University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2019
Industrial IT and Automation

Modeling of building occupation using motion sensor data

Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis 2019
Title: *Modeling of building occupation using motion sensor data*
Pages: *124*
Keywords: *PIRMod, PIRSim, Modeling, Simulation, Unified Process, UML*

Student: *Jørund Martinsen*
Supervisor: *Nils-Olav Skeie, Veralia Gabriela Sánchez*
External partner: *SMART Research group at USN*
Availability: *Open*

Summary:
An adaptive model of occupancy, based on simple motion data is proposed. The model predicts the occupation of the test building with a prediction horizon of one week.

In addition to this model, a simulator is proposed. This simulator has been used to simulate data, similar to the measured data from the test building.

Both the simulator and model have been implemented in C#, as PIRSim and PIRMod. This implementation has been done using the Unified Process and documented using UML documents.

Preface

This report is written to document the project performed as a part of the work of the SMART research group at USN. The goal of the project is to implement a model in C#, using UP, and document the software using UML.

The model is to predict the occupation of a building, using motion sensor data. This model has been implemented, and used to predict a week of occupation. If the data contains too many exceptions, the model will not be able to predict accurately.

In addition to the model, a simulator is implemented. This simulates a configured sensor setup. To complement the data from the simulator, data measured in a building is also used. This building is the home of a SMART group participant. As the data is of a personal nature, it is not available here. Only fragments of this data has been used to demonstrate the function of the model, and evaluate the simulator.

The source code for the software is available online, from [1] and [2] via azure devops services. It is also given to the supervisors via email.

The author would like to extend gratitude to the supervisors and the faculty of USN for their teachings and support.

Porsgrunn, 13th May 2019

Jørund Martinsen

Contents

Nomenclature

Symbol	Explanation
.NET	A software environment
C	A programming language
C#	Programming language
C++	Programming language
CSV	Comma Separated Values
FDUC	Fully Dressed Use Case
FDUCD	Fully Dressed Use Case Document
GUI	General User Interface
MIT	Massachusetts Institute of Technology
MVVM	Model-View-ViewModel
PIR	Passive Infrared sensor
UI	User Interface
UML	Unified Modeling Language
Unix	Operating system
UP	Unified Process
VS	Visual Studio
WPF	Windows Presentation Foundation
XAML	eXtensible Application Markup Language
XML	eXtensible Markup Language

1 Introduction

As SMART house technology has emerged, the need to estimate position of residents has been important for several cases. In this project, a model of PIR sensor data is presented. The model should be able to predict occupation status, now or in the near future. This does not include the location inside the house.
Previous work in this area is summarized in [3]. Here, several approaches and methods are reviewed. Some of these are touched upon here and also implemented in the model. The software is written in WPF, C#, using the Unified Process as a development process. Documentation is done using UML schemes.

1.1 Background

As part of the work at USNs research group SMART, a model of movement in homes is required. This model will enable several other parts of a larger control system to function more efficiently. The main focus of this research is to be able to minimize energy consumption of buildings, while maintaining a comfortable environment. A large part of this work is to minimize heating efforts, as these contribute around two thirds of the energy consumption of residential buildings. [4]
In addition, the model could be used to improve burglary detection and prevention systems.

1.1.1 Previous work

In the previous work, the focus has been to predict where in the building residents are going to be. This has been done by using advanced sensory equipment and other SMART devices. Examples of this is described in Sanchez, Pfeiffer and Skeie, where several other applications are also mentioned. [3]

Dodier, Henze, Tiller *et al.* describe a system where the occupants are tracked using simple motion data, however the goal of the project was to determine future location of occupants. This is close to the work presented here, but does not consider the occupation status of the building. [5]

1.2 This work

A simulator for motion data has been created. This simulator is made for simulating several setups. To test and validate the simulator, a house with a sensor system is used. Data from this house has been gathered since 2018, with additional sensor added in 2019. Section 2.1 describes the setup of the house.

To predict future occupation, a model has been created. This model uses a top-hat model. Several other model types have been considered, as described in section 2.5.

Both of these systems have been implemented and tested using C# and WPF. The development process has been Unified Process.

1.3 Report structure

This report is structured in three main parts; Theory, implementation and results.

Theory

In chapters 2 and 3, different methods for the model and simulator development are described. Chapter 2 describes the overall system, as well as the methods for simulation and modeling. Chapter 3 describes the methods for software development, and the technology used in the implementation.

Implementation

In chapter 4 the implementation of the theory is described. The software is documented by using class diagrams and descriptions. The documents created as a part of UP are also included here.

Results

The simulator and model are compared to biologically created data from the house. In addition the test results are included. These results are then commented on and discussed in the discussion chapter, chapter 6

2 System description

The system consists of a house, with PIR sensors and a centralized logging system. Two adults reside in this house, and trigger the PIR sensors naturally. Each trigger of a sensor is an event. These events are logged to a CSV file.

To develop and test a good model, more types of setups need to be used. To accomplish this, a simulator has been developed. This simulator has been compared to the naturally generated data. It is possible to configure the simulator to generate events for multiple sensors and floors. In addition, different rates of events for different time periods can be configured.

Finally, a model for estimating future occupation has been developed. This model can read the data from the house, or the simulator. It is adaptive, as it uses historic data to retrain itself periodically. Each part of the complete system is designed as a standalone system, so each part is reusable for future development.

2.1 Technical properties

When using a PIR sensor several issues need to be considered. In combination with the limitations of the sensors, there are issues to consider with a system to monitor a building. These issues are described in the following sections.

PIR sensor

A PIR sensor consists of two main parts. These parts are the actual sensor and a Fresnel lens. The sensor detects temperature, while the lens splits the detection area into sections. Separation of detection area allow for motion detection. As an item moves from one section to another, the temperature sensor will spike for each section. Consecutive spikes are then an indication of movement. This is shown in figure 2.1 [6]

The main limitation of this setup is the lack of static detection. If there is no movement, the sensor will not trigger, and the system will not detect occupation. Overcoming this limitation is a part of this work.

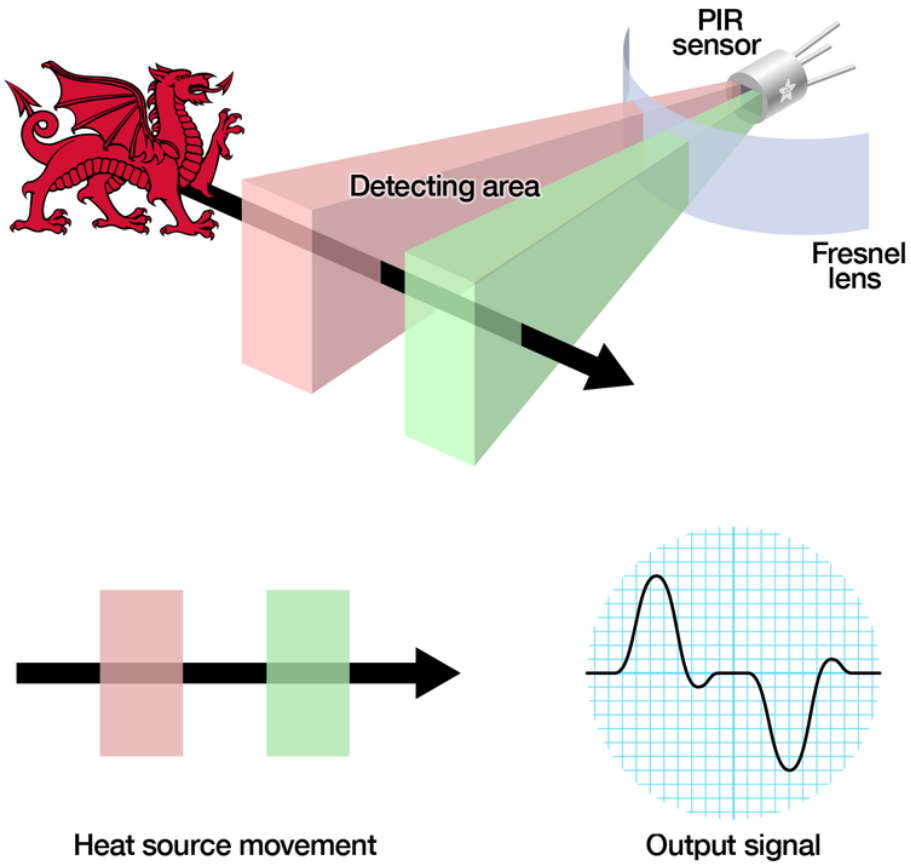


Figure 2.1: PIR sensor function [6]

Sectioning the detection area modifies it. An example of a modified detection area is shown in figure 2.2.

To cover a large complicated area, like a building, several sensors are combined. If one sensor triggers, there is movement in the area. With a detection area like the one in figure 2.2, the location of sensors inside the building is an important factor when creating the system. To enable detection of occupation, all passages should be covered, especially the exit and entry points of the area.

House setup

Biologically created data is generated from a house with PIR sensors mounted at various locations. The building has two floors with sensors. Some of these sensors were installed in 2019. Two sensors were installed prior to 2018. For these two sensors, there is data for most of 2018.

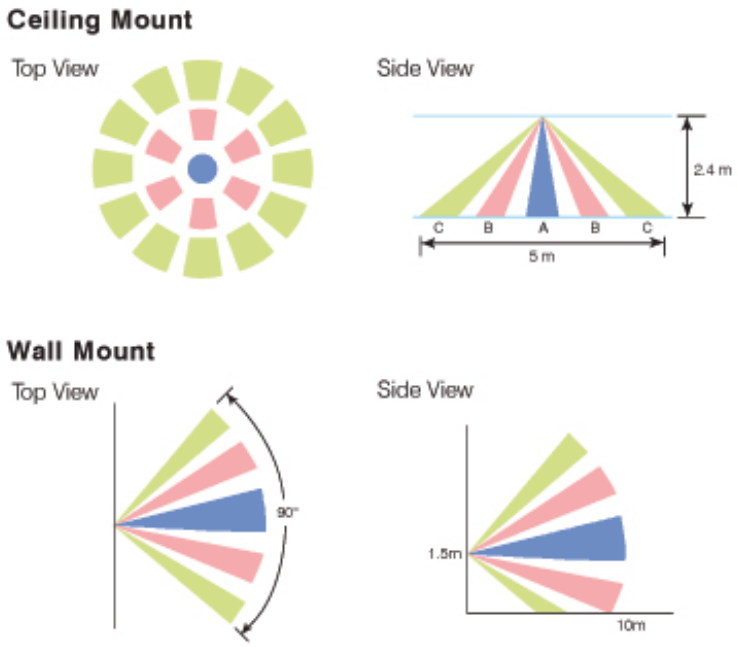


Figure 2.2: Example of detection are for PIR sensor with Fresnel lens. [6]

These two sensors cover the upstairs living room and the kitchen. The newer sensors cover the second living room and the top of the stairs. None of these sensors cover the main door.

2.2 Simulator requirements

The simulator should be able to simulate different setups, like the one in the house. In addition, the simulation should be faster than reality, i.e. simulating a month should not take a month.

Simulated data should also be saved in a CSV file, using the same format as that which is generated in the house. The format is shown in appendix C, which also shows generated data from the simulator.

In addition to this, the simulator and the probabilities of events should be configurable.

2.3 Simulator method

To generate events, a random function is used. This function already exists in the .NET library as a class. This class has the method Next. To be able to adjust the probability of an event, an array of integers is created. The Next function is used to select a random index of this array. Figure 2.3 shows the selection from the array. [7]

However, as the probability of a consecutive event is higher than normal events, the array is filled with the specified ratio of values, plus a margin. The margin area is filled with 1s, and the rest is filled with 2s and 0s.

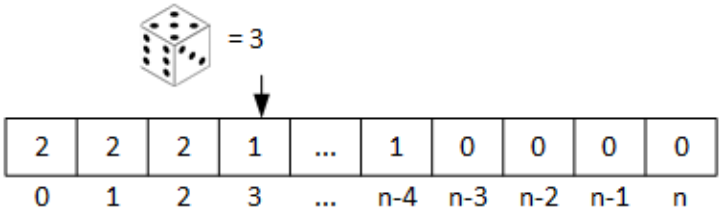


Figure 2.3: Random selection from array

If the randomly selected value is greater than or equal to 2, an event is triggered and the limit is set to 1. If the next value is greater than or equal to 1, a new event is triggered. Should the value be 0, the limit is reset to 2.

2.4 Model requirements

Prediction of occupation enables improvement of heating efficiency. By using a sensor array, that is already installed, the model is a cost efficient method for prediction. The model requirements therefore include the usage of PIR-sensors.

In addition, the model should be adaptive, to handle the change in habits from season to season, changes like summer-time, vacations, spending more time outside and/or less time outside.

The model should predict when occupants leave the house, and when they come back. The process of heating is a slow process, so the accuracy requirement is not very strict. An error of $\pm 10 - 15$ min is acceptable. As the occupation is Boolean, the location is not of interest, nor the number of people.

This is a simpler model than what is presented in the chapter Previous work, and relies entirely on PIR sensors. This makes it available to most consumers, without a high investment. It is also applicable to several areas, such as burglary alarm and vacation light control.

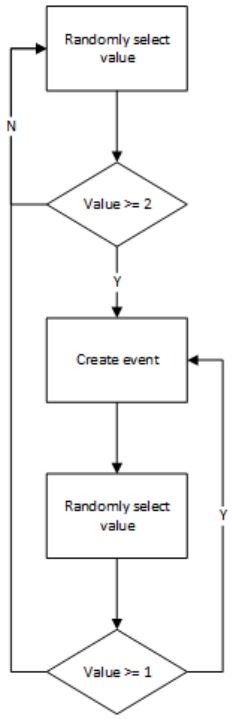


Figure 2.4: Flow chart of event triggering

2.5 Different model methods

To model a process, there are many methods. Some of these methods have been considered and are described here. Each method has some properties that can help model the occupation state of the house. First, it is necessary to describe the process. This is done as a state-space model.

2.5.1 State space model

This process consists of many inputs, outputs and disturbances. However, there is only one state. The process can be described using several states, but here the focus is only occupation.

Figure 2.5 shows a simple state space model. This model has input vector u, disturbance vectors v and w, state vector x and output vector y

The vectors u, v, w and y contain variables to describe the number of occupants, errors in the sensor system, changes in occupant schedule etc. Most of these are not observable to this system. The only observable input is the motion events.

x contains the occupation state. This state is Boolean, and with only one variable. Equation 2.1 shows this. x is of course dependent upon the number of occupants and/or visitors, and their location, but these variables are not observable.

As a consequence of the state space model setup, several methods can be excluded, such as linear regression or other common machine learning methods.

$$x = x_1 = \begin{cases} True, \text{for } u_{\text{number of occupants}} + v > 0 \\ False, \text{for } u_{\text{number of occupants}} + v < 1 \end{cases} \tag{2.1}$$

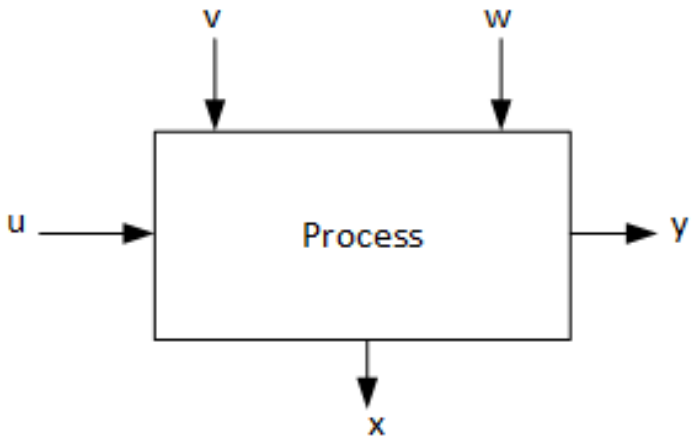


Figure 2.5: State space model

2.5.2 Decision trees

This method consists of several 'decisions' and follows the choice to the next decision. This gives a tree structure as shown in figure 2.6. This figure is an example of part of a complex decision tree. Already, there is uncertainty in the model even without prediction. This is due to the probibalistic nature of the process. [8]

2.5.3 Bayesian models

When there is a set of observations, that infer hidden states, a Bayesian network can be utilized. This has been used successfully in [9], to predict the next location of a person. In this work it is of no interest where the person is going to be, unless it is inside the building.

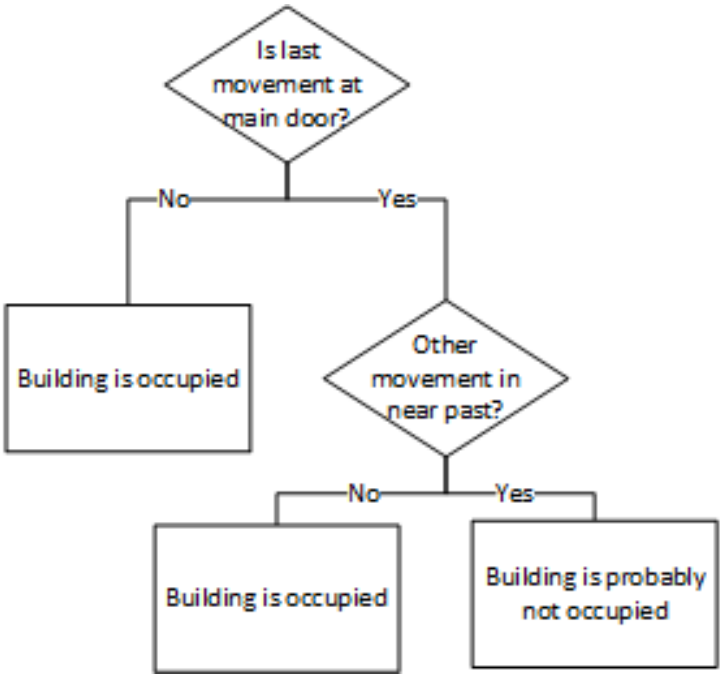


Figure 2.6: Example decision tree

Recursive Bayesian estimation or Bayesian filter is a probabilistic approach to the Kalman filter. Given linear transitions and normally distributed values, the Bayesian filter is equal to a Kalman filter [10, p. 56].

2.5.4 Markov chain

The Markov chain is a special implementation of the Bayesian network. Utilizing the present state, the next states can be inferred. There are many implementations of the Markov chain, including Hidden Markov Model(HMM).

This method has been implemented and tested in the work of Dodier, Henze, Tiller *et al.*, where it showed the capability to estimate occupation of individual offices. [5] This is at a greater detail than the goal of this work.

2.5.5 Neural network

Neural networks are a way to find good estimates of a function. This requires a large dataset, containing both input data and output data. As the occupancy is not directly measured, this data is unavailable. Therefore, neural networks are not a good candidate for prediction of future state of this system. [11]

2.5.6 Probability function

As an approximation, it is suggested to use a function to estimate the probability of occupancy. These functions should give an approximation of the probability of occupancy, based on the previous events. Two functions have been evaluated. These are a triangle function and a top-hat function.

To create a more complex model, several functions are combined as shown in equation 2.2

$$F(x) = max(f_1(x), f_2(x), \ldots, f_{n-1}(x), f_n) \tag{2.2}$$

Triangle function

The triangle function is a simple triangle, consisting of two linear functions. This approximates the probability of occupation, which is between 0 and 1. A triangle function is seen in figure 2.7.

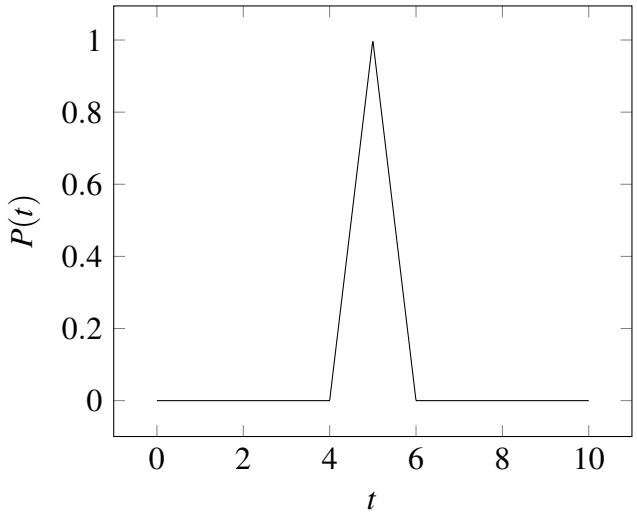


Figure 2.7: Triangle function example. Plot of equation 2.3, where $a = 4$ and $b = 6$

$$P(t) = max(0, min(x-a, b-x)) \tag{2.3}$$

Smoothed Top Hat Function

A similar function to the triangle is the top hat. There are two main differences that make the top hat a better model candidate. Firstly, the plateau, where the function is 1 for a defined period. Secondly, the inclination from 0 to 1 is a smooth sigmoid. This is a better representation of the probability of occupation, as there is a period of time where there is high certainty of occupation, before this moves towards 0. Figure 2.8 shows a top hat function. There are several parameters to define the shape of the top hat. These are listed in table 2.1. The equation for this is shown in equations 2 - 5 in Boyd. [12]

Table 2.1: Smoothed Top hat parameters

Parameter	Comment
C	Center of top hat function. This enables a shift of the function.
χ	Physical region. This defines the area where the function is 1.
Ψ	Smoothing region. The area where the function moves between 0 and 1
L	Defines the shape of the smoothing region.

Combining the smoothed top hat with the max function gives an approximation of the probability of occupation throughout a given timespan. It also allows for a complex function that includes several regions where the probability is 1. An example of two regions is shown in figure 2.9

Definition of model regions

To combine several top hat functions, the regions must be defined. This is done by finding the first and last events in a series of events. This is described in section 4.2.2 and 4.2.3.

Each region is set to before and after a point in the day. This yields two regions per day, one for the morning and one for the afternoon. Each region has one top hat, with parameters.

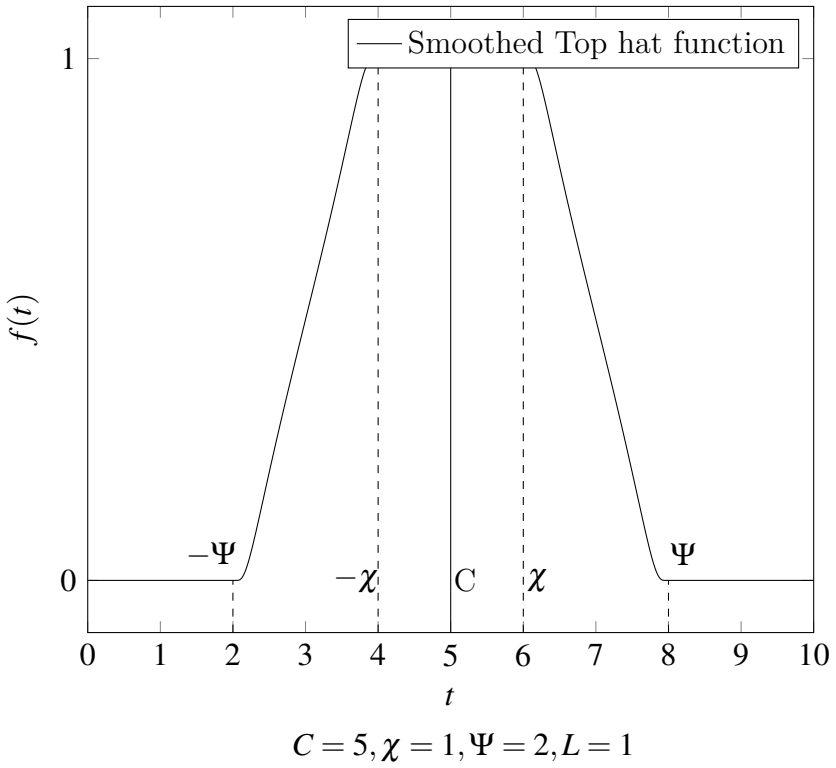


Figure 2.8: Smoothed Top Hat function

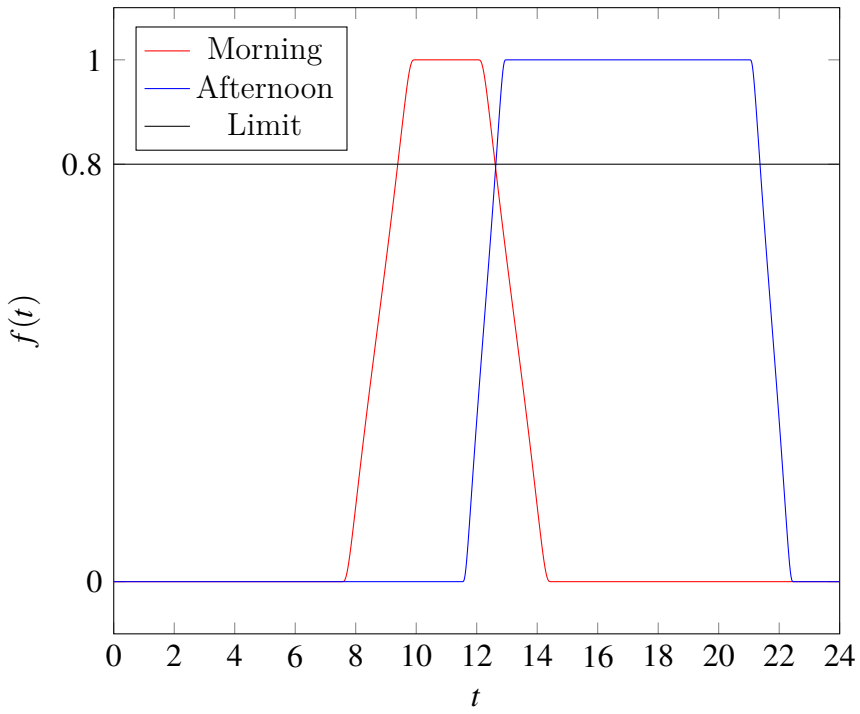


Figure 2.9: Two Top hat functions overlapping

3 Software development methods

When developing software, there are a lot of development process to select from. Of these, the Unified Process is chosen. This process defines a work flow and a few documents. These documents aid the development process.

In the following sections, the Unified Process and the documents are explained and put in context with the development of the software.

In addition to the process, several technologies are used. These technologies are Windows Presentation Foundation, C#, Prism and Unit testing. Sections 3.2 to 3.5 describe these technologies and why they were chosen.

3.1 Unified Process

Among the many development processes, the Unified Process is one of the most descriptive. This process is also iterative, meaning it follows a defined pattern repeatedly.

Each step of the process is called an iteration. These iterations are organized in four project phases. These phases are:

1. Inception, where the project is planned, estimated and potentially started.
2. Elaboration, where the use cases and requirements are identified and architecture decided.
3. Construction, where the software is written, based on the Elaboration phase.
4. Transition, where the software is released.

Phase 2-4 builds upon the work of the previous phase. The following sections elaborate on each phase.

3.1.1 Inception

The first phase of a project is the inception phase. Here, the project is evaluated and cost estimations are performed. Based on these evaluations and estimations the project is moved to the next phase or terminated. The UP requires business use cases to be created at this stage.

For this project, the inception phase consisted of a brief literature study and defining the business use cases. As the development of the system is a part of the assignment, the cost estimation and the decision to carry out the project has already been made. The business use cases are also a part of the assignment, shown in appendix A.

A progress plan has also been made, describing the number of iterations. This is available as a GANTT chart in appendix B.

3.1.2 Elaboration

After the inception phase, the elaboration phase starts. This phase contains definitions of requirements, architecture, technology to be used, how many iterations and so on. The documents produced in this phase consist of use cases for the system,

Figures 3.1 and 3.2 show the use cases made for the simulator and model. Here, the use case is a short definition of the interaction between the user and the system.

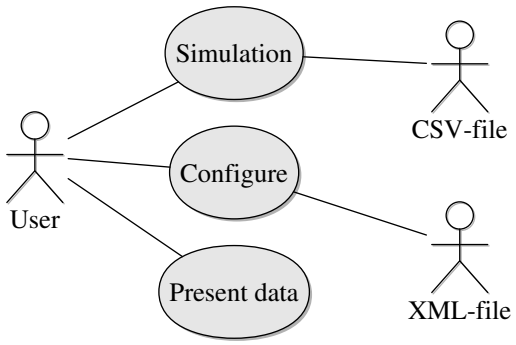


Figure 3.1: Simulator use cases

3.1.3 Construction

This is the phase where the software is written. Here, several iterations have been repeated. Each iteration generates several documents. The documents are the fully dressed use case, interaction diagram and class diagram. The interaction and class diagrams are updated in each iteration. The fully dressed use cases are made for each use case from

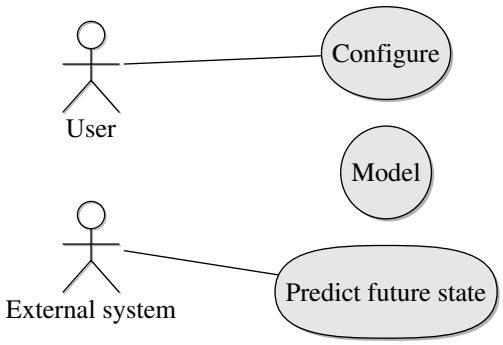


Figure 3.2: Model use cases

the elaboration phase, and as each iteration handles one use case, there is one FDUC for each iteration.

Fully Dressed Use Case

A FDUC consists of several sections. These sections are shown in table 3.1. Each section is numbered and the main parts are in bold text. Section 8 describes the main flow of the function, and section 9 describes the error handling or other exceptions, related to the items in section 8.

Appendices E and F shows the FDUCDs as developed. These appendices also contain the class and interaction diagrams

Class diagram

As C# is an object oriented language, the use of classes and objects is necessary. To keep an overview of the architecture, one or more class diagrams are used. The class diagram is a representation of each class in the software, and the properties and methods within. It also shows the dependencies as links. The main class diagram for the simulator is shown in figure 3.3. Here, several classes are shown. These classes have defined parameters and methods, with inputs and a return type. The + and − signs indicate the access levels. To indicate interfaces, a stereotype is used, indicated by << and >>. Associations are shown as simple lines, while inheritance is shown with an arrow, pointing to the class inherited from.

Interaction diagram

As the class diagram shows dependencies, properties and functions, it does not indicate behavior over time. To show this behavior, an interaction diagram is used. One form of

Table 3.1: FDUC template [13]

Section	Comment
1. Use case name	Start with a verb
2. Scope	The system under design
3. Level	'User goal' or 'Sub function'
4. Primary Actor	The main user of the function
5. Stakeholders and interests	Who cares and what they want
6. Preconditions	What must be fulfilled before starting
7. Success guarantee	What must be fulfilled at a successful completion
8. Main Success scenario	8.1 - A typical set of events
9. Extensions	9.1.a - Alternative scenarios of success or failure
10. Special requirements	Related non-functional requirements
11. Technology list	Different I/O methods and data formats
12. Frequency of occurence	Investigation, testing and timing of implementation
13. Miscellaneous	Such as open issues

interaction diagrams is the sequence diagram.

A sequence diagram shows the threading, timing and interaction of the objects. Figure 3.4 shows a sequence diagram for the simulator. This diagram shows a continuation of the initialization of the simulator, the first part of which is shown in appendix E. In addition it shows the binding event handling, which is described in section 3.2.2.

Each instance is shown as a box with a 'lifeline'. These lifelines are used to show interaction between the instances, as arrows with a function call name. To group function calls, a comment frame is used.

3.2 Technology

The software is developed using C#, WPF and Prism. C# is a programming language in the .NET suite. In combination with this language, WPF is used. This defines a

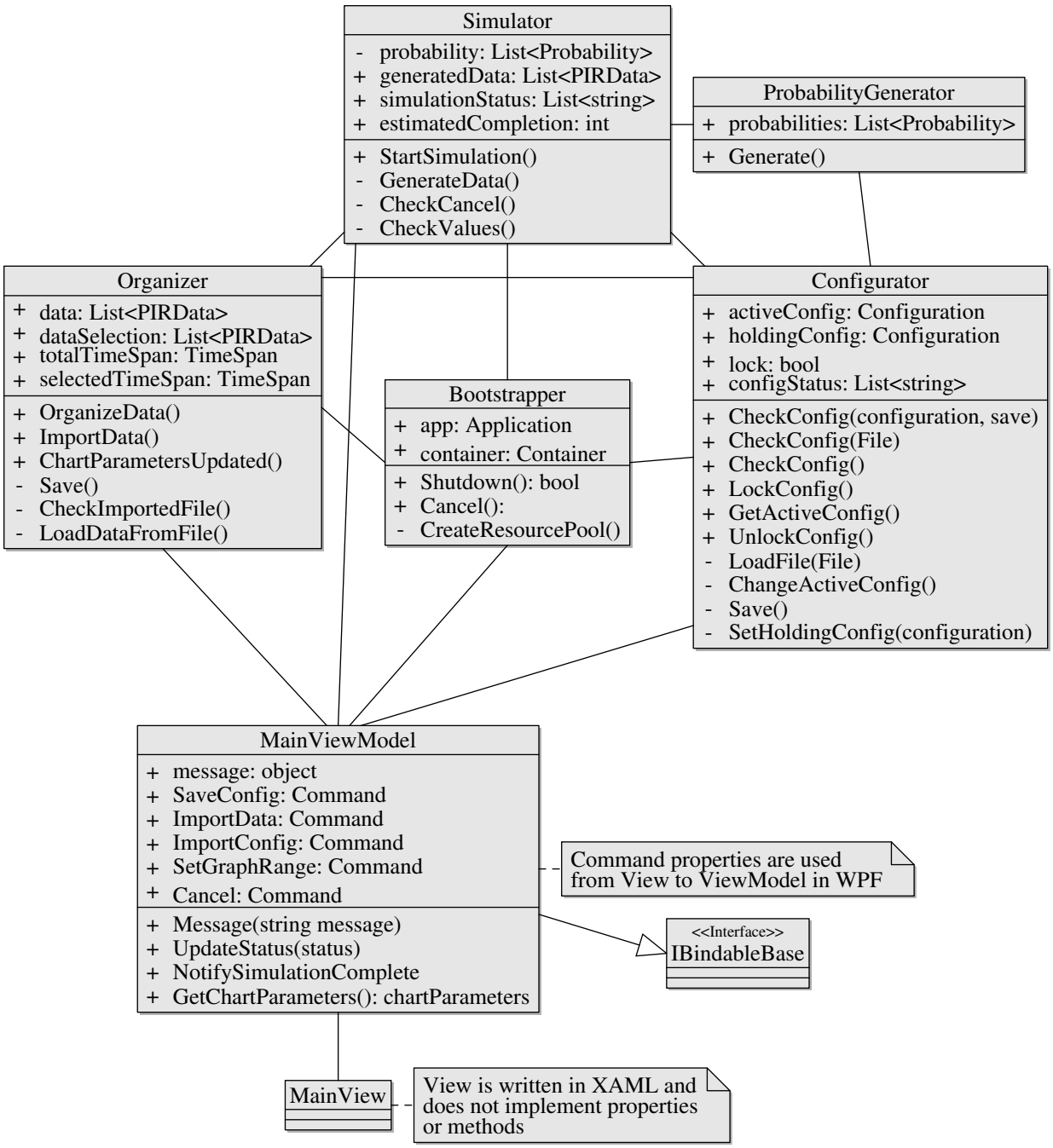


Figure 3.3: Main class diagram - Simulator

different language for the frontend language, XAML. To connect the frontend to backend, the Prism library is used. The following sections cover these topics, in addition to the development architecture used.

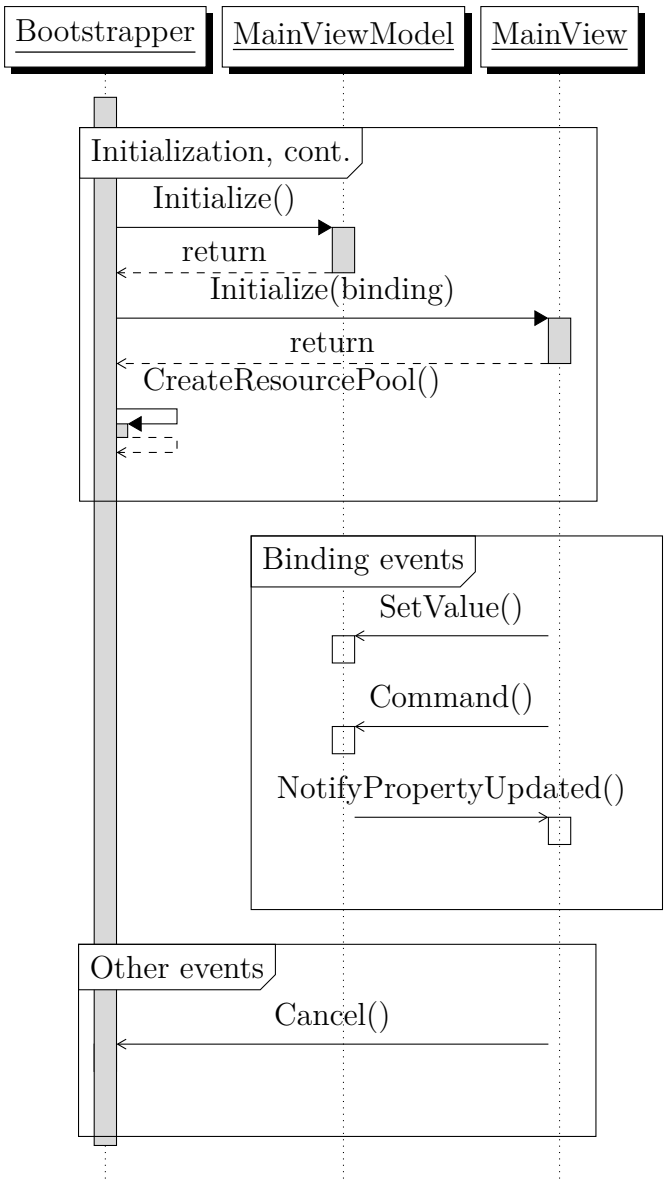


Figure 3.4: Interaction diagram - Simulator

3.2.1 Programming languages

There are many programming languages to choose from. Each language has its advantages and disadvantages, and these are too many to list here. However, some comments on the most popular languages are included.

.NET is a suite of several languages and functions. These are documented in an online library from Microsoft. [14]

C family

C is one of the oldest software languages. This language is cross-platform between Microsoft products and Unix systems. As such it is also very detailed and memory handling must be configured by the developer. [15]

However, as it is interacting closely with the hardware, it is capable of very fast computations. This advantage has been reduced due to better hardware. Another disadvantage is that it is sequentially oriented.

A competitor to the C language is the C++language. This is similar to C, but is object oriented.

Finally, the C# language, which is object-oriented and supports high abstraction. It does not support Unix systems, and is slower than its sibling languages. It has automatic memory handling and the syntax is simpler. It is similar to Java, but benefits from the functions of the .NET suite.

Java

Java is similar to C#, but is not in the .NET suite. It does support many platforms, and is a popular language for embedded devices.

3.2.2 WPF and XAML

Windows Presentation Foundation is a UI framework. It allows for defining the UI in XAML, a markup language. This setup allows for a clear separation of frontend and backend. There is also a function in WPF where bindings can be made. These bindings tie the data shown in the interface to the data in the system. In addition, commands can be invoked. [16]

This allows for a multithreaded program, with clear connections between the UI and the data behind it. To create a proper architecture, with separation of logic, data and UI, the Prism framework is used. This framework handles the binding and commands to simplify usage of the MVVM pattern by predefined event handlers and methods. [17]

3.2.3 Model-View-ViewModel

The architecture is chosen for its separation of logic, data and UI. This separation allows for re-usage of all parts, simplifying the implementation of this model in the external system. It is also a natural choice in combination with the other technologies. Figure 3.5 shows the MVVM pattern, with a databinding layer. There are three parts to the MVVM pattern, each explained in the following subsections. [18]

Model

The model consists of the data. This is where the events, configuration and model/simulator parameters are stored during runtime. As this layer is designed to be a storage layer, there should be little or no logic.

ViewModel

This layer is responsible for the logic and exposing data to the view. This is also where the commands are executed and error handling performed.

View

This is the UI layer. It utilizes binding to be able to connect to the ViewModel. In the Prism framework, each View is connected to a ViewModel, by using a naming convention. This convention is 'Name'View, 'Name'ViewModel and 'Name'Model. [17]

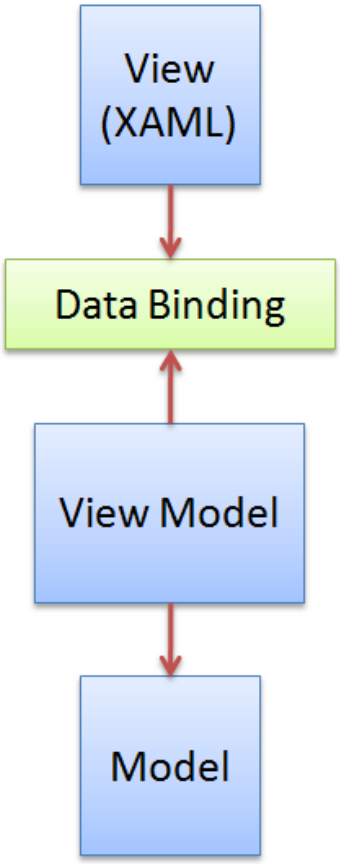


Figure 3.5: Model-View-ViewModel [18]

4 System development

The system is developed as two separate programs. These programs are the simulator and the model. These programs will be called 'PIRSim' and 'PIRMod'.

PIRSim is a standalone program for simulating events in a building. It produces a log file in a specified format that PIRMod can read. This is described in section 4.1. This section is separated in three parts, representing each use case.

PIRMod reads events from a log file, and generates a model from the data in that log file. Usually there are a lot of events per day, and most of these events occur consecutively. All events except the first and last of a series do not contribute any information about the occupation. Because of this, the data must be treated to a selection algorithm. The first selection is to find the bounds of series. The second is to select what series to use for the model parameters. This is explained in section 4.2. The first selection is described in section 4.2.2, the second in 4.2.3.

Testing plans have been made for the PIRSim software, and these tests have been performed. For the PIRMod software, Unit test plans have been implemented. These test are programmed in advance, and are run on each compilation. These unit tests are available in the source code. The test plan for PIRSim is available in appendix G. [19]

4.1 PIRSim

PIRSim generates data for a user specified time period. There are several parameters the user can set. These parameters are listed in table 4.1, with description of each parameter.

When developing the simulator, the configuration had to be created first. All other functions depend on the configuration and its parameters. Therefore, the configuration was created in the first iteration.

The second iteration elaborated the simulation use case. This function is responsible for creating the event data.

Finally, the 'present data' function was created. Here, the data is organized and shown in a chart in the UI.

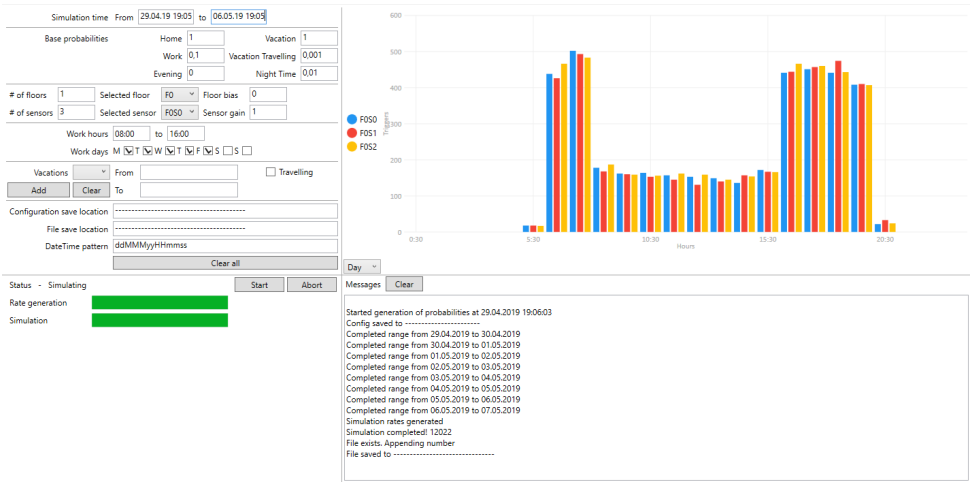


Figure 4.1: PIRSim user interface

4.1.1 Configure use case

To simplify debugging of the configuration, the GUI was created simultaneously. In the subsequent iterations, major changes have been done to both the configuration and the GUI. These changes have been performed after learning of other requirements and capabilities of WPF, MVVM and other technologies in the project.

The user interface shows the configuration, progress bars, a chart area and a notification area. All this is shown in figure 4.1. Table 4.1 shows the parameters in the configuration. Each parameter is represented by a control in the UI. The UI controls are bound to the ViewModel, where these parameters are exposed.

The ViewModel is connected to the Model, which holds the configuration. In addition, the ViewModel has the simulator and the event generator. In appendix E, the class diagrams show these connections.

In order to handle errors and exceptions, a message function has been created. This function adds the message to a string, which the UI subscribes to. This error handling gives a good indication of any errors that occur. It also simplifies debugging. Each part of the system throws events upon error messages or status messages. The ViewModel listens to these events and fires the message function.

4.1.2 Simulation use case

The second major function to be implemented was the actual generation of simulation data and saving it to file. This is separated in two main parts, the simulation and the organization of the data.

Probability generator and simulator

To speed up simulation, the probabilities are generated first. These probabilities are generated for each minute for the entire time period, for each sensor on each floor. The formula for the probability is shown in equation 4.1, where 'Base probability' is the probability at that time of day. For a reference of base probabilities, see table 4.1.

$$\text{Floor bias} + \text{Sensor gain} \times \text{Base probability} \tag{4.1}$$

This list of probabilities are then made available to the simulator, which uses the probability to create the array as described in section 2.3. To simulate, a for-loop is used, iterating through the entire timespan. In each iteration, the flow chart in figure 2.4 is repeated.

Organizer

All events are stored in a list of events. This list is passed to an organizer, where it is organized and saved in the proper format. This format is shown in appendix C, where some example data can be seen. The table in this appendix is generated from a simulated CSV file, using a LaTeXpackage, pgfplotstable [20].

4.1.3 Present data use case

As there is a lot of Boolean data, the presentation of this data is not readable. Therefore metadata is used in a histogram, showing the number of events grouped by a time period. This time period can be specified by the user and ranges from days to months. When selecting days, the plot displays number of events per hour, summing all days, as shown in figure 4.1. If the user selects week, the groups are the days of the week. For month, the days of the month.

This selection and generation of metadata is handled by the organizer. By exposing the data from the organizer to the UI, binding is possible. The controller used for displaying the histogram is from LiveCharts, published under the MIT license on GitHub. [21]

Table 4.1: Simulator parameters

Parameter	Description
Simulation time	Time to simulate
From	
To	
Base probabilities	Probability of event at different times of day
Vacation	$t \in t_{Vacation}$
Vacation travelling	$t \in t_{Vacation}$ and travelling is true
Working	When occupant is at work
Nighttime	$00:00 < t < 06:00$
Evening	$20:00 < t < 00:00$
Home	When occupant is home
Sensor setup	Set sensor configuration
# of floors	Sets number of floors. Each floor has a name, sensor bias and an array of sensors
Selected floor	Floor to configure
Floor bias	Probability to add to each sensor of the floor
# of sensors	Sets number of sensors on the selected floor
Selected sensor	Sensor to configure
Sensor gain	Probability gain for selected sensor
Work time	Sets the work schedule
Work hours	Sets the work hours
Work days	Sets the workdays of the week
Vacations	Allows for configuring vacations
From	Datetime for when the vacation starts
To	Datetime for when the vacation ends
Travelling	Sets if the vacation is spent travelling
Configuration settings	Sets the save location of configuration and data
Configuration save location	Sets the save location for the configuration
Data save location	Sets the save location for the generated data
DateTime pattern	Sets the pattern used for the timestamp in the data file

Parameters
Nighttime - 0.01
Evening - 0.05
Home - 0.1
Other - 0.0

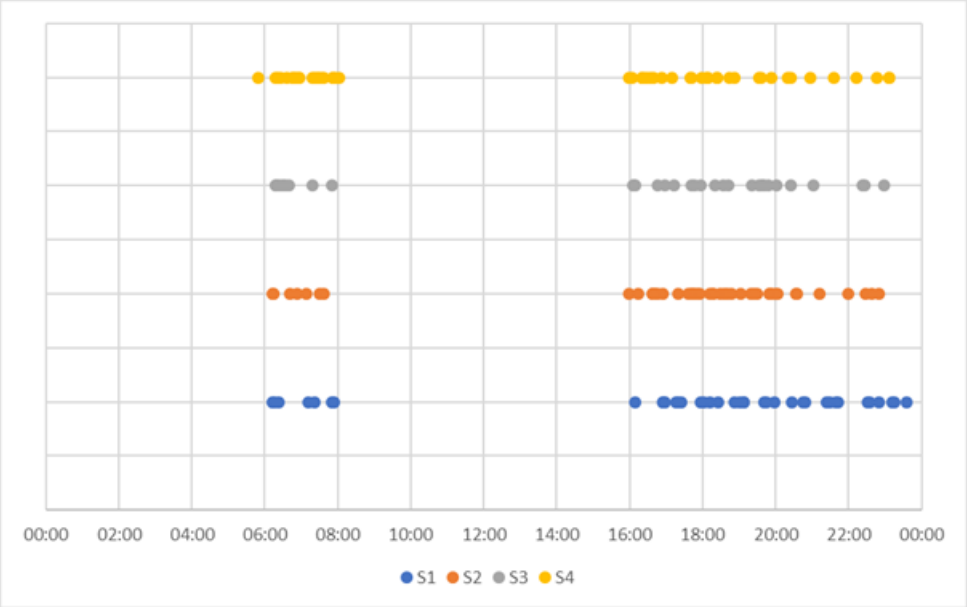


Figure 4.2: Simulation of one day

4.1.4 Simulation realism

To verify the function of the simulator, the generated data is compared to the biological data. This comparison is made by loading biological data into the simulator, where it is plotted using the day setting, as described in section 4.1.3. This gives the two plots shown in figures 4.3 and 4.4.

The biological data is gathered from 01.01.18 to 30.11.18, using two sensors. The events are mostly registered during the morning and afternoon. This justifies the use of two main periods in the model, as described in section 2.5.6.

Compared to the simulated data, the main difference is the smooth transition between different periods of time in the biological data. There is also some events during the night.

In figure 4.4, the same time period has been used. By tuning the simulation parameters, it is possible to adjust these results. If more time periods are added, the smooth transitions are possible to achieve, however this will increase the number of parameters.

4.2 PIRMod

The model is based upon a top hat function as described in section 2.5.6. This top hat is used to define a probability function over a time period.

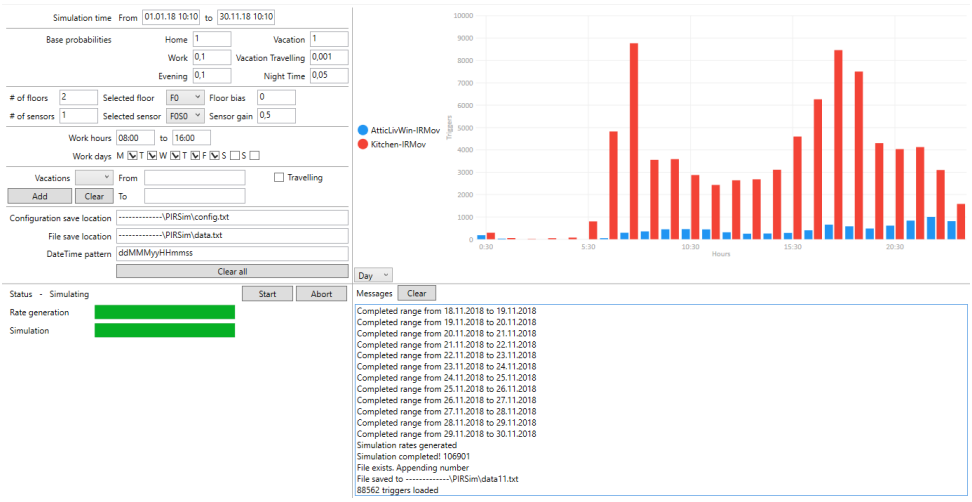


Figure 4.3: Loaded biological data and plotted with day setting in PIRSim

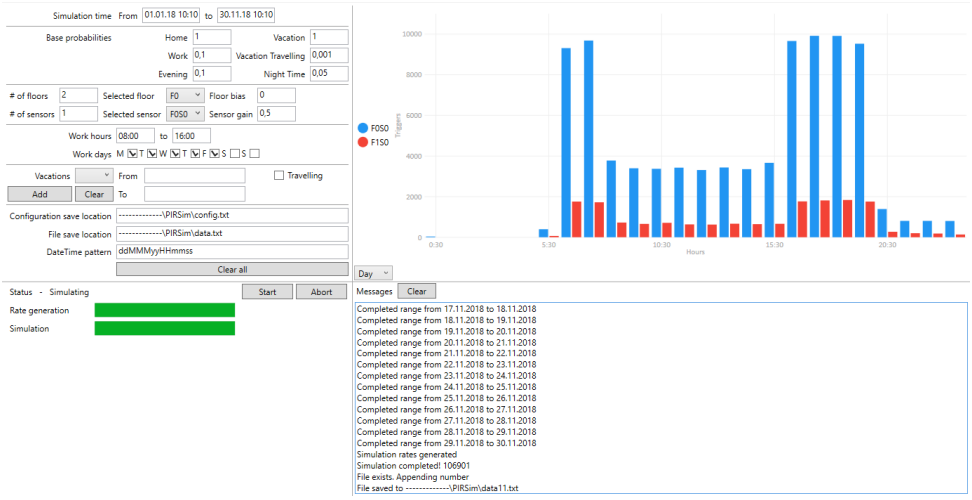


Figure 4.4: Simulated data for the same time period as the biological data

The top hat function does not describe a complete day or any length of time accurately. To extend the prediction horizon for the model, several top hats are combined. Sections 4.2.1, 4.2.2 and 4.2.3 describes how this combination is designed.

PIRMod is created, using the same architecture and technology as PIRSim. The exception is that the modeling functions are separated in a pure C# library and imported into the main software.

The use cases of PIRMod are described in sections 4.2.4, 4.2.5 and 4.2.6. Each use case has, as with PIRSim, several documents to describe the function of the software. These documents are available in appendix F. The use case diagram can also be seen in figure 3.2.

Results of the model are described in section 5.3. The UI is also shown there.

4.2.1 Top hat combination

Each day of the modeling time period is divided in two regions. These regions are assumed to be representative of an event series, as shown in figure 4.3 and described in section 4.1.4 and 4.2.2.

Each event defines a movement, and therefore, the probability of occupation is 1 at that point in time. Such events occur in a series, as movement usually is continuous for a certain time period.

If there is a long time between two movements, the building is probably not occupied during that period. If it is a short time, the building should be occupied. Movement periods occurring with a short time period between them, is therefore defined as a single event series. These event series are used to define the parameters for the top hats.

4.2.2 Event series bounds

The first period is in the morning, after residents wake up and perform their morning routine. Then most people leave for work or other activities. The second period is then the afternoon, when residents come home after their activity. This makes for a natural divide in the day. There are exceptions for this, like the weekend.

The top hat allows for handling exceptions in the smoothing region. If there is an overlap of the two top hats, the estimated probability can stay above the confidence limit, as shown in figure 2.9.

These periods are different for office or other business buildings, where there, usually, is one main period.

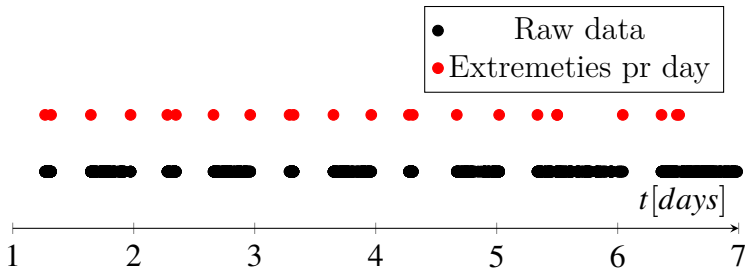
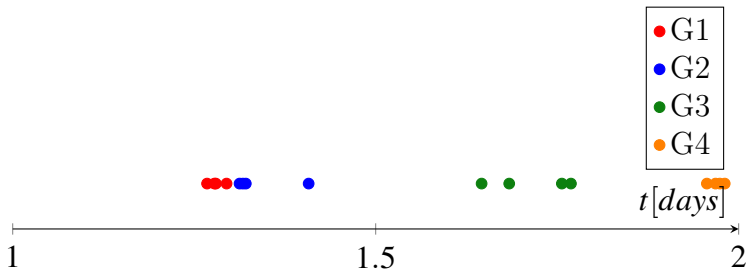


Figure 4.5: Event extremities example

To extend the model further, several days are combined to a prediction horizon of one week. The pattern throughout the week is estimated using data from the previous n weeks. Data from each weekday is grouped and the extremities in the data is found. An example of this is shown in 4.5. These extremities are found for each weekday of several weeks, creating a set of candidates for parameters.

4.2.3 Parameter generation

After finding candidates for the desired time periods, the candidates are evaluated. This evaluation handles outliers and selects witch candidates are used as parameters. Figure 4.6 shows four groups of candidates for a single day. These groups represent the extremities of that weekday from 4 weeks of data.



	Morning (G1)	Noon (G2)	Afternoon (G3)	Evening (G4)
W1	1.27	1.32	1.65	1.97
W2	1.28	1.32	1.76	1.96
W3	1.29	1.41	1.68	1.98
W4	1.28	1.31	1.77	1.97

Figure 4.6: Candidates example with matrix

The parameters are found from the four groups. By finding the middle point of the first in G1 and the last in G2, and comparing each point to this middle point, the smallest and largest distances are used as parameters for the top-hat model. Repeating this for each day of the week and for G3 and G4, yields a complete model for an entire week. The resulting parameters consist of a matrix of four by n values. The middle point is used for C, L is set as a constant, the smaller distance is used for χ and the larger for Psi

4.2.4 Predict future state use case

To predict the future state, several functions must be implemented. These functions are the top function itself, the combination of several top hats and the search for candidate

parameters. This use case is concentrated around the two functions regarding the top hats.

To allow for reuse, the model is contained in a separate VS project, a library. In future systems, this library can easily be imported or referenced.

There is no defined model in PIRMod, as this role is filled by the model library and its functions. These functions are referenced and reused in the ViewModel, where they are made available to the View. This View is constructed in the same manner as with PIRSim. The main difference is the plotting and configuration areas.

The plotting function shows the predicted future state for the entire prediction horizon, along with the identified candidates.

4.2.5 Model use case

The identification of candidates and model properties is also included in the library. This function is created as described in section 4.2.3.

To find candidates, the events must be made available to the software. PIRMod is configured to be able to read the files generated by PIRSim, or the biologically created data file. These files contain the events.

After importing the events, these are iterated through and candidates are identified. After all candidates are identified, the average of each candidate group is found. If there are outliers from this average, they are removed and a new average is found and the process is repeated until no more outliers are found or a quarter of the candidates are removed.

The final set of candidates are used to create the parameters for the combined top hats.

4.2.6 Configure use case

The final use case is to allow for configuring the model. The configuration includes any planned vacations, the location of data files, limits for finding candidates and model mode. The mode is either business or residential. The difference between the modes is whether the building is occupied at night or not.

The probability of occupation is set to 0.95 at night, unless the last sensor triggered is an exit sensor. Should the last sensor be an exit sensor, it is assumed the building was left, and is therefore not occupied.

In addition, the 'Now' time can be set manually. This is for testing purposes and is not intended to be a part of the model.

5 Results

This chapter evaluates the PIRSim and PIRMod software. The first part handles the PIRSim software. Here the main focus is the speed performance and the similarity compared to biologically created data.

The second part evaluates the model created in PIRMod. Here, the model is created from biological data and then it is compared to the following week of data. As the biological data is gathered from a Norwegian household, there are several exceptions from the average week. In Norway, there were 251 working days in 2018. [22]

5.1 PIRSim speed performance

A requirement for the PIRSim software is a simulation time less than the time simulated. This requirement is met, as shown in table 5.1 and figure 5.2. These times are the average of 10 samples, gathered from 4 different settings. These settings are shown in figure 5.1. For different experiments, the setting changes, specifically number of sensors and simulation time.

Table 5.1: Average simulation times

Experiment	**Probabilities**	**Simulation**	**Presentation**
1 Month 1 Sensor (1)	1.76	1.89	0.08
1 Month 5 Sensors (2)	8.69	9.66	0.40
3 Months 1 Sensor (3)	11.99	5.64	0.18
3 Months 5 Sensors (4)	59.39	27.91	1.07

As these figures and table show, as the number of sensors increases, so does the computational time. However, the time span contributes more than the number of sensors, 5 times the sensors has approximately the same computational time as 3 times the timespan.

5.2 Simulation data comparison

Table 5.2 shows two random segments of data. One from the biological data and one from the simulated. The data is the same as shown in figures 4.3 and 4.4. From the table, the

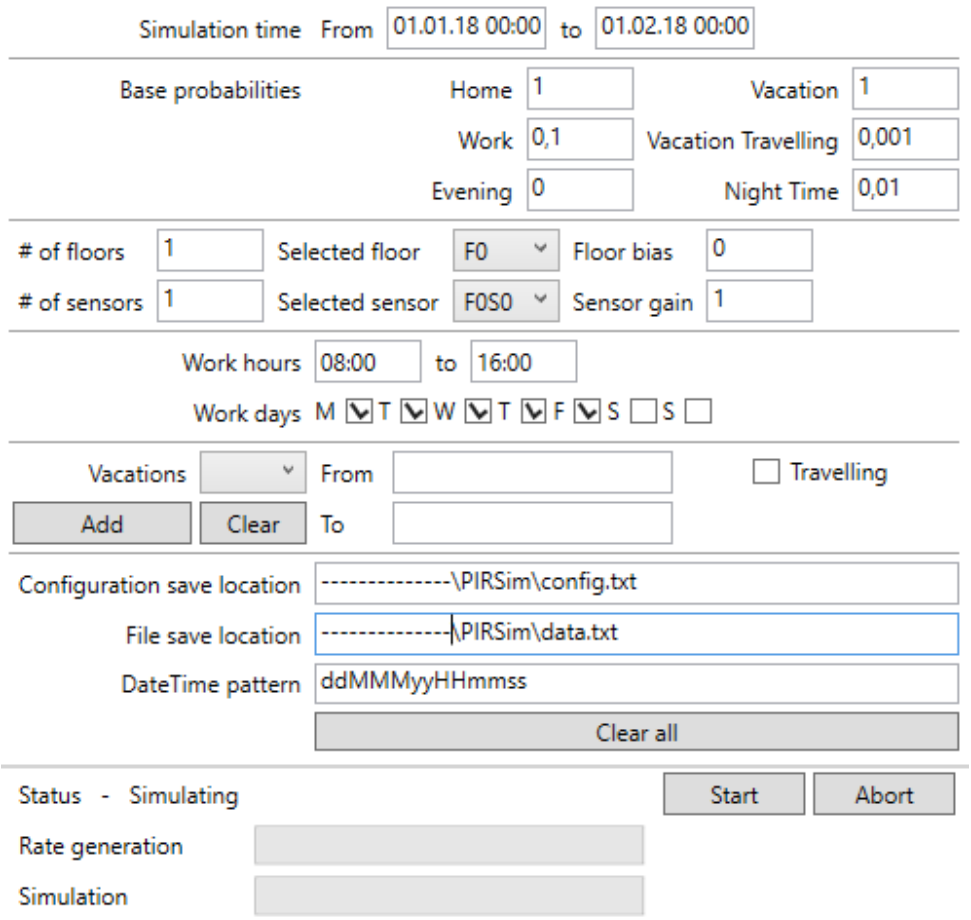


Figure 5.1: Settings for time measurement

biological data contains more clustered data than the simulated. Tuning of the margin, described in section 2.3 affects this.

The amount of data per time period is shown in the figures 4.3 and 4.4. These figures show approximately the same amount of data for the simulated and biological data. The difference being more smoothed transitions for the biological data.

5.3 PIRMod performance results

The model software shows the model as a line chart. This line chart is plotted with the candidates shown as points. The model sets the occupation probability to 0.95 at night,

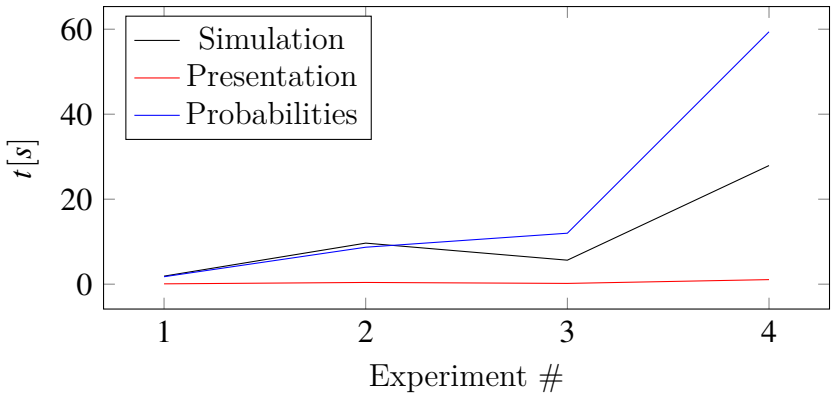


Figure 5.2: Average simulation times, as shown in table 5.1

Table 5.2: Simulated and biological data sequence

Simulated Time	**Time change**	**Biological Time**	**Time change**
15:36:39	-	13:21:33	-
15:42:21	05:42	13:22:08	00:35
15:53:47	11:26	13:22:55	00:47
16:00:13	06:26	13:23:05	00:10
16:01:25	01:12	13:23:18	00:13
16:06:10	04:45	13:23:54	00:36
16:06:48	00:38	13:24:19	00:25
16:08:57	02:09	13:24:48	00:29
16:12:02	03:05	13:24:59	00:11
16:12:56	00:54	13:25:18	00:19
16:21:34	08:38	13:25:44	00:26
16:22:45	01:11	13:26:15	00:31
16:23:00	00:15	13:26:29	00:14
16:23:10	00:10	13:27:30	01:01
16:23:33	00:23	13:28:24	00:54
16:24:34	01:01	13:29:29	01:05
16:26:53	02:19	13:36:08	06:39
16:27:20	00:27	13:39:15	03:07
16:28:13	00:53	13:39:18	00:03
16:31:16	03:03	13:40:23	01:05

unless the last event of the day is created by a sensor defined as an exit point.

Figures 5.3 and 5.4 shows two models generated from 4 and 8 weeks of data. This shows the result of using more data for training; the more data, the more detailed the model is. When introducing more data, more error is also introduced.

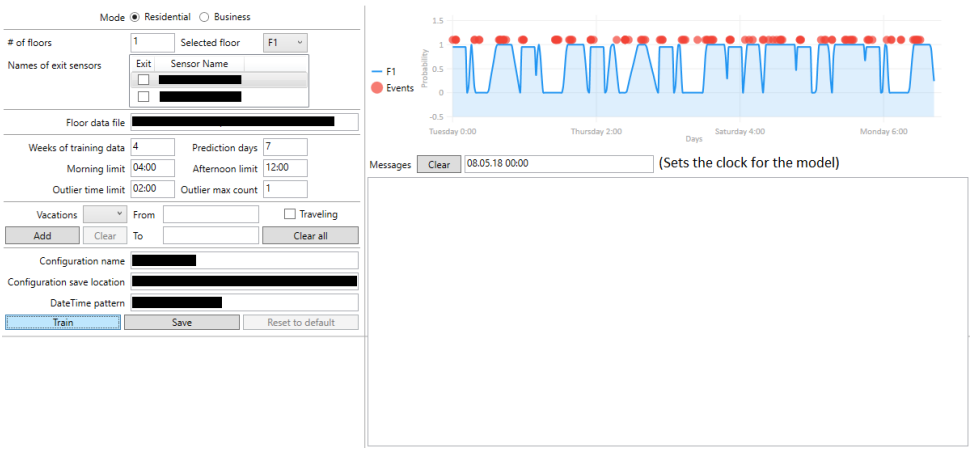


Figure 5.3: Modelling one week from 4 weeks data

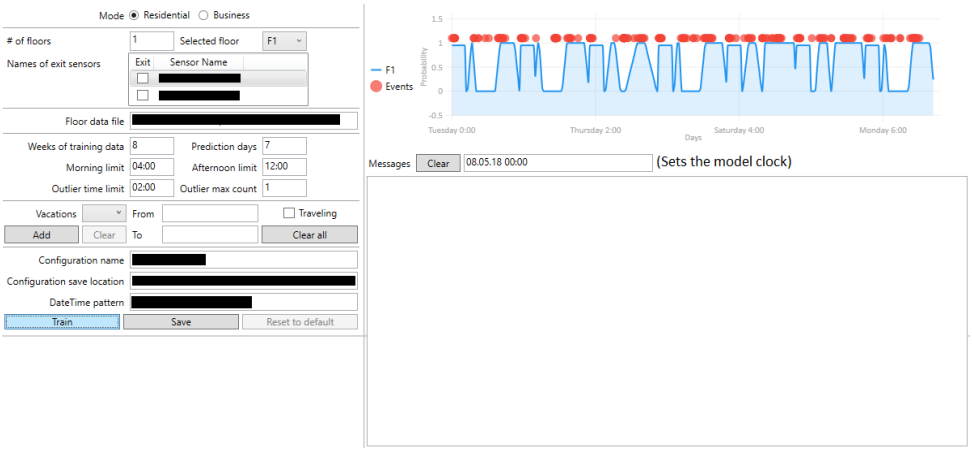


Figure 5.4: Modelling one week from 8 weeks data

From these two figures, the model can be seen to be adaptive. If the model is retrained by intervals, it will adapt to changes in behavior.

5.3.1 Model performance

Three cases have been evaluated. Each case presents a model predicting one week of occupation. These cases are listed and commented in table 5.3.

Normal model

Figure 5.5 shows the model and the movement data from the predicted week. There are no known exceptions during this week, so the measured data is representative of the

Table 5.3: Model cases

Case name	Prediction horizon	Comment
Normal model	05.02.18 - 13.02.18	A model with high quality training data. The data contains few exceptions.
Depraved model	05.07.18 - 13.07.18	This model is created from training data with some exceptions. Outliers have been removed, yielding a lack of candidates.
Friday model	30.11.18 - 07.12.18	This model is created on a Friday. There is little data to compare the model to, as there is no data from December.

occupation. The figure shows a good relation between the predicted and measured state. An exception is during Thursday and at night. The building is assumed to be occupied at night.

Figure 5.5: Normal model with candidates and measured data

Depraved model

The depraved model shows a strong mismatch between the model and the measured data, as seen in figure 5.6. The training data is not consistent enough for a complete model. As this is from summer, it is probably exceptions to the normal pattern that is the cause of the inconsistency.

Figure 5.6: Depraved model with candidates and measured data

Friday model

To demonstrate the effect of inconsistent data, the Friday model is included. It is shown in figure 5.7, where the only measured data is completely inconsistent with the model.

Figure 5.7: Friday model with candidates and measured data

Pattern consistency

As shown, the model is very susceptible to inconsistency in the behavior pattern. This is a consequence of the adaptive nature of the model.

If the behavior pattern is rigid, the model will be adequate for heating purposes, and the pattern created can be used for vacation light control. However, should the pattern change abruptly, or frequently, the model can not create the necessary parameters for the top hat function.

6 Discussion

From the previous work, several methods of determining resident location have been described. These methods often involve complicated sensory equipment or large amounts of data. This work has shown the function of a model, created from simple motion sensor data.

This data has been collected from a house in Norway, during the period of 1.1.18 to 1.12.18. In addition to this series of biologically created data, a simulator has been created.

The simulator method has been chosen for the capability of adjusting probability of event, and the margin parameter. This margin enables a higher probability of repeated events.

The model method has been selected for its simplicity and the possibility of defining time periods with slip, based upon previous events.

6.1 Model

Sections 2.3 and 2.5 describes the different methods considered. From the state space description, where the state is Boolean, Markov chain, Decision trees and Bayesian models are not suitable. Neural network requires a large amount of data, and preferably a target value, for training. As there is not target data, and the data collected is limited to a few sensors detecting motion, Neural network is not suitable for this model. Finally, a probability function is described, where the probability of occupation is approximated using a simple function. This function is then repeated for each time period.

The time periods are chosen to be the morning and afternoon. Other selections of time periods could improve the model. The function used is a smooth top hat, shown in figure 2.8. This function could be improved, by allowing for different smoothing parameters on rising and falling edge.

With the model chosen, the results shown in section 5.3.1 have been achieved. The results show an adaptive model that reflects the behavior pattern in the training data. As there are many exceptions to the behavior pattern, regarding occupation of a building, such as vacations, sick days and other non-working days, the model requires more input to achieve greater accuracy. This input is described in section 6.3.

6.2 Simulator

The simulator implemented in PIRSim performs as described in the requirements. The simulation time can be further decreased, by using a faster language like C++. As the chosen language is C#, a language designed for desktop application, multithreading and automatic memory handling, the simulation time is increased.

Simulated data is not representative of the biologically created data. This is due to the harsh limits between the time periods, and the lack of repeating events. In section 6.3, several suggestions are made to improve the simulator.

Although the simulated data is not representative, the configuration allows for adjusting and simulating many setups. The adjustment of the probabilities controls the number of events that are created, similar to effect of more residents. The adjustment of the floor bias and sensor gain also allows for control of which sensors generate more events, similar to the location difference of the physical sensors i.e sensors in trafficked areas generate more events than other sensors.

6.3 Future work

PIRSim and PIRMod have been developed to a first working version. There are many errors in the code to amend. As the software is designed to create a model and simulate motion data, most of the auxiliary functions like the color of the charts, abort function, the possibility of running multiple simulations, certain execution exceptions and so on, are not listed as improvements here. These errors are assumed to be of minor consequence to the overall goal of this work.

The points of improvement listed here are therefore oriented towards the modelling and simulation. The simulation software, PIRSim, does not produce data that is comparable to the biologically created data. The modelling software, PIRMod, generates an adaptive model that predicts the occupation of a building, one week in advance, unless the pattern of the data is not rigid enough.

6.3.1 Improving the simulation

To make a data set that is comparable to the biologically created data, the method requires more tuning. The main difference is the smoothing of the transition between time periods and the lack of repeating events.

In order to create a smooth transition, the probabilities generator should 'remember' the last generated probability, and then move towards the goal. This creates a new smoothing parameter, used to control the rate of change.

The frequency of repeating events can be increased by increasing the margin parameter. This frequency is not shown in the software, but has been determined by evaluating the difference in time between the generated events. A function to show this in PIRSim should be included, and the margin parameter made available to the user.

The configuration of sensors should be made more intuitive. A suggestion is to allow for a graphic drawing of the floor, with rooms and doors and then placing the sensors on the drawing.

6.3.2 Improving the model

The choice of model has limitations in compensating for exceptions in the behavior pattern. Other models should be implemented and tested. To allow for better testing and more models, the collection of data should include a measured occupation. Simple motion data shows only the occupation at a moment in time. If the occupation was directly measured, Neural Network or other machine learning methods could be used.

When using the top hat function, the smoothing regions should be made independent, as the time for leaving for work is very consistent, and the time when one gets out of bed is varying. This is not represented by making the smoothing regions dependent upon each other.

In addition, more sensor setups should be included. This work has been limited to one building. The system was extended early in 2019, but the amount of data collected from 2019 is very limited. There is no sensor covering the exit points in the building, and therefore, functions including exit sensors have not been tested.

The model has not been implemented in the external system. This should be done, so that the model can be tested in a real system.

7 Conclusion

This work has shown a simple model for predicting occupation from simple motion data. The model accuracy has not been measured, as there is no measured data that shows the occupation. However, the model is shown to predict the approximate occupation, when compared to the motion data.

Several modeling methods have been considered, where the Probability function method was chosen. This method uses a max of smoothed top hat functions, combined with an algorithm for creating parameters for the top hats, using the motion data.

Further, a simulator has been created. The data generated from this simulator, does not show the same properties as the biologically created data. Suggestions for improving the simulator are made.

Both of these programs have been created using the Unified Process. As described in the process, several UML documents have been created, available in the appendices. These documents describe both PIRSim and PIRMod.

If improved, the model can be used for controlling heating and burglary detection in residential and business buildings. Other usage for the model is vacation light control.

Bibliography

[1] *PIRSim - Repos.* [Online]. Available: https://automasjonservice.visualstudio.com/PIRSiMo/{_}git/PIRSim/ (visited on 11/05/2019).

[2] *PIRMod - Repos.* [Online]. Available: https://automasjonservice.visualstudio.com/PIRSiMo/{_}git/PIRMod (visited on 11/05/2019).

[3] V. Sanchez, C. Pfeiffer and N.-O. Skeie, 'A review of smart house analysis methods for assisting older people living alone', *Journal of Sensor and Actuator Networks*, vol. 6, no. 3, p. 11, 2017.

[4] Norwegian Ministry of Petroleum and Energy, *Energy use by sector - Energifakta Norge*, 2019. [Online]. Available: https://energifaktanorge.no/en/norsk-energibruk/energibruken-i-ulike-sektorer/ (visited on 21/01/2019).

[5] R. H. Dodier, G. P. Henze, D. K. Tiller and X. Guo, 'Building occupancy detection through sensor belief networks', *Energy and Buildings*, vol. 38, no. 9, pp. 1033–1043, 2006, ISSN: 03787788. DOI: 10.1016/j.enbuild.2005.12.001.

[6] A. lady, *How PIRs Work | PIR Motion Sensor | Adafruit Learning System*, 2014. [Online]. Available: https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work (visited on 29/04/2019).

[7] *Random Class (System) | Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/dotnet/api/system.random?view=netframework-4.8 (visited on 02/05/2019).

[8] M. Kulkarni, *Decision Trees for Classification: A Machine Learning Algorithm | Xoriant Blog*, 2017. [Online]. Available: https://www.xoriant.com/blog/product-engineering/decision-trees-machine-learning-algorithm.html (visited on 01/05/2019).

[9] J. Petzold, A. Pietzowski, F. Bagci, W. Trumler and T. Ungerer, 'Prediction of indoor movements using bayesian networks', in *International Symposium on Location- and Context-Awareness*, 2005, pp. 211–222.

[10] S. Särkkä, *Bayesian filtering and smoothing*. Cambridge University Press, 2013, vol. 3.

[11] N. Siddique and H. Adeli, *Computational intelligence: synergies of fuzzy logic, neural networks and evolutionary computing*. John Wiley & Sons, 2013.

[12] J. P. Boyd, 'Asymptotic fourier coefficients for a C∞ bell (smoothed-"top- hat") & the fourier extension problem', *Journal of Scientific Computing*, vol. 29, no. 1, pp. 1–24, 2006, ISSN: 08857474. DOI: 10.1007/s10915-005-9010-7. [Online]. Available: http://link.springer.com/10.1007/s10915-005-9010-7.

[13] N.-O. Skeie, 'Object-oriented Analysis, Design, and Programming using UML and C#', 2017.

[14] Msdn.microsoft.com, *Microsoft API and Reference Catalog*, 2015. [Online]. Available: https://msdn.microsoft.com/library.

[15] *C vs. C++: Comparing Two Foundational Programming Languages*. [Online]. Available: https://www.upwork.com/hiring/development/c-vs-c-plus-plus/ (visited on 06/05/2019).

[16] *Getting Started (WPF) | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/ (visited on 06/05/2019).

[17] *Introduction to Prism | Prism*. [Online]. Available: https://prismlibrary.github.io/docs/ (visited on 06/05/2019).

[18] *Model-View-ViewModel (MVVM) Explained*. [Online]. Available: https://www.wintellect.com/model-view-viewmodel-mvvm-explained/ (visited on 06/05/2019).

[19] *Unit Test Basics - Visual Studio | Microsoft Docs*. [Online]. Available: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019 (visited on 09/05/2019).

[20] *CTAN: Package pgfplotstable*. [Online]. Available: https://ctan.org/pkg/pgfplotstable?lang=en (visited on 07/05/2019).

[21] *Live Charts*. [Online]. Available: https://lvcharts.net/App/examples/v1/wpf/BasicColumn (visited on 07/05/2019).

[22] *Working days in year 2018 in Norway*. [Online]. Available: http://norway.workingdays.org/workingdays{_}holidays{_}2018.htm (visited on 09/05/2019).

Appendix A

Task description - Model development for building usage patterns based on PIR sensor data

University of
South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Model development for building usage patterns based on PIR sensor data

USN supervisor: Nils-Olav Skeie
Co-supervisor: Veralia Gabriela Sánchez
External partner: SMART Research group at USN

Task background:
Some of the research areas for the SMART Research group at USN are energy efficiency and welfare systems in SMART buildings. When a building is not in use, reducing the energy usage is the main goal. Assisting the people in the building will be the main goal of the welfare systems. A prediction of the building usage is thus important to decide how and when to use the energy and welfare systems.
The basis for the model development is the inputs from a set of simple Passive InfraRed (PIR) motion detection sensors. These PIR sensors will indicate motion of any objects in the monitored area, with a higher temperature than the environment temperature. These sensor devices are also common in alarm systems, and in future systems this information should be shared among the different systems within a SMART building environment. A sensor should be located in almost each room of the building and the sensor information will give an overview of which room is in use at which time. Based on a set of historically indications from these sensors, a data driven model has to be developed. This model will predict if the building is in use or not at a specific time and date.

Task description:

1. Make a literature survey of any equivalent systems,
2. Make a literature survey of methods that can be used when developing such a model,
3. Develop a C# simulator that will generate a set of movement events and saving these events on a log file. Develop the simulator using UP and UML.
4. Develop a C# application that will include the model, generated based on the historical movement events, for predicting the usage of the building. Develop the application using UP and UML.
5. Make a test plan and test the application according to the test plan.
6. Test the application with a real data set from a building.
7. Discuss an extension of the model for handling each floor of a building.
8. Discuss an extension of the model for burglary estimation.
9. Some suggestions for making the system autonomous, using an adaptive model.

Student category: (PT, EET or IIA students)
Student with interest for software engineering and C# programming

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

Practical arrangements:
A data set from a SMART building will be available.

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): 22-JAN-19

Student (write clearly in all capitalized letters): Jørund Martinsen

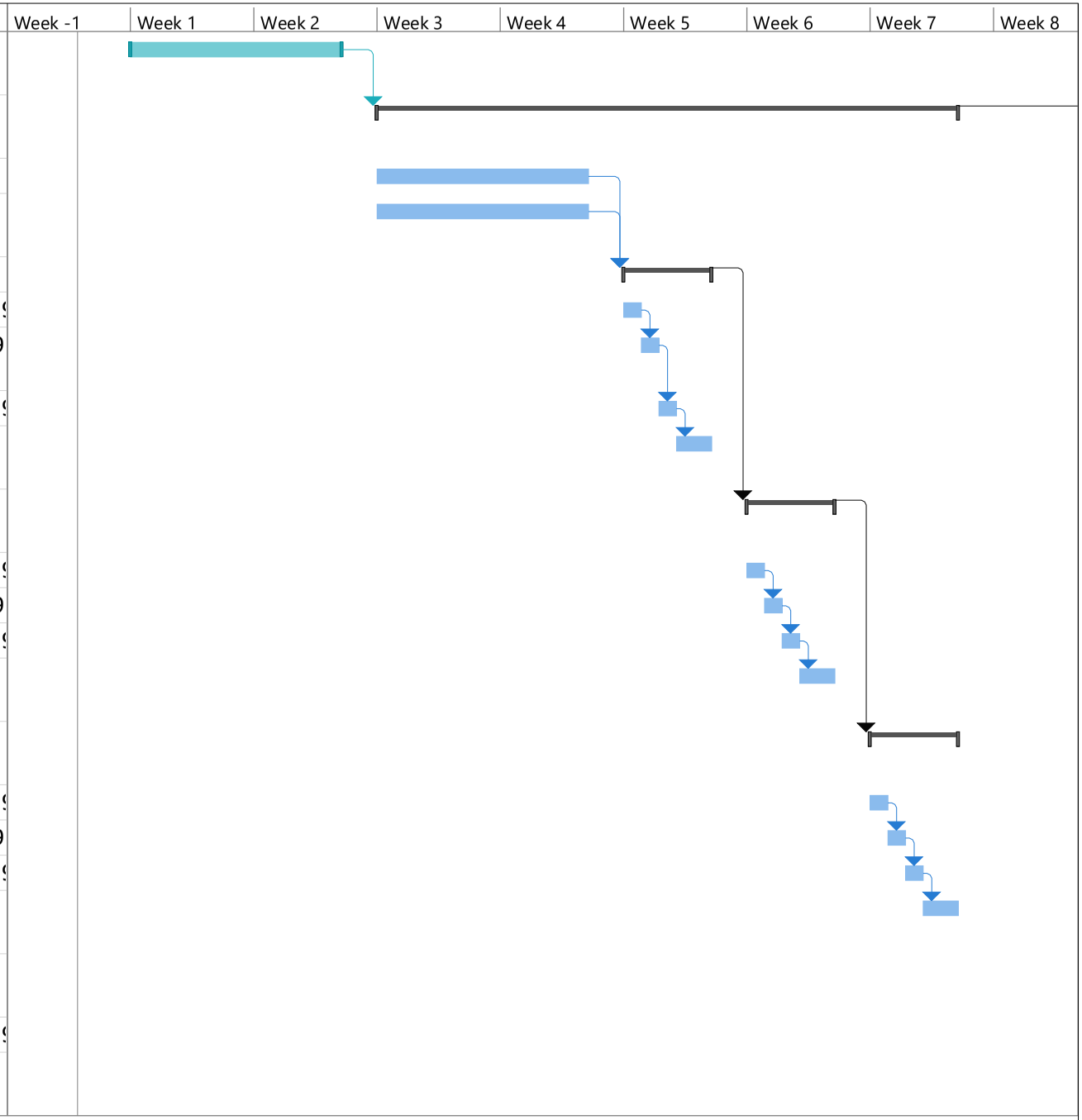
Student (date and signature): 22.1.19

Appendix B

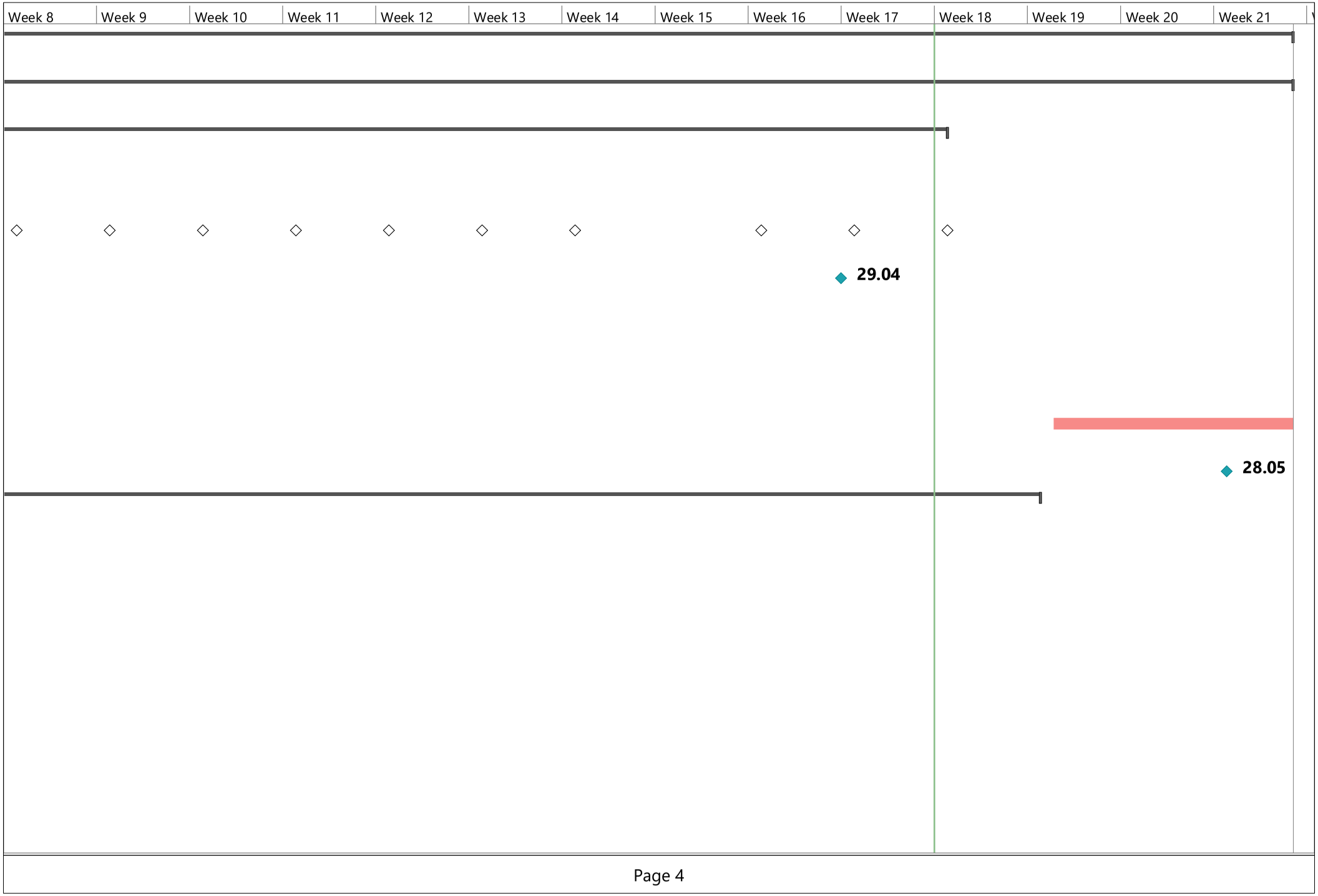
Project Schedule - GANTT

ID	WBS	Task Name	Start	Finish
1	**1**	**Master thesis Spring 19**	**Fri 04.01.19**	**Sat 01.06.19**
2	**1.1**	**Project Management**	**Fri 04.01.19**	**Sat 01.06.19**
3	**1.1.1**	**Meetings**	**Fri 04.01.19**	**Tue 07.05.19**
4	1.1.1.1	Initial meeting with supervisor	Fri 04.01.19	Fri 04.01.19
5	**1.1.1.2**	**Weekly meeting**	**Tue 15.01.19**	**Tue 07.05.19**
22	1.1.2	Set date of presentation	Mon 29.04.19	Mon 29.04.19
23	1.1.3	Project description handin	Fri 01.02.19	Fri 01.02.19
24	1.1.4	Data set given	Fri 01.02.19	Fri 15.02.19
25	1.1.5	EXPO preparation	Wed 15.05.19	Sat 01.06.19
26	1.1.6	Presentation	Tue 28.05.19	Tue 28.05.19
27	**1.2**	**Report**	**Wed 09.01.19**	**Tue 14.05.19**
32	**1.3**	**Litterature study**	**Mon 07.01.19**	**Fri 18.01.19**
33	1.3.1	Previous work	Mon 07.01.19	Fri 18.01.19
34	**1.3.2**	**Similar systems**	**Mon 14.01.19**	**Fri 18.01.19**
35	1.3.2.1	Bayesian network	Mon 14.01.19	Fri 18.01.19
36	1.3.2.2	Fuzzy logic	Mon 14.01.19	Fri 18.01.19
37	1.3.2.3	LeZi model	Mon 14.01.19	Fri 18.01.19
38	1.3.2.4	Hiden Markov model	Mon 14.01.19	Fri 18.01.19
39	1.3.2.5	Other	Mon 14.01.19	Fri 18.01.19

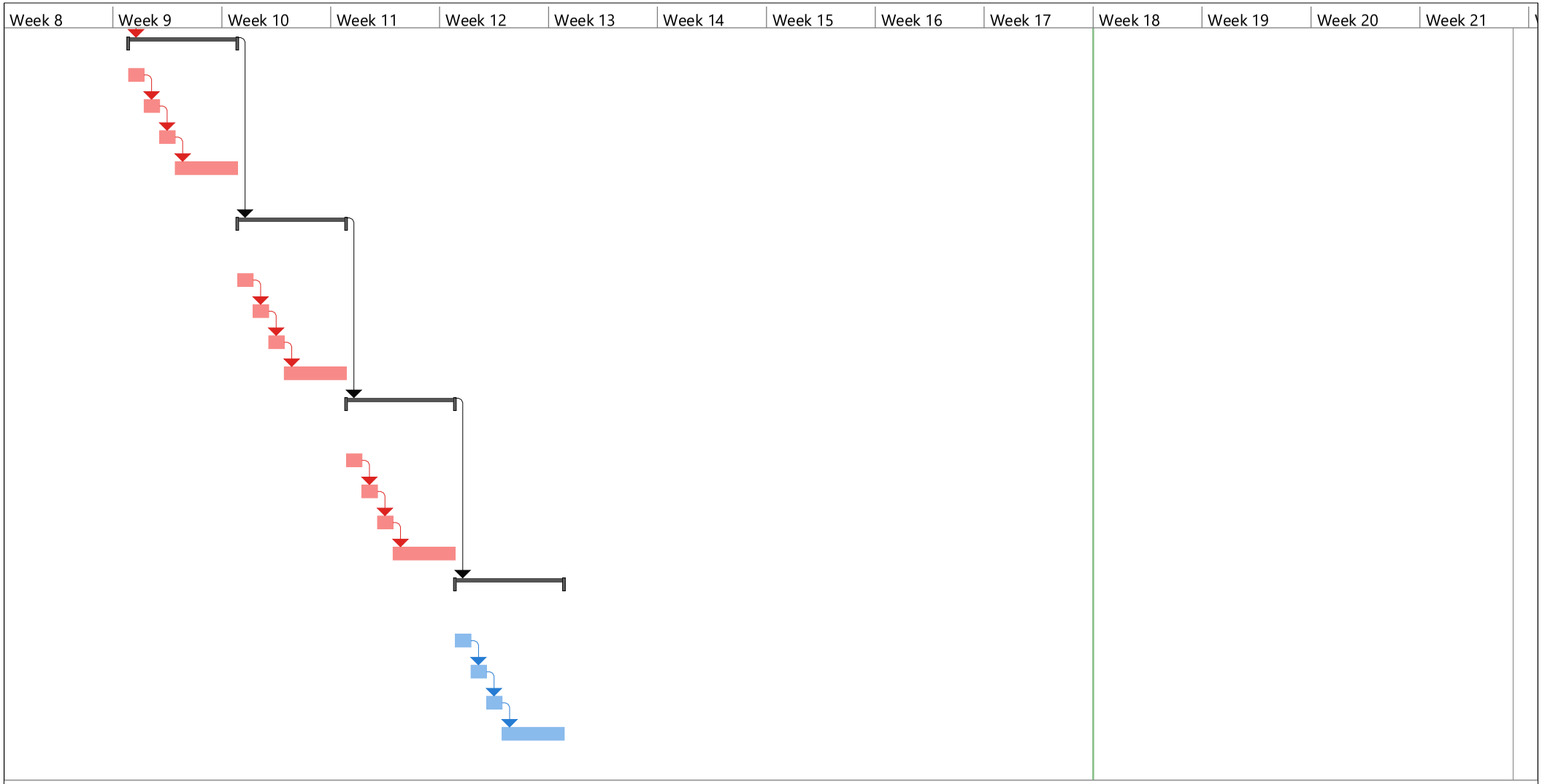
ID	WBS	Task Name	Start	Finish
40	1.3.3	Simulation methods	Mon 07.01.19	Fri 18.01.19
41	**1.4**	**Simulator**	**Mon 21.01.19**	**Fri 22.02.19**
42	1.4.1	Specification	Mon 21.01.19	Fri 01.02.19
43	1.4.2	Test specification	Mon 21.01.19	Fri 01.02.19
44	**1.4.3**	**First iteration**	**Mon 04.02.19**	**Fri 08.02.19**
45	1.4.3.1	Use case	Mon 04.02.19	Mon 04.02.19
46	1.4.3.2	Class diagram	Tue 05.02.19	Tue 05.02.19
47	1.4.3.3	Unit tests	Wed 06.02.19	Wed 06.02.19
48	1.4.3.4	Working version	Thu 07.02.19	Fri 08.02.19
49	**1.4.4**	**Second iteration**	**Mon 11.02.19**	**Fri 15.02.19**
50	1.4.4.1	Use case	Mon 11.02.19	Mon 11.02.19
51	1.4.4.2	Class diagram	Tue 12.02.19	Tue 12.02.19
52	1.4.4.3	Unit tests	Wed 13.02.19	Wed 13.02.19
53	1.4.4.4	Working version	Thu 14.02.19	Fri 15.02.19
54	**1.4.5**	**Third iteration?**	**Mon 18.02.19**	**Fri 22.02.19**
55	1.4.5.1	Use case	Mon 18.02.19	Mon 18.02.19
56	1.4.5.2	Class diagram	Tue 19.02.19	Tue 19.02.19
57	1.4.5.3	Unit tests	Wed 20.02.19	Wed 20.02.19
58	1.4.5.4	Working version	Thu 21.02.19	Fri 22.02.19
59	**1.5**	**Model**	**Mon 04.03.19**	**Mon 01.04.19**
60	1.5.1	Specification	Mon 04.03.19	Mon 04.03.19
61	1.5.2	Test specification	Mon 04.03.19	Mon 04.03.19



ID	WBS	Task Name	Start	Finish	Week -1	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8
62	**1.5.3**	**First iteration**	**Tue 05.03.19**	**Mon 11.03.1**									
63	1.5.3.1	Use case	Tue 05.03.19	Tue 05.03.19									
64	1.5.3.2	Class diagram	Wed 06.03.1	Wed 06.03.1									
65	1.5.3.3	Unit tests	Thu 07.03.19	Thu 07.03.19									
66	1.5.3.4	Working version	Fri 08.03.19	Mon 11.03.19									
67	**1.5.4**	**Second iteration**	**Tue 12.03.19**	**Mon 18.03.19**									
68	1.5.4.1	Use case	Tue 12.03.19	Tue 12.03.19									
69	1.5.4.2	Class diagram	Wed 13.03.1	Wed 13.03.1									
70	1.5.4.3	Unit tests	Thu 14.03.19	Thu 14.03.19									
71	1.5.4.4	Working versi	Fri 15.03.19	Mon 18.03.1									
72	**1.5.5**	**Third iteration**	**Tue 19.03.19**	**Mon 25.03.19**									
73	1.5.5.1	Use case	Tue 19.03.19	Tue 19.03.19									
74	1.5.5.2	Class diagram	Wed 20.03.1	Wed 20.03.1									
75	1.5.5.3	Unit tests	Thu 21.03.19	Thu 21.03.19									
76	1.5.5.4	Working versi	Fri 22.03.19	Mon 25.03.1									
77	**1.5.6**	**Fourth iteration**	**Tue 26.03.19**	**Mon 01.04.19**									
78	1.5.6.1	Use case	Tue 26.03.19	Tue 26.03.19									
79	1.5.6.2	Class diagram	Wed 27.03.1	Wed 27.03.1									
80	1.5.6.3	Unit tests	Thu 28.03.19	Thu 28.03.19									
81	1.5.6.4	Working versi	Fri 29.03.19	Mon 01.04.1									
82	1.5.6.5												


Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Week 21
29.04
28.05


Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Week 21


Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Week 21

Appendix C

Example data file from simulator - PIRSim generated data

University of
South-Eastern Norway

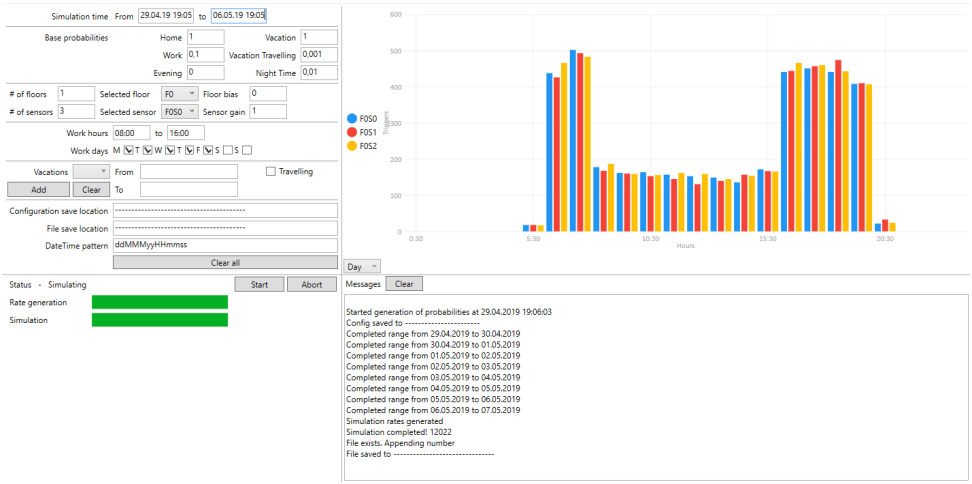
www.usn.no

PIRSim generated data

Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Porsgrunn



Simulator UI

Part of raw data from simulator. Separator = ';'

Timestamp	F0S0	F0S1	F0S2
29apr19192845	100.00	0.00	0.00
29apr19193016	100.00	0.00	0.00
29apr19193023	100.00	0.00	0.00
29apr19193017	0.00	100.00	0.00
29apr19193030	0.00	0.00	100.00
29apr19193203	0.00	100.00	0.00
29apr19193237	100.00	0.00	0.00
29apr19193256	0.00	100.00	0.00
29apr19193406	100.00	0.00	0.00
29apr19193351	0.00	0.00	100.00
29apr19193430	100.00	0.00	0.00
29apr19193508	100.00	0.00	0.00
29apr19193419	0.00	0.00	100.00
29apr19193520	100.00	0.00	0.00
29apr19193555	100.00	0.00	0.00
29apr19193509	0.00	100.00	0.00
29apr19193540	0.00	100.00	0.00
29apr19193547	0.00	100.00	0.00
29apr19193636	0.00	0.00	100.00
29apr19193846	100.00	0.00	0.00
29apr19193908	100.00	0.00	0.00
29apr19193814	0.00	100.00	0.00
29apr19193844	0.00	100.00	0.00
29apr19193814	0.00	0.00	100.00
29apr19193951	0.00	0.00	100.00
29apr19194044	0.00	0.00	100.00
29apr19194149	100.00	0.00	0.00
29apr19194140	0.00	100.00	0.00
29apr19194155	0.00	100.00	0.00
29apr19194110	0.00	0.00	100.00
29apr19194117	0.00	0.00	100.00
29apr19194217	0.00	100.00	0.00
29apr19194224	0.00	100.00	0.00
29apr19194300	0.00	100.00	0.00
29apr19194316	0.00	0.00	100.00

Appendix D

Specification for PIRSiMo

University of South-Eastern Norway

www.usn.no

PIRSiMo Specification and Requirements

Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Porsgrunn

Contents

1 Introduction

This is the specification of the PIRSiMo software (*PIR* data, *Si*mulation and *Mo*deling). The requirements are also included, to keep the information in a single document. In addition to this, two separate documents are created. These contain the test specifications and are separated pr. program, i.e. the simulator and model.
The next chapter describes the specification for PIRSiMo, and the next two contain the requirements from each program.

2 Specification

To extend the SMART house technology being developed at USN Porsgrunn, an important part is to be able to determine whether someone is currently occupying a house, or will be in the future. To generate data, a simulator should be created. To model that data, a model program must be developed.

2.1 Simulator

This simulator must generate data at a semi-random rate. This rate should be controlled by a schedule reflecting the normal usage of the building. Two types of buildings should be included; residential and business. The data generated should be stored in a CSV file of the format shown in table 2.1. Each Boolean sample is represented by the value 0.0 (False) or 100.0 (True). These values are event driven, meaning that each entry should be triggered by a rising or falling edge event. In addition, a sensor can have several rising edge events without a falling edge, within the same timestep, as shown in figure 2.1. The number of sensors should be configurable and should include which sensors cover the entry/exit points of the building.
The data generated should be displayed in a readable manner. Suggestion is to use a histogram with different bars pr. sensor. In addition, the simulator should be able to import a set of data or configurations.

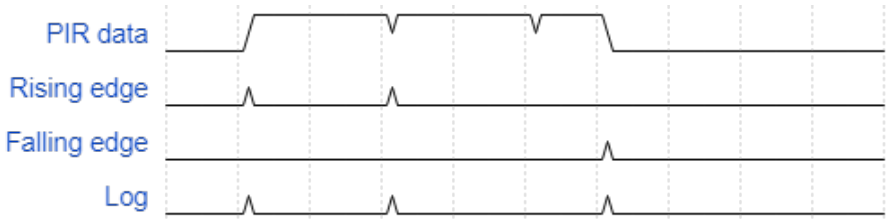


Figure 2.1: Timing diagram for PIR sensor events

Table 2.1: Data file format

Timestamp;	LivingRoom;	Kitchen;	Hall;	...;
01jan18151259;	0.0;	0.0;	100.0;	0.0;
01jan18151305;	0.0;	100.0;	0.0;	0.0;
01jan18151325;	0.0;	0.0;	100.0;	0.0;
01jan18151335;	100.0;	0.0;	0.0;	0.0;
⋮	⋮	⋮	⋮	⋮
ddMMMyyHHmmss	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0	0.0 / 100.0

2.2 Model

To adapt to each implementation, or house, the model needs to consume the data file from table 2.1. This file is generated by either using sensors or the simulator. This model should be able to receive a request containing a timestamp and reply with an indication of whether someone is occupying the building or not. To make the model usable by high-rise office buildings, each floor is to be treated as a separate building. This is also usable by residential homes.
The model should be a data driven model, utilizing probabilistic properties of the motion patterns. It should be able to predict the state, with good accuracy, 24 hours into the future. It should also be able to predict at least one week ahead, where lower accuracy is accepted.
Model logic should be separated from the graphics, to make it simpler to re-use model in other projects

3 Requirements

The requirements of the system are shown in sections 3.1 and 3.2. These reflect the information in the previous chapter and is divided in the same manner, as the two programs should function independently. The requirements are collected using the FURPS+ list as described in [1].

3.1 Simulator

3.1.1 Functional

F1 Generate Boolean PIR data for separate sensors as numeric value 0.0 for false and 100.0 for true. Rising edge events can occur multiple times before a falling edge.

F2 Use different rates to generate data. These rates are based on a schedule provided by the user

F3 Store event based data in a CSV file, using the format provided in specification. Store both rising and falling edge.

F4 Use a configurable number of sensors and persons.

F5 Simulate data for a specified time span. The simulation should be faster than realtime.

F6 _____

F7 _____

3.1.2 Usability

U1 Graphical interface should be one page only.

U2 Graphics must include a way to enter work hours and vacation time.

U3 Graphics should display the generated data in a histogram.

U4 Be able to import dataset and configuration. Check validity of files on import.

U5 Store last configuration for use later, with a configurable name.

U6 Indicate simulation progress.

U7 Document the software in a .pdf manual

U8 _____

U9 _____

Figure 3.1 shows a proposal for the GUI used with the simulator

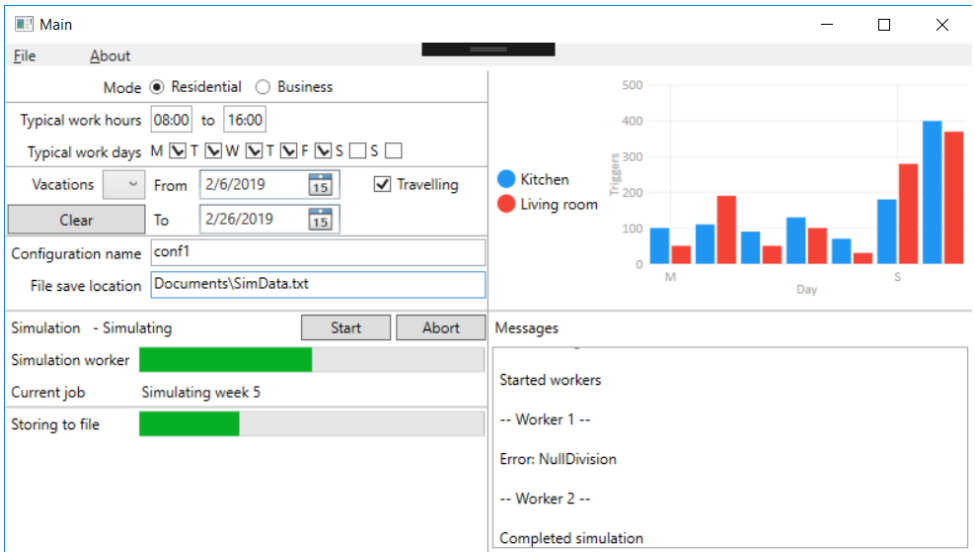


Figure 3.1: GUI proposal for simulator program

3.1.3 Reliability

R1 The generated data should not be repeatable. There should however be a pattern reflecting the calendar configuration.

R2 Configuration should include; Type of building, number of floors, number of sensors on each floor and maximum number of people.

R3 _____

R4 _____

3.1.4 Performance

P1 The simulation should not wait for an amount of time. Do simulations as fast as possible.

P2 _____

P3 _____

3.1.5 Supportability

S1 Write in C# on WPF platform.

S2 Use a Model-View-ViewModel pattern.

S3 Write software in English.

S4 Software supports Win10.

S5 _____

S6 _____

3.1.6 Other

O1 The software should run on a standard WIN 10 operating system.

O2 The software should run without administrator privileges.

O3 _____

O4 _____

3.2 Model

3.2.1 Functional

F1 Read data from file

F2 Create model from patterns in data

F3 Respond to requests, with an indication of occupation

F4 _____

F5 _____

3.2.2 Usability

U1 The user interface should be one page only.

U2 Graphics should include a way to enter work hours and planned vacations.

U3 There should be a way to set location of the datafile.

U4 Configuration should be stored to a file.

U5 Configuration should be imported from file at startup

U6 Creation of model should be possible from user interface

U7 There should be a way to test model performance

U8 Document the software in a .pdf manual

U9 _____

U10 _____

3.2.3 Reliability

R1 Indicate error messages

R2 Store errors to a log file

R3 There should be a configuration for each floor.

R4 Configuration should include type of building

R5 ... typical work time

R6 ... planned vacations

R7 ... number of sensors

R8 ... if a sensor is at an entry/exit point of the floor

R9 _____

R10 _____

3.2.4 Performance

P1 The accuracy of the model should be more than 70% for a time period of 24h

P2 The accuracy of the model can be less than 70% for a time period exceeding 24h

P3 _____

P4 _____

3.2.5 Supportability

S1 Graphic interface should be written in C# on WPF platform

S2 Graphic interface should use a Model-View-ViewModel pattern

S3 Model logic should be written in C#

S4 Model should be in a separate project to enable simple import in later projects

S5 Software should be written in English

S6 Software should support Win10

S7 _____

S8 _____

3.2.6 Other

O1 The software should run on a standard WIN 10 operating system

O2 The software should run without administrator privileges

O3 _____

O4 _____

Bibliography

[1] N.-O. Skeie, 'Object-oriented Analysis, Design, and Programming using UML and C#', 2017.

Appendix E

UML documents for PIRSim

University of
South-Eastern Norway

www.usn.no

PIRSim Documents

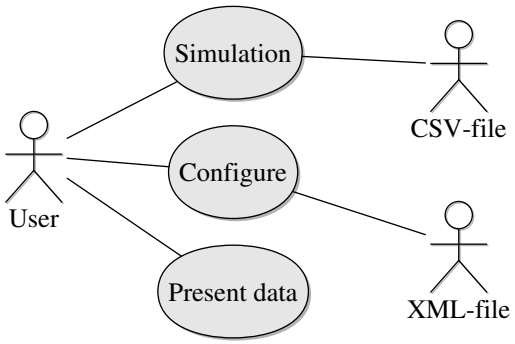
Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Porsgrunn

Contents

1 Use cases



Use case diagram

Section	**Comment**
1. Use case name	Configure
2. Scope	System
3. Level	User goal
4. Primary Actor	'User'
5. Stakeholders and interests	'XML-file'
6. Preconditions	'Simulation' not running
7. Success guarantee	Configuration saved to XML-file
8. Main Success scenario	8.1 'User' enters data 8.2 'User' selects 'start' 8.3 Configuration is checked for errors 8.4 'Simulation' is started 8.5 Configuration is saved to file

Continued on next page...

Continued from previous page...

Section	Comment
9. Extensions	9.0.a Initial values are set to empty 9.1.a Data is entered manually 1 Check data as it is entered 2 If error - Show error in data field 9.1.b Data is entered from saved file 1 Check file format 2 If error - Show error message 3 Update fields with data from file 4 Go to 9.1.a 9.2.a User selects 'save' - Skip 8.4 9.3.a If error in configuration - Show error message 9.5.a If 'Create rates using config' does not start - Show error message 9.+.a 'Cancel' is selected 1 Ask user to confirm 2 If yes, fire cancel event 3 If no, continue
10. Special requirements	None
11. Technology list	None
12. Frequency of occurence	Upon request
13. Miscellaneous	None

Section	Comment
1. Use case name	Simulation
2. Scope	System
3. Level	User goal
4. Primary Actor	'User'
5. Stakeholders and interests	'CSV-file'
6. Preconditions	Configuration is created/loaded, and simulation is started
7. Success guarantee	Data is generated and stored, using the configuration
8. Main Success scenario	8.1 Simulation started 8.2 Configuration checked 8.3 Start generating probabilities 8.4 Probabilities are generated 8.5 Start simulating values 8.6 Wait for simulation to complete 8.7 Build data set 8.8 Save data to file 8.9 Send data to 'Present data'
9. Extensions	9.2.a Error in configuration - Show error message and stop 9.4.a Probailities <= 0 - Show error message and stop 9.7.a No data - Show error message and stop 9.8.a File access denied - Show error message and stop

Continued on next page...

Continued from previous page...

Section	**Comment**
	9.+.a Error occurs - Show error message and stop 9.+.b Cancel requested - Show cancel message and stop 9.+.c Data file imported 1 Check file format - If error, Show error message and stop. 2 If no error, go to 8.9
10. Special requirements	None
11. Technology list	None
12. Frequency of occurrence	Upon request
13. Miscellaneous	None

Section	**Comment**
1. Use case name	Present data
2. Scope	System
3. Level	User goal
4. Primary Actor	'User'
5. Stakeholders and interests	'Simulation'
6. Preconditions	Data exists in memory
7. Success guarantee	Data is presented in a chart
8. Main Success scenario	8.1 Data is recieved 8.2 Select data based on config 8.3 Show selected data in chart
9. Extensions	9.1.a Configuration changes - Go to 8.2
10. Special requirements	None
11. Technology list	nuget Package
12. Frequency of occurence	On request
13. Miscellaneous	None

2 Interaction diagrams

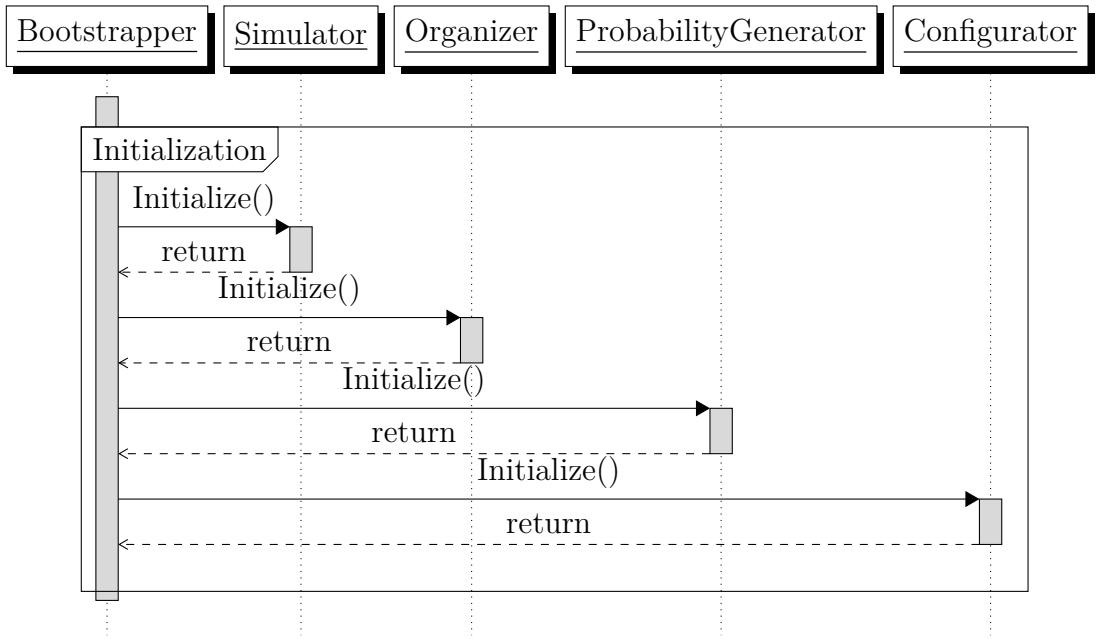


Figure 2.1: Initialization interaction diagram

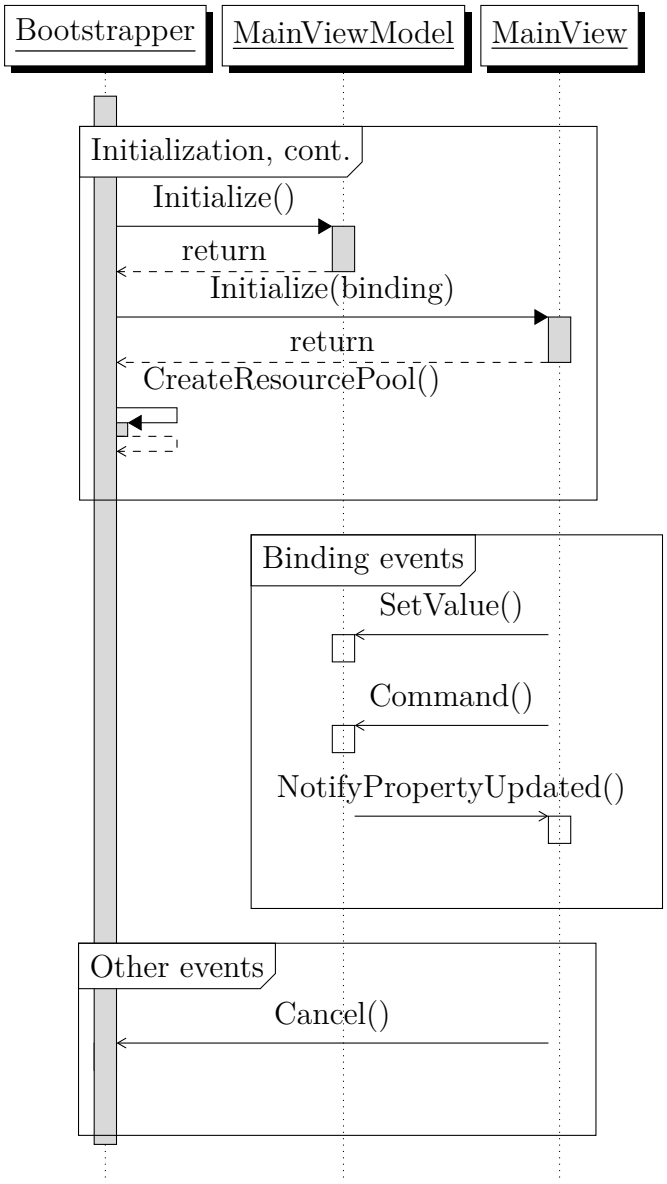


Figure 2.1: Initialization interaction diagram, continued

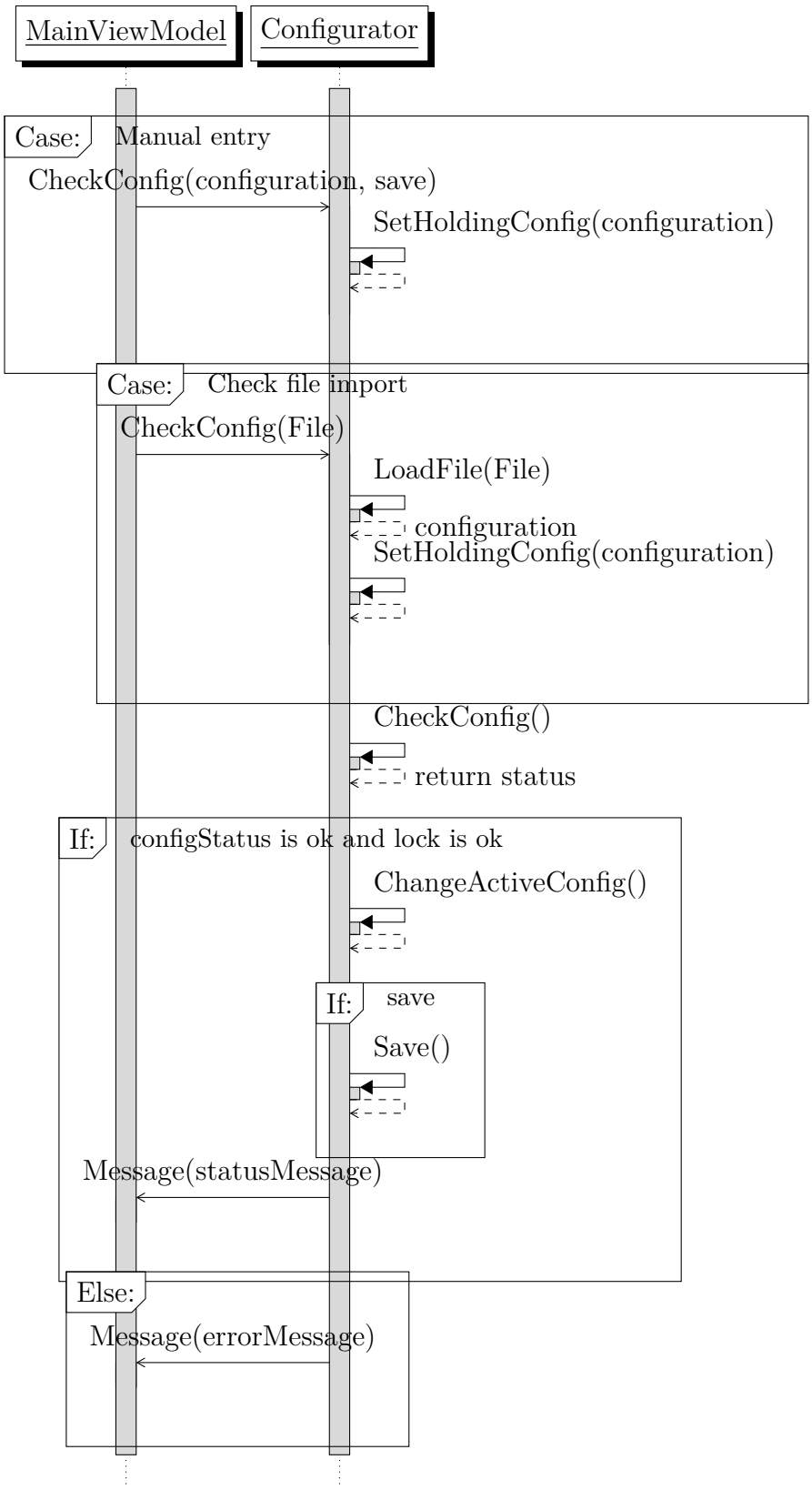


Figure 2.2: Configuration interaction diagram

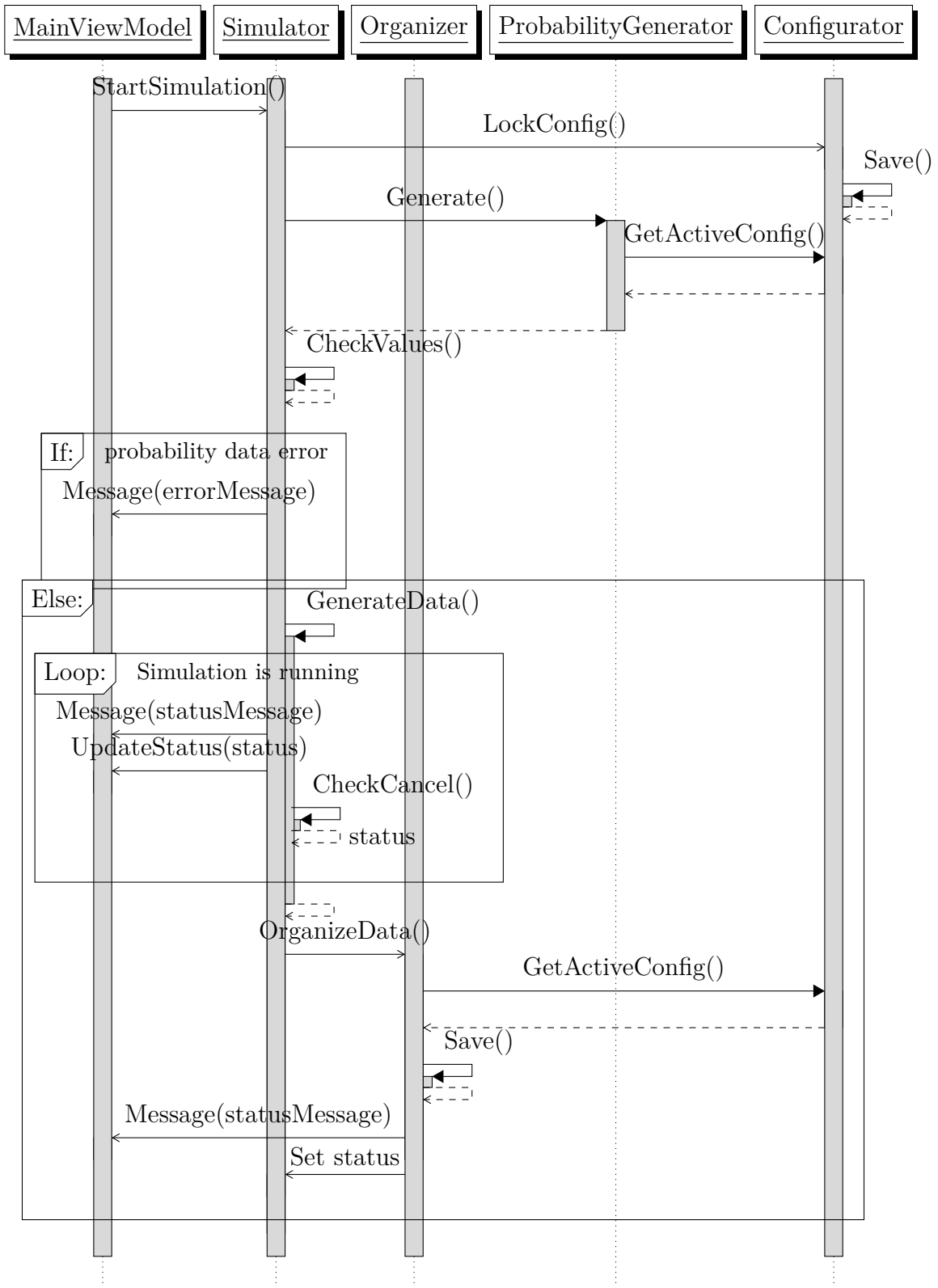


Figure 2.3: Simulation interaction diagram

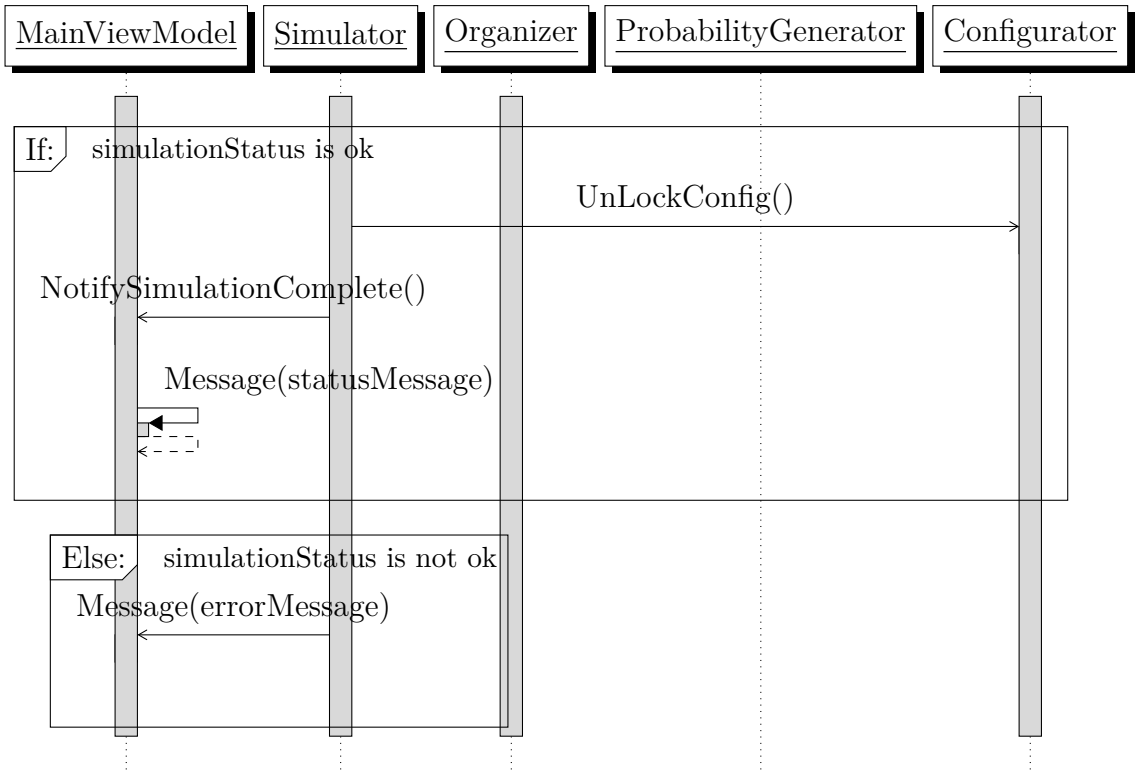


Figure 2.3: Simulation interaction diagram, continued

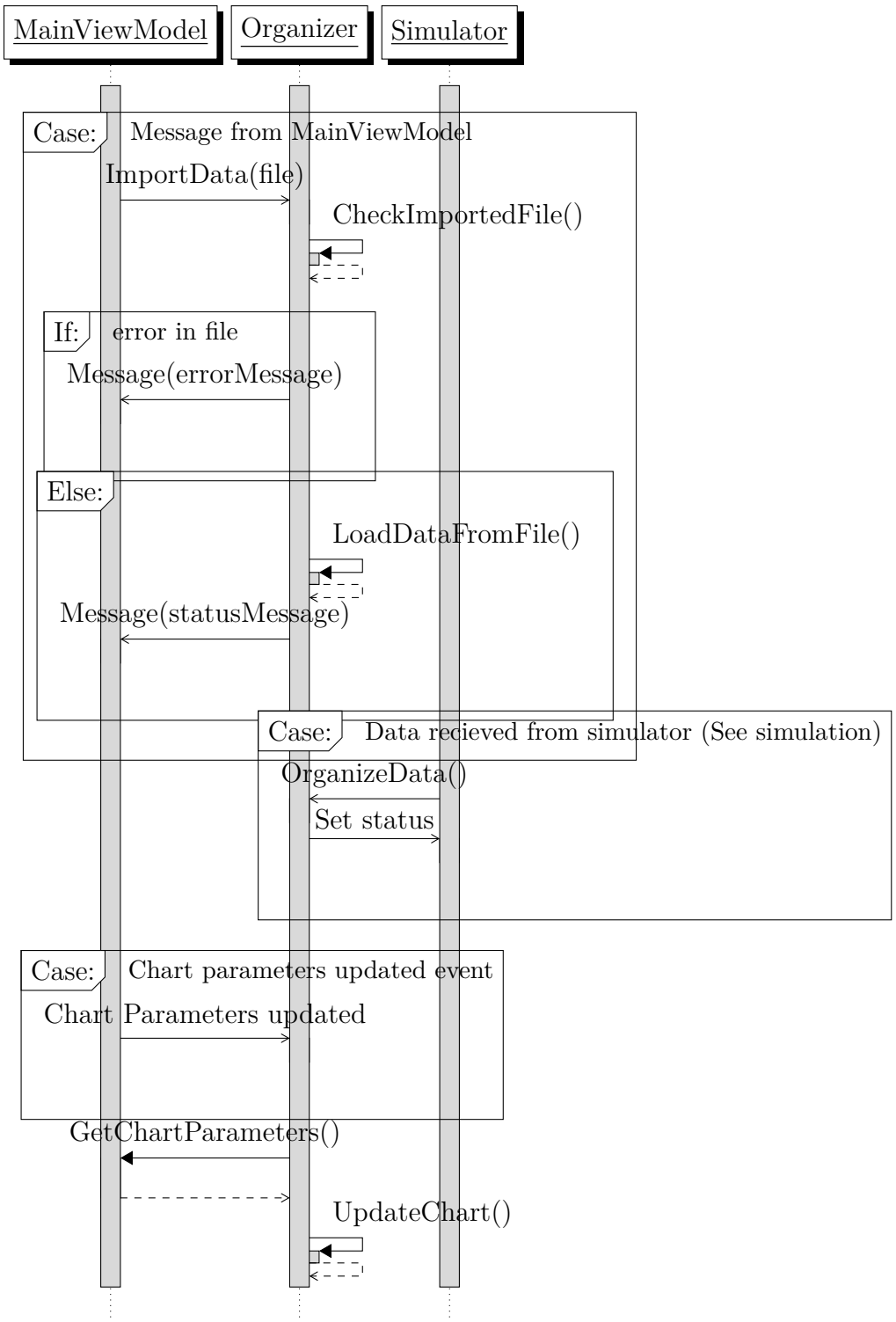
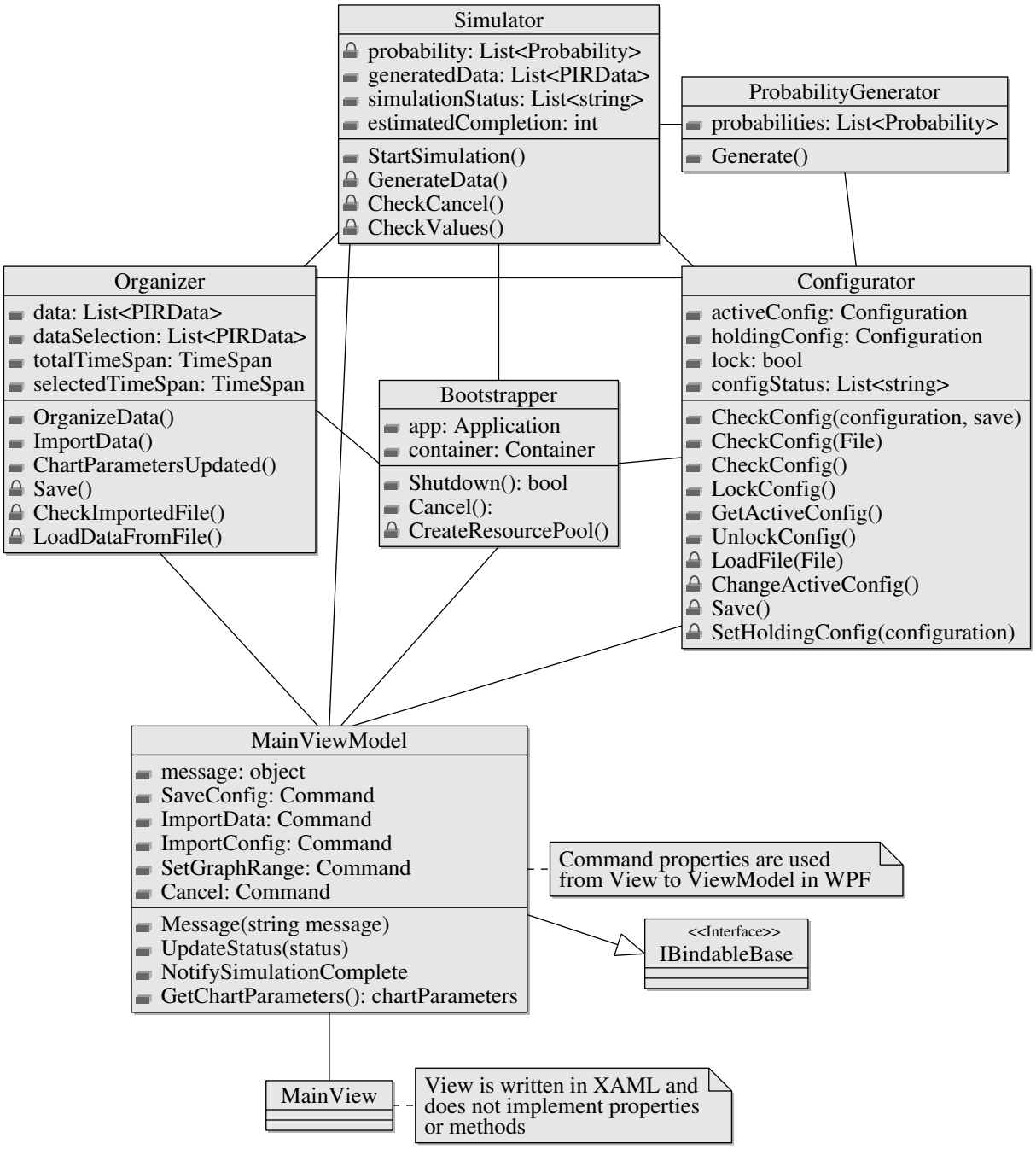
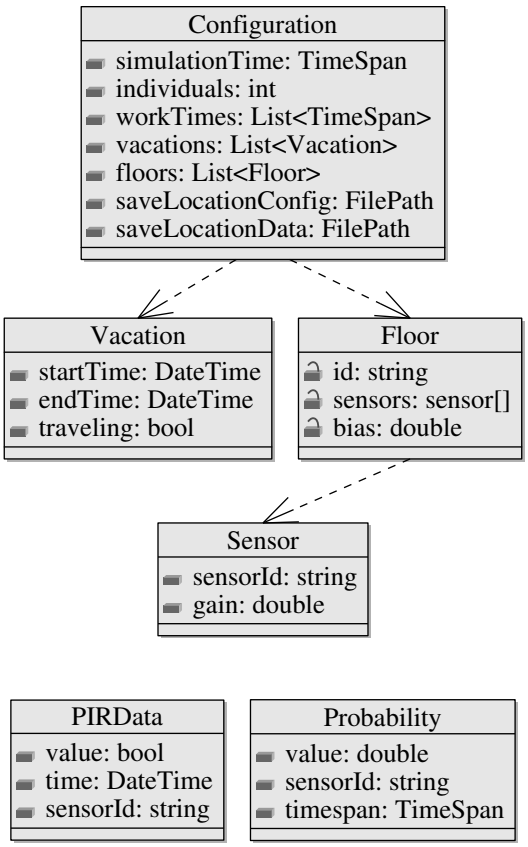


Figure 2.4: Data presentation interaction diagram

3 Class diagrams



Main class diagram



Types class diagram

Appendix F

UML documents for PIRMod

University of
South-Eastern Norway

www.usn.no

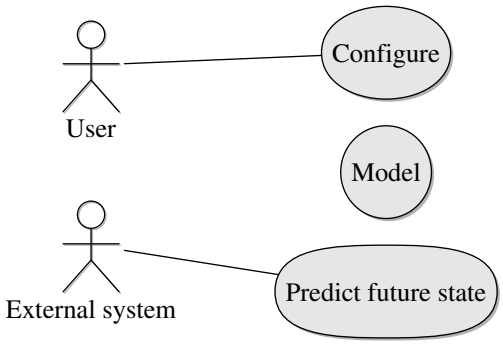
PIRMod Documents

Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

Contents

1 Use cases



Use case diagram

Section	**Comment**
1. Use case name	Model
2. Scope	System
3. Level	Sub-function
4. Primary Actor	None
5. Stakeholders and interests	Predict
6. Preconditions	Configuration completed
7. Success guarantee	Model created
8. Main Success scenario	1 Import Configuration 2 Import data 3 Find first event of each weekday 4 Find last event of each weekday, before 12:00 5 Find first event of each weekday, after 12:00 6 Find last event of each weekday 7 Add candidate events to lists 8 Remove outliers 9 Find first and last event of candidates per weekday 10 Create top hat parameters from candidates
9. Extensions	1.a If no configuration, notify user and stop 1.b If configuration error, notify user and stop 2.a If no data, notify user and stop 8.a If number of outliers exceedes limit, move to 8.9
10. Special requirements	XML reader, CSV reader
11. Technology list	None
12. Frequency of occurence	Upon request
13. Miscellaneous	

Section	**Comment**
1. Use case name	Predict future state
2. Scope	System
3. Level	Sub-function
4. Primary Actor	External system
5. Stakeholders and interests	None
6. Preconditions	Model is configured
7. Success guarantee	Result of top hat function returned
8. Main Success scenario	8.1 External system asks for $P(t)$ 8.2 Use model parameters to create top hat function 8.3 Return value of top hat function at t
9. Extensions	9.3.a If value < 0 return 0 9.3.b If value > 1 return 1
10. Special requirements	
11. Technology list	
12. Frequency of occurence	
13. Miscellaneous	

Section	**Comment**
1. Use case name	Configure
2. Scope	System
3. Level	Sub-function
4. Primary Actor	'User'
5. Stakeholders and interests	'Model'
6. Preconditions	None
7. Success guarantee	Configuration saved to XML-file
8. Main Success scenario	8.1 'User' enters data 8.2 'User' selects 'train' 8.3 Configuration is checked for errors 8.4 Configuration is saved to file
9. Extensions	9.0.a Initial values are set to empty 9.1.a Data is entered manually 1 Check data as it is entered 2 If error - Show error in data field 9.1.b Data is entered from saved file 1 Check file format 2 If error - Show error message 3 Update fields with data from file 4 Go to 9.1.a 9.3.a If error in configuration - Show error message 9.+.a 'Cancel' is selected 1 Ask user to confirm 2 If yes, fire cancel event 3 If no, continue
10. Special requirements	None
11. Technology list	None
12. Frequency of occurence	Upon request
13. Miscellaneous	None

2 Interaction diagrams

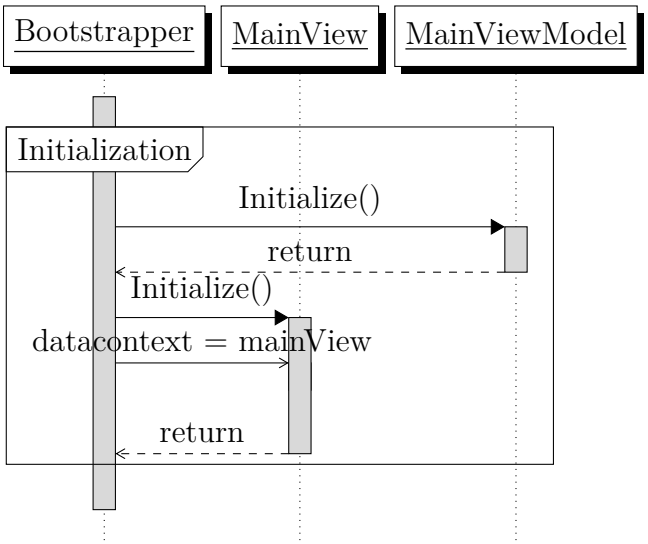


Figure 2.1: Initialization interaction diagram

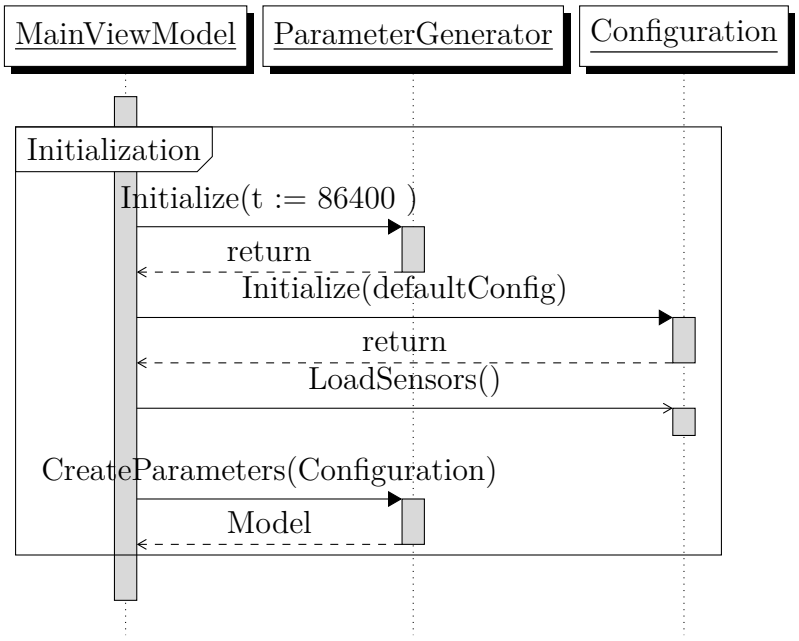


Figure 2.1: Initialization interaction diagram

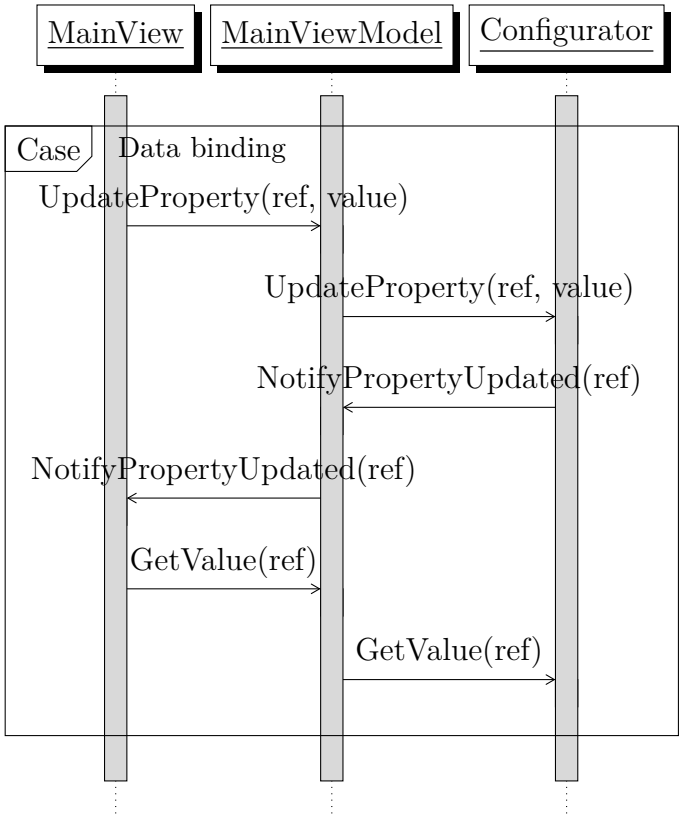


Figure 2.2: Data Binding interaction diagram

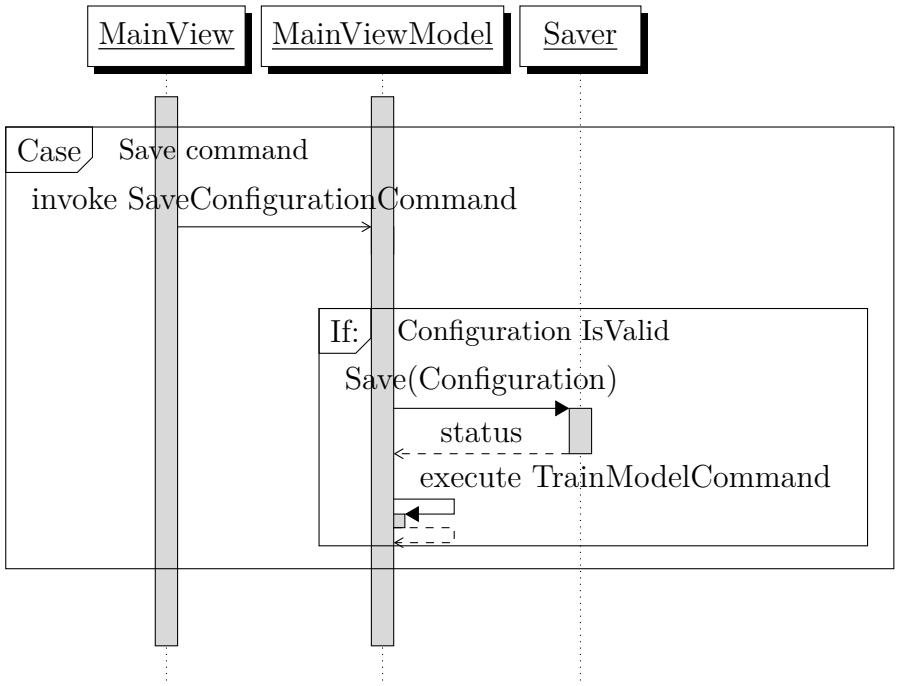


Figure 2.3: Command interaction diagram, Save

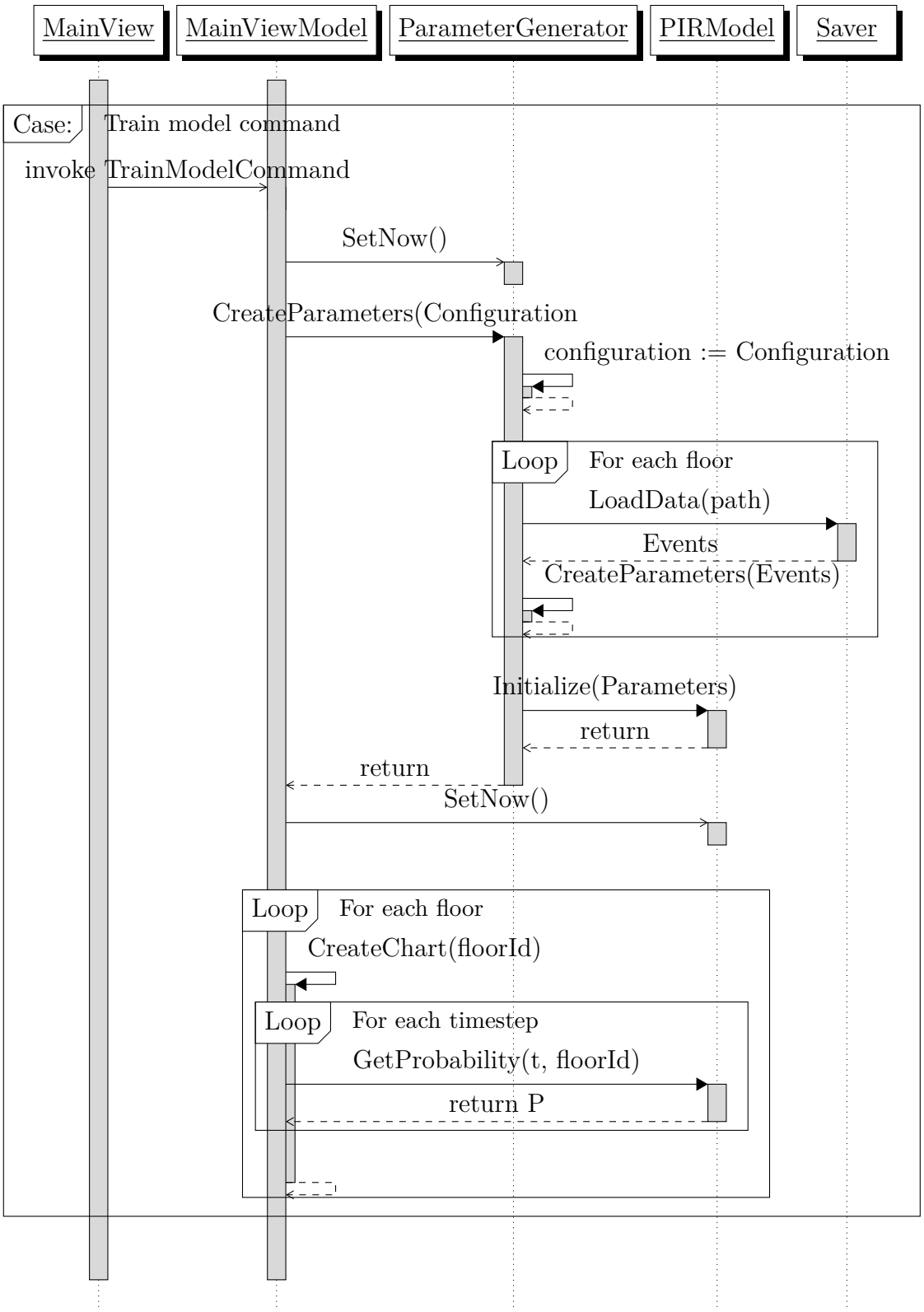
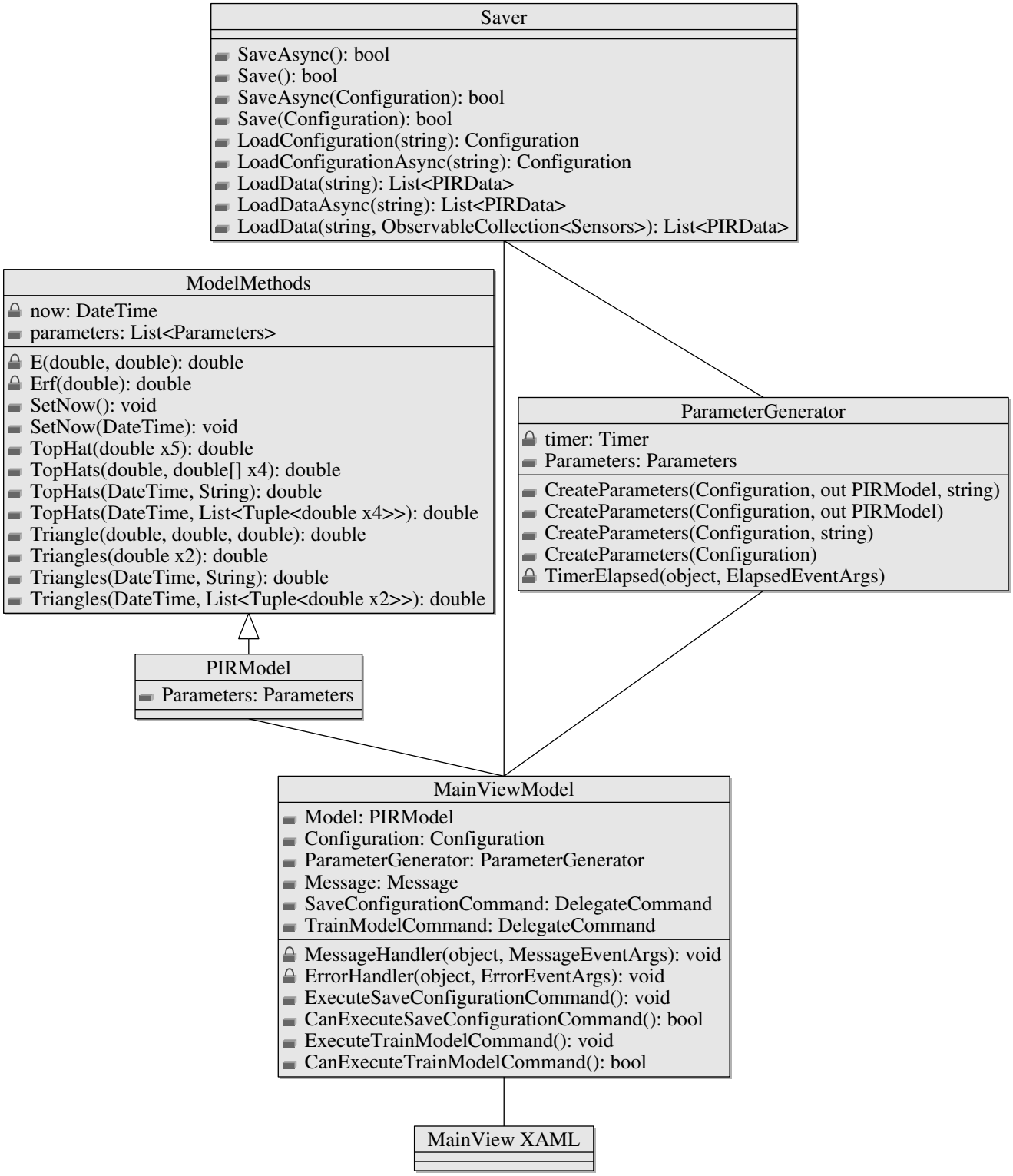
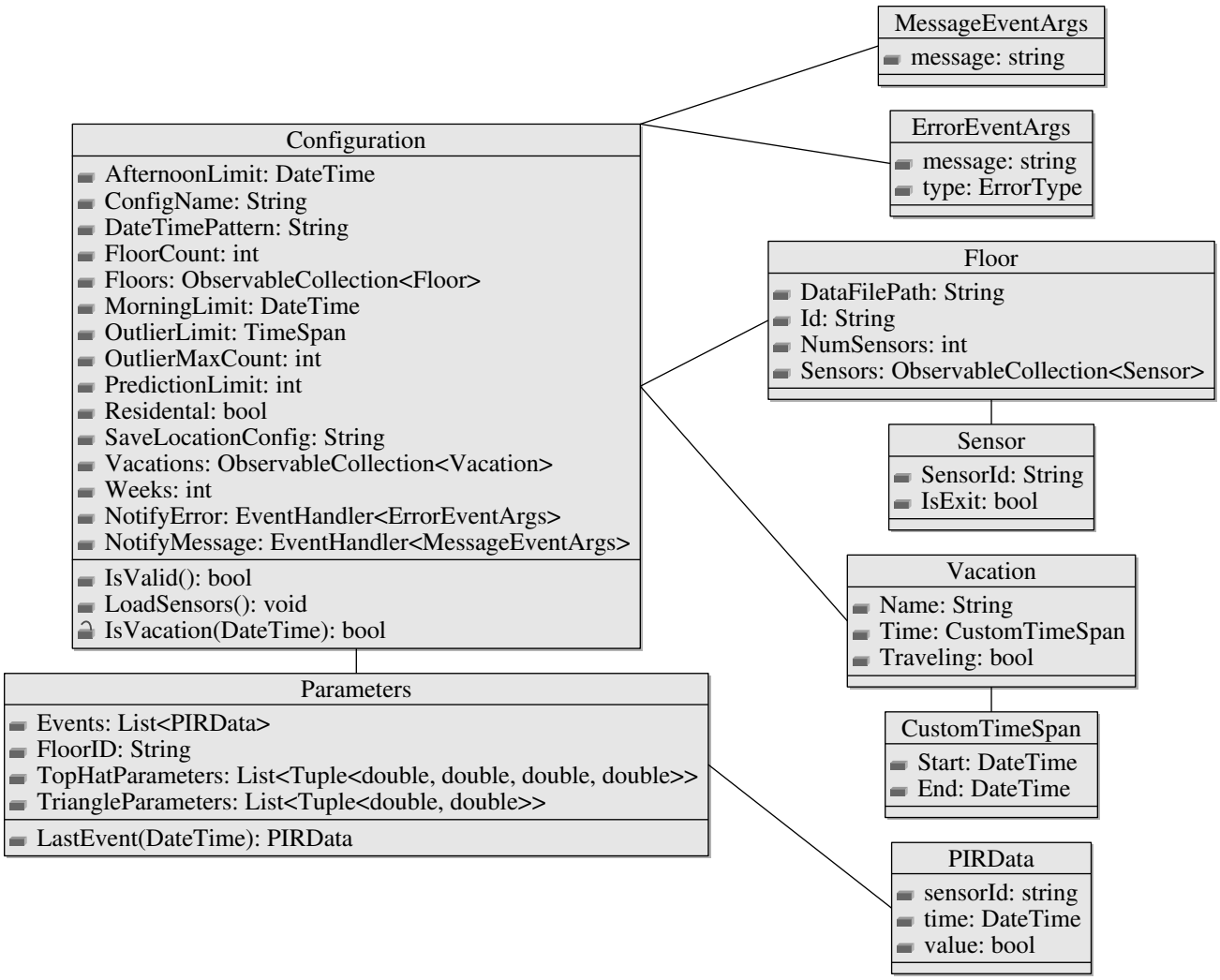


Figure 2.4: Command interaction diagram, Train model

3 Class diagrams



Main class diagram



Types class diagram

Appendix G

PIRSim test document

University of
South-Eastern Norway

www.usn.no

PIRSim Test document

Jørund Martinsen

Faculty of Technology, Natural Sciences and Maritime Sciences

Campus Porsgrunn

Contents

Introduction

This is the test document for the simulation part of the PIRSiMo software (*PIR* data, *Si*mulation and *Mo*deling). The requirements are available in a separate document. The chapters describes the test for PIRSim, following the FURPS+ procedure, as described in Skeie 2017.

Date	15.03.19
Iteration	3+
Passed all	Yes
Comment	

1 Function

Data file

(TF1) has at least two records _____

(TF2) has at least one data value of 0.0, if two sensors or more _____

(TF3) has at least one data value of 100.0 _____

(TF4) has one column for timestamp _____

(TF5) has one column for each sensor configured _____

(TF6) is accessible after simulation _____

(TF7) is in a CSV format _____

(TF8) follows the format given in specifications _____

(TF9) has exactly one header row _____

(TF10) header row has as many columns as data _____

Rates

(TF11) are higher for nonworking hours, residential _____

(TF12) are lower for nonworking hours, business _____

(TF13) are lower for weekend, residential and business

(TF14) are never less than zero

(TF15) are, overall, higher for higher number of persons

PIR data

(TF16) is generated for the configured timespan

Configuration

(TF17) accepts number of sensors greater than one

(TF18) saves in a XML format

(TF19) file is accessible to user after saving

(TF20) file can be imported

(TF21) errors are notified and discarded

(TF22) file format errors are notified and import is stopped

(TF23) name is configurable

(TF24) is saved before simulation

2 Usability

Graphical user interface

(TU1) is shown in one window _____

(TU2) is one page only _____

(TU3) includes inputs for configuration _____

a probabilities for each type _____

b number of floors _____

c number of sensors on each floor _____

d simulation timespan _____

e residential or business _____

f workdays _____

g work time _____

h vacations, when?, traveling? _____

i config name _____

j save location _____

(TU4) includes status presentation _____

(TU5) includes message box for errors, status etc _____

(TU6) includes simulation presentation, histogram _____

(TU7) histogram time selection _____

a day _____

b week _____

c month _____

d year _____

(TU8) includes method for importing data _____

(TU9) includes method for importing configuration _____

(TU10) includes method for storing configuration _____

3 Reliability

Data

(TR1) reflects configured schedule. (Pattern matches) _____

(TR2) is not repeatable _____

Configuration includes

(TR3) type of building _____

(TR4) number of floors _____

(TR5) number of sensors on each floor _____

(TR6) number of people

(TR7) configured schedule

4 Performance

(TP1) Simulation is faster than real time

(TP2) Simulation does not wait for a time

5 Supportability

(TS1) Written in C#

(TS2) Uses WPF platform

(TS3) Uses Model-View-ViewModel pattern

(TS4) Uses English for labels and content

(TS5) Runs on Windows 10

6 Other

(TS1) Runs on a normal laptop

(TS2) Does not require administrator privileges