Contains two settings:
 - "Start Detection Delay:" with a text input field containing "4" and the label "Minutes".
 - "Movement Detection Timer:" with a text input field containing "5" and the label "Minutes".
- Recording:** Contains one setting:
 - "Recording Location:" with a text input field containing "/home/security/Recording" and a browse button (three dots) to its right.
- Log:** Contains one setting:
 - "Log Location:" with a text input field containing "/home/security/Log" and a browse button (three dots) to its right.
- Config File:** Contains one setting:
 - "Config File Location:" with a text input field containing "/home/security/Config" and a browse button (three dots) to its right.

At the bottom of the window, there are three buttons: "Ok", "Cancel", and "Apply...".

The start detection delay is the time the administrator needs to get out of the building before the detection starts. If the system is started immediately then the alarm will start, because it detects the administrator leaving the building. The movement detection timer can be adjusted. This time is the time between the detected movement and the starting of the bell.

In the bottom part the locations of the recordings, logging file and configuration file can be entered.