

FMH606 Master's Thesis 2018
Industrial IT and Automation

Developing an activity simulator of a person living in a smart house

Stian Høibjerg Glittum

Course: FMH606 Master's Thesis, 2018

Title: Developing an activity simulator of a person living in a smart house

Number of pages: 84

Keywords: Smart House, Activity Simulator

Student: Stian Høibjerg Glittum

Supervisor: Nils-Olav Skeie and Veralia Gabriela Sánchez

External partner: SMART research group at USN

Availability: Open

Summary:

People tend to follow specific patterns in their daily lives. In a smart house context, this can be useful information to determine normal and abnormal behavior of smart house inhabitants. By using patterns based on the daily activities of a person living in a smart house, a system can be developed to alert family or caregivers of abnormal behavior.

The goal of this project has been to develop an activity simulator which gives a graphical representation of a person living in a smart house. The graphical representation is based on a set of sensor and activity data stored in a database. The system was developed using software development methods to analyze, design, implement and test the system. The resulting system is able to read an activity dataset, where each activity has a start time and an end time and animate the movement of the person depending on the activity being performed at that time. The system also supports insertion of new activities while the simulation is running as well as replaying previous performed activities. An import and export module were added to import new datasets to the database and export simulation data to a file.

Preface

This project has been done by a fourth semester master student from the Industrial IT and Automation program at The University College of Southeast Norway (USN). The report is mainly aimed at members of the SMART research group at USN, but also towards those with an interest in software development and/or smart house technology. The signed topic description, which defines the scope of the project, can be found in Appendix A while the work schedule can be found in Appendix B.

Software tools used in this project include Microsoft Office 365, Microsoft Visio, Microsoft Visual Studio, Microsoft SQL Management Studio, Microsoft Project, StarUML and EndNote.

It is advantageous that the reader has some knowledge about C# programming and database design to get full understanding of this report.

I would like to thank Nils-Olav Skeie and Veralia Gabriela Sánchez for invaluable help and input during the work of this project. I would also like to thank Francisco Javier Ordóñez of the Carlos III University of Madrid for the sensor and activity dataset used during this project.

Porsgrunn, 15.05.18

Stian Høibjerg Glittum

Contents

Preface	3
Contents.....	4
Nomenclature	6
1 Introduction	7
2 Existing Systems	8
2.1 HOME I/O	8
2.1.1 <i>Movement</i>	8
2.1.2 <i>Time Panel and Time Slider</i>	8
2.1.3 <i>Weather Panel</i>	8
2.1.4 <i>Devices</i>	9
2.1.5 <i>CONNECT I/O</i>	9
2.1.6 <i>Smart Home Console</i>	10
2.1.7 <i>Power Panel</i>	11
2.2 Smart House Online Simulation	11
2.2.1 <i>House Layout</i>	12
2.2.2 <i>Devices</i>	12
2.2.3 <i>Phone</i>	13
2.2.4 <i>Voice Control Module</i>	14
2.2.5 <i>User Profiles</i>	14
2.3 Comparison to the Activity Simulator	14
2.3.1 <i>HOME I/O</i>	14
2.3.2 <i>Smart House Online Simulation</i>	15
3 System Overview	16
3.1 Sensor- and Activity Data	17
3.2 Development	17
3.2.1 <i>Unified Process</i>	17
3.2.2 <i>Object-Oriented Analysis and Design</i>	18
4 Development.....	19
4.1 Iteration 1: Collecting of requirements.....	19
4.1.1 <i>Requirements</i>	19
4.1.2 <i>Use Case Diagram</i>	19
4.1.3 <i>UI Prototype</i>	20
4.1.4 <i>Database Logical Structure</i>	20
4.2 Iteration 2: Import/Export	20
4.2.1 <i>Analysis</i>	20
4.2.2 <i>Design</i>	20
4.3 Iteration 3: Simulate activity data.....	21
4.3.1 <i>Analysis</i>	21
4.3.2 <i>Design</i>	21
4.4 Iteration 4: Add sequence of activities	21
4.4.1 <i>Analysis</i>	21
4.4.2 <i>Design</i>	22
4.5 Iteration 5: Replay previous set of activities	22
4.5.1 <i>Analysis</i>	22
4.5.2 <i>Design</i>	22

4.6 Iteration 6: Configure properties and parameters	22
4.6.1 Analysis.....	22
4.6.2 Design	22
4.7 Class Diagram	23
4.7.1 Iteration 1: Collecting of requirements	23
4.7.2 Iteration 2: Import/Export	24
4.7.3 Iteration 3: Simulate Activity Data	24
4.7.4 Iteration 4: Add sequence of activities	24
4.7.5 Iteration 5: Replay previous set of activities.....	24
4.7.6 Iteration 6: Configure properties and parameters	24
5 Implementation	25
5.1 User Interface	25
5.1.1 Drawing Area	25
5.2 Model-view-viewmodel.....	25
5.3 Data Context and Data Bindings	26
5.4 Database Server.....	28
5.5 Database Connection	29
5.6 Import/Export	29
5.7 Simulate Activity Data	31
5.8 Add Sequence of Activities	35
5.9 Replay Previous Set of Activities.....	36
5.10 Configure Properties and Parameters.....	37
5.10.1 Extending with additional parameters	38
6 Testing	40
6.1 Import/Export	40
6.2 Simulate Activity Data	40
6.3 Add Sequence of Activities	40
6.4 Replay Previous Set of Activities.....	41
6.5 Configure Properties and Parameters	41
7 Discussion	42
7.1 Including Additional Persons in the Simulator	42
8 Suggestions for further work	43
8.1 Improve Animation	43
8.2 Save and Load House Drawings	43
8.3 Improve Sequence Addition	43
8.4 Improve Import and Export	43
9 Conclusion.....	44
References.....	45
Appendices.....	47

Nomenclature

.NET	-	Software framework created by Microsoft
3D	-	Three-Dimensional
ADL	-	Activities of Daily Living
CSV	-	Comma-Separated Values
FDUCD	-	Fully Dressed Use Case Document
HAR	-	Human Activity Recognition
IDE	-	Integrated Development Environment
MVVM	-	Model-View-ViewModel
OOAD	-	Object-Oriented Analysis and Design
PLC	-	Programmable Logic Controller
SQL	-	Structured Query Language
UI	-	User Interface
UP	-	Unified Process
USN	-	University of Southeast Norway
WPF	-	Windows Presentation Foundation
XAML	-	Extensible Application Markup Language

1 Introduction

The SMART research group at USN is currently working on a system to determine normal and abnormal behavior of a person living in a smart house. By monitoring the daily activities of a person over some time, the system can figure out the daily patterns of the person and determine if a given pattern is normal or abnormal. If a pattern is abnormal, the system can send an alert to the family or care workers. In order to give a graphical representation of a person living in a smart house, and potentially create a model to determine pattern statuses, a graphical simulator will be developed which is described in this report.

In order to map out possible existing simulator systems similar to the one developed in this project, a literature survey will be carried out before development of the simulator system. The literature study can be found in Chapter 2 of this report.

The system to be developed during this project will simulate the activities of a person living in a smart house over several days and the simulation will be based upon external activity datasets saved in a database. The system will also let the user be able to add new sequences of activities at given times which may result in normal or abnormal behavior. The development of the simulator will be done using OOAD for analysis and design of the system.

Chapter 2 describes the literature study done on existing simulator systems.

Chapter 3 gives an overview of the developed system as well as methods used.

Chapter 4 describes the development process which includes analysis and design of the system.

Chapter 5 describes the implementation of the software.

Chapter 6 contains an overview of the testing of the software.

Chapter 7 gives a discussion of the results of this project.

Chapter 8 gives suggestions for further work upon the system.

Chapter 9 gives the conclusion for the project.

2 Existing Systems

There exist some available systems online that simulate the workings of a smart house. Two of these systems are HOME I/O [1] developed by Real Games and Smart House Online Simulation [2] by Astea Solutions.

2.1 HOME I/O

HOME I/O is a 3D smart home automation simulator developed by Real Games. The main goal of the project is to use an interactive house with smart technology to introduce home automation concepts to the user [3]. In the program, the user is able to monitor and control different parts of the house and use these features to create smart home scenarios. The user controls a person in a first-person perspective around a three-dimensional environment.

2.1.1 Movement

The user is controlling a person in first-person perspective through the interior and exterior of a smart house using either a computer keyboard or a gamepad [4].

2.1.2 Time Panel and Time Slider

The time panel provides information about the simulation running time which includes the current simulation date and a toggle option for day light saving time [5]. The time slider represents one day from 0 to 24 hours and the user can interact with this slider to accelerate the simulation time [5]. Figure 2.1 shows a screenshot taken from the site [5] with the time panel to the left and time slider to the right.



Figure 2.1: Time panel and time slider with simulator running

2.1.3 Weather Panel

The user is able to monitor and control the weather in the simulation. The monitorable values are air temperature, relative humidity and wind speed [5]. The controllable values are minimum temperature of the day, maximum temperature of the day, humidity, wind speed and cloudiness [5]. Figure 2.2 shows a screenshot taken from the site [5] with the panel containing the values in the simulation.

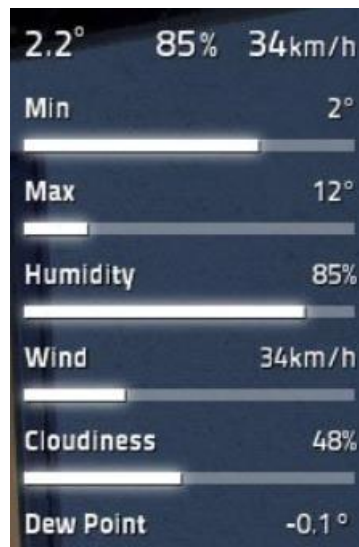


Figure 2.2: Weather Panel in the simulation

2.1.4 Devices

Different devices are located all around the house which includes lights and light switches, roller shades with switches, heaters and thermostats along with different kinds of detectors [3].

The devices may be used in three different modes which includes wired mode, wireless mode and external mode [6]. In wired mode, the device is not automated and uses a standard electrical installation. If the device is in wireless mode, the device may be controlled with the home automation console or a central wireless controller which may be used to automate the house using programming. The third mode, external mode, allows the I/O points of the devices to be accessed with CONNECT I/O, a tool also developed by Real Games, or through the HOME I/O SDK [7].

2.1.5 CONNECT I/O

CONNECT I/O [8] is a tool developed by Real Games that allows HOME I/O to be integrated with automation technologies. The interface consists of a diagram with nodes that can be linked together. Figure 2.3 shows a screenshot taken from the website [8] of the CONNECT I/O interface where nodes are linked together.

The tool can be used to connect external automation technologies to HOME I/O like PLC or microcontrollers which means that the tool can serve as a gateway between external technologies and the simulation [8]. CONNECT I/O can also be used to design a controller with the function blocks [8].

2 Existing Systems

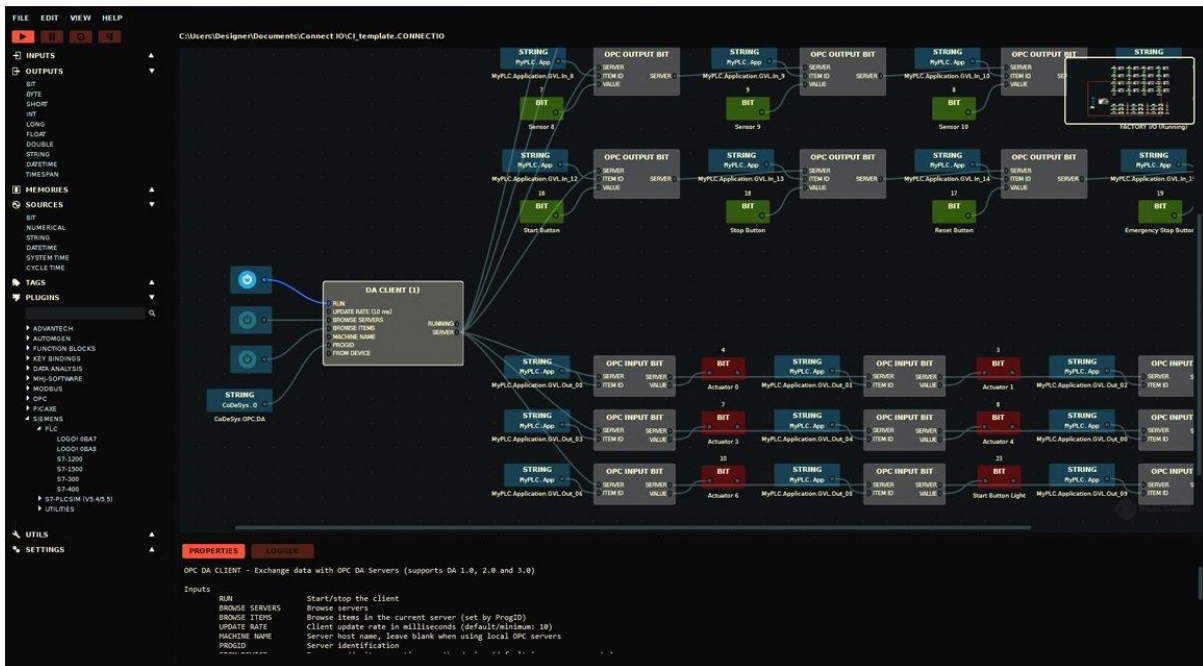


Figure 2.3: CONNECT I/O interface

2.1.6 Smart Home Console

The Smart Home Console [9] is an in-simulation tool that allows the user to create simple scenarios with the help of common home automation devices. The scenarios and devices are grouped by categories which include lighting, motorized, heating, intrusion security and domestic safety [9]. A scenario may be defined, for instance, to have the lights in a room turn on or off whenever a person enters or leaves a room. Figure 2.4 shows a screenshot taken from the website [9] of the Smart Home Console while running the simulation.

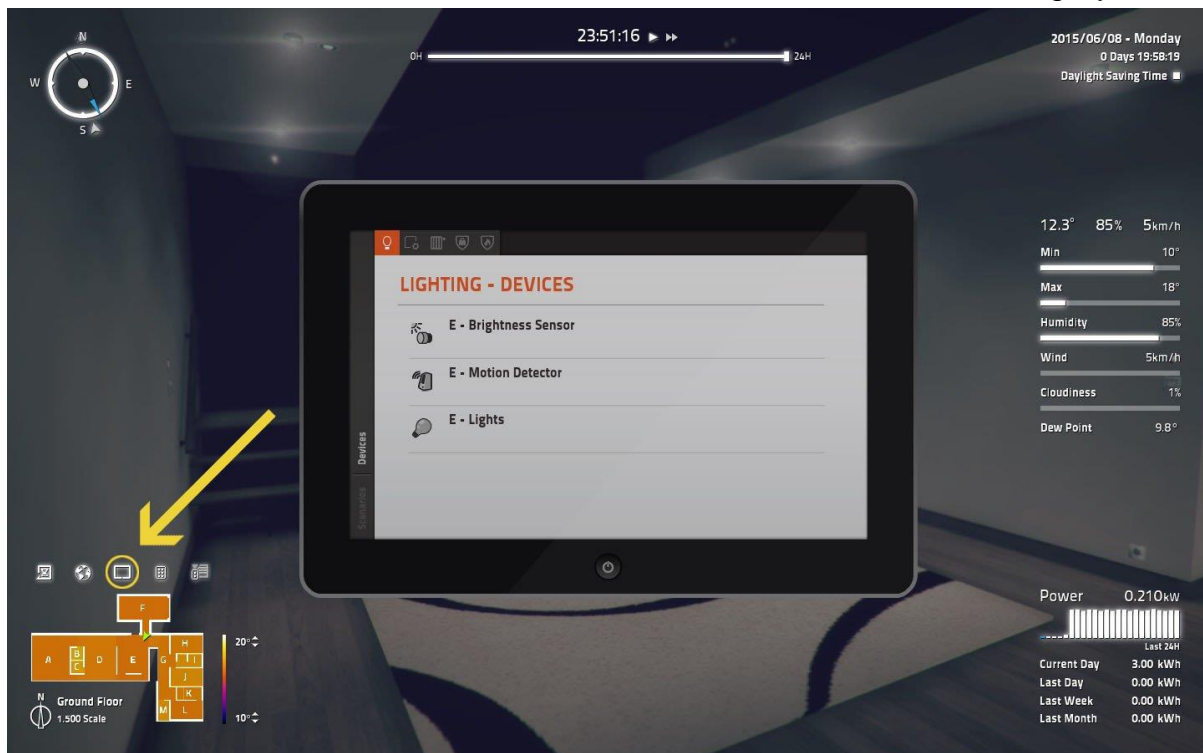


Figure 2.4: Smart Home Console while running the simulation

2.1.7 Power Panel

The power panel [5] signifies how much power is being consumed by the different devices in the smart house. It shows the instant energy consumption as well as energy/consumption of the current day, previous day, previous week and previous month. Figure 2.5 shows a screenshot taken from the website [5] of the power panel within the simulator.



Figure 2.5: Power Panel within the simulation

2.2 Smart House Online Simulation

The Smart House Online Simulation is a simulation developed by Astea Solutions as part of their Cloud4all project. Its goal is to research the capability of “auto-personalization from profile” in the home environment [10]. This means that the smart house can be customized depending on the profile logged in to the system.

2.2.1 House Layout

The house can be seen from a bird's eye view perspective with five different rooms, whereas three of them are interactable: laundry room, living room and kitchen. Clicking on one of the rooms, zooms in on the room and allows the user to interact with devices in that room. Figure 2.6 shows a screenshot from the website [2] of the house layout.



Figure 2.6: Smart house layout

2.2.2 Devices

If a certain room is selected, the simulator will zoom into that room and give the user options to interact with the devices in that room. When the user clicks on a device, a new interface appears depending on the device selected.

In the kitchen, the oven is a device that can be interacted with. The interface [2] that appears when the oven is clicked on be seen in Figure 2.7. From the figure, it can be seen that the user has the option to set a temperature for a specific amount of time along with other heating options.

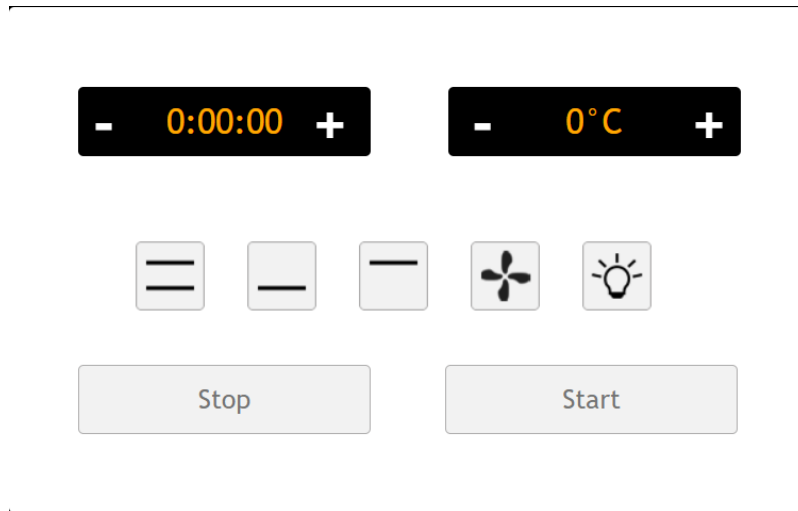


Figure 2.7: Interface for the oven

2.2.3 Phone

The simulator features an in-simulator tool called the phone which allows the user to control the different devices in the house from this tool. Figure 2.8 shows the interface from the website [2] for the phone with the different devices in the house.

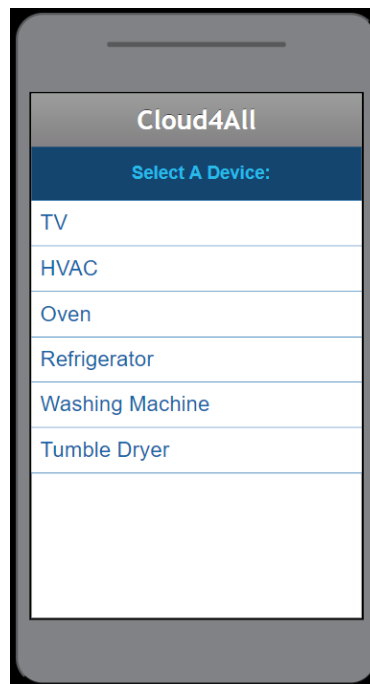


Figure 2.8: Phone interface

2.2.4 Voice Control Module

The simulator includes a voice control module which allows the user to go to a room or select a device with voice commands. To go to a room, the user can say “go to” followed by the room [2]. To select a device, the user can say “use” followed by the device [2].

2.2.5 User Profiles

A user login system is included in the simulator. This allows for users with specific disabilities to have the different interfaces tailored for them. The simulator includes a set of sample users [2] to select from as can be seen in Figure 2.9. Each of the sample users have some kind of disability and may speak different languages. The program can change the language of the interfaces and make other adjustments to make it easier for the person to live in the house.

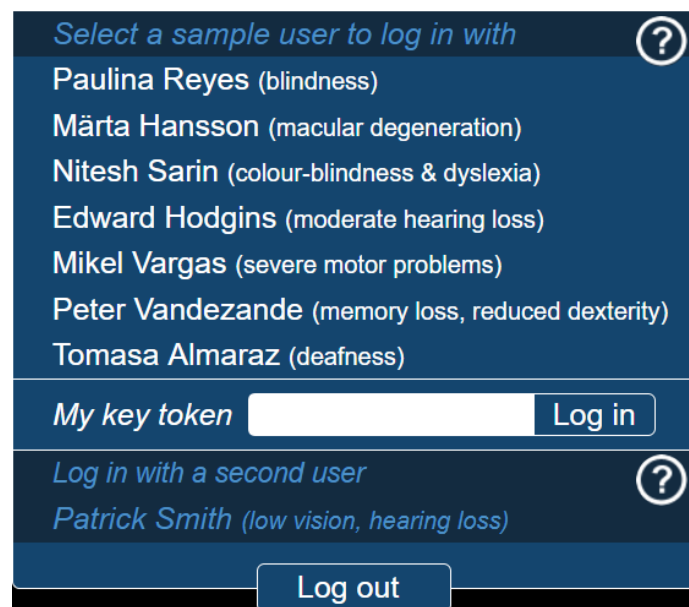


Figure 2.9: User login interface

2.3 Comparison to the Activity Simulator

The two systems described in this chapter have some similarities with the activity simulator system developed in this project. However, they are developed for different reasons. HOME I/O was designed to give the user the possibility to learn about home automation principles, while Smart House Online Simulation was developed to investigate the possibility of “auto-personalization from profile”.

2.3.1 HOME I/O

Listed below are the functional requirements for the activity simulator developed in this master’s project with comments on how HOME I/O compares to these requirements.

- Import/Export – As HOME I/O does not use activity data for the simulation, there is nothing to import or export.

2 Existing Systems

- Simulate Activity Data – HOME I/O does not simulate activity data but is still a simulation of a smart house. As opposed to the Activity Simulator, where a person is simulated to perform activities depending on the dataset, the user gets to control the person directly to perform activities at any given time. Like the Activity Simulator, the time moves forward in real time.
- Add sequence of activities – As the simulation is not based on any dataset, there is no place for any activities to be inserted. The user can instead choose where they want to go and what to do at all times.
- Replay previous set of activities – There is currently no way to replay the simulation, as the time only moves forward.
- Configure properties and parameters – There are several properties and parameters that can be configured within the simulation. Some of these include weather and devices. The user can make scenarios where devices are controlled by a certain action, for instance, turning on the lights when a person enters the room.

2.3.2 Smart House Online Simulation

Listed below are the functional requirements for the activity simulator developed in this master's project with comments on how Smart House Online Simulation compares to these requirements.

- Import/Export – The simulation does not depend on datasets, so there is nothing to import or export.
- Simulate Activity Data – The simulation comes in the form of the user interacting with rooms and devices. Time is not included in this simulation.
- Add sequence of activities – No dataset to insert activities. Devices are interacted with directly by the user, for instance, turning on the TV.
- Replay previous set of activities – As time is not a concept in this simulation, there is nothing to replay.
- Configure properties and parameters – The user is able to configure several properties of the different devices and the user experience may be different depending on the user logged in.

3 System Overview

The following chapter gives an overview of the system developed as well as the methods used throughout the development cycle.

The system consists of a C# application developed in Microsoft Visual Studio as well as a database implemented in Microsoft SQL Server. The purpose of the system is to simulate the activity of a person living in a smart house as part of research on human pattern recognition. The activity data is downloaded externally from a data storage online [11] and is based on sensor data of a person living in a smart house with the help of Human Activity Recognition (HAR). The simulator gets activity data from the database and gives a graphical, real-time representation of the data. The data used is converted to csv format with semicolons as the separator and this is also the format used for export. Figure 3.1 shows an overview of the system created in this project. The simulator system can possibly, in the future, be used to create a model to recognize if a person's behavior is normal or not and give a notification if the person behaves abnormally.

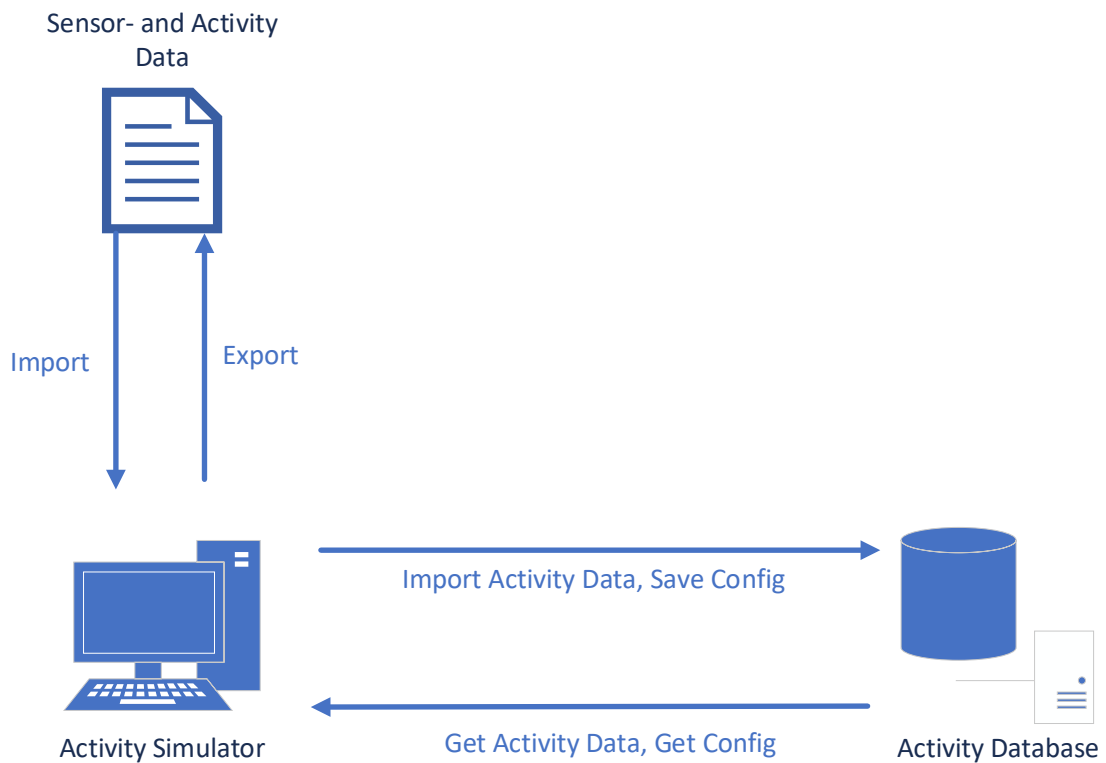


Figure 3.1: Overview of the system. The simulator imports the dataset from a file and saves it in the database. The simulator can then get the data from the database in order to simulate it.

3.1 Sensor- and Activity Data

The sensors- and activity data [11], which the simulation is based upon, is recorded by Francisco Javier Ordóñez from the Charles III University of Madrid. The dataset consists of two text files, where one is recorded sensor data and the other is activity data. The activity data is acquired by using Human Activity Recognition (HAR) on the sensor data to get Activities of Daily Living (ADLs).

3.2 Development

The system is developed using a modified version of the Unified Process framework, and through the use of Object-Oriented Analysis and Design.

3.2.1 Unified Process

Unified Process is a use-case-driven and iterative development process framework commonly used in development of software systems [12]. UP has four main phases, which includes Inception, Elaboration, Construction and Transition.

- Inception is the startup phase and includes business modeling and project management decisions [12].
- Elaboration focuses on collecting the requirements, creating use case diagram and designing the core architecture of the system [12].
- Construction is the main phase where singular use cases are selected, analyzed, designed, implemented and tested [12].
- The last phase, Transition, focuses on releasing a working version of the system [12].

The system made in this project is developed using a modified version of UP and the main phases of UP are followed loosely.

- The Inception phase, or the startup of the project, included acquiring the task description from the supervisor and creating a work schedule in order to plan the project progress.
- The Elaboration phase describes Iteration 1 in the development process which includes the collecting of the requirements and designing the use case diagram.
- The Construction phase describes Iteration 2-6 where a singular use case is selected for each iteration and is analyzed, designed, implemented and tested.
- The Transition phase describes the delivery of the system to the supervisors as well as the documentation which comes in the form of this report and its appendices.

3.2.2 Object-Oriented Analysis and Design

Software development and programming languages today are often based on classes and objects. OOAD is the process of transforming any collected requirements into objects through the use of analysis and design [12].

For the system described in this report, OOAD is used by first collecting the requirements from the task description. The requirements are then analyzed to investigate and understand the problems through the use of use cases, domain model etc. The design phase builds on the analysis by defining objects, and how they interact, as well as the classes. This is done by making interaction diagrams and the overall class diagram.

4 Development

The following chapter describes the development process which includes analysis and design using principles from OOAD. The first iteration describes the collecting of the requirements and creation of the use cases. It also describes creation of the UI prototype and the logical architecture of the database. Iterations 2 – 6 each selects a use case for analysis and design where a FDUCD is created during the analysis phase and an interaction diagram is created during the design phase. Finally, a class diagram is created from the interaction diagrams. The FDUCDs are created using Microsoft Word using a template from lecture notes [12] while the interaction diagrams and class diagram are created using StarUML [13].

4.1 Iteration 1: Collecting of requirements

Iteration 1 starts with the collection of the requirements from the task description and making a Use Case diagram based on these requirements. Furthermore, a UI prototype and logical architecture of the database is made.

4.1.1 Requirements

Functional:

Import/Export; The application should be able to import data to the database and also export simulation data to a file.

Simulate activity data; The application should transform the activity data from the database into a graphical representation of the activities in real-time.

Add sequence of activities; A module to add new sequences of activities should be available in the application.

Replay previous set of activities; The user should be able to create replays of previous activities and play the replays in the application.

Configure properties and parameters; There should be a configuration section in the application to change properties and parameters.

Usability:

Windows-based graphical interface with view of the building and person moving inside.

Performance:

Real-time graphical representation.

4.1.2 Use Case Diagram

From the functional requirements of the requirements document found in 4.1.1, the Use Case Diagram is created. The diagram can be found in Figure 4.1.

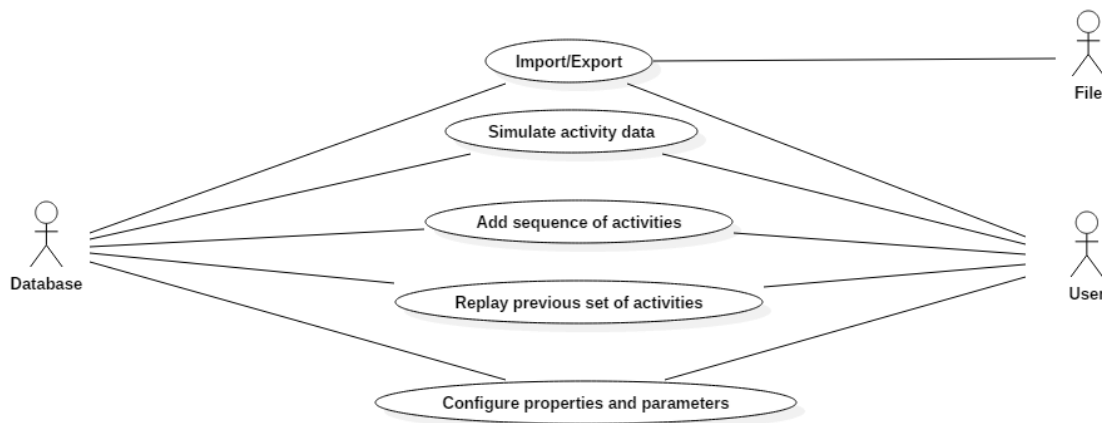


Figure 4.1: Use Case Diagram created from the functional requirements.

4.1.3 UI Prototype

The UI prototype is designed in Microsoft Visual Studio 2017 with Windows Presentation Foundation. The design is done partly by dragging and dropping items from the WPF toolbox and writing the XAML code. The UI is further worked upon later in the development cycle.

4.1.4 Database Logical Structure

The logical structure of the database is designed using Microsoft Visio with Crow's Foot Database Notation. Appendix C includes the logical structure for the database.

4.2 Iteration 2: Import/Export

Iteration 2 selects the “Import/Export” use case for analysis and design.

4.2.1 Analysis

“Import/Export” describes two separate functions that are combined into one use case. Import describes the function of importing a text file into the database to have a new set of activity data available. Export describes the function of exporting simulation data to a text file. Appendix D contains the FDUCD for Import/Export which describes the use case in more detail.

4.2.2 Design

The use case starts with the UI calling on the “Import” method in the “ActivityViewModel”. The view model then calls on the “ImportFromCSV” method in the “ImportExport” class. The “SelectFilePath” and “ReadCSV” methods are then run to get the data from the file. “ImportExport” calls on the “SaveActivityData” method, contained in the “DatabaseHandler” class, with the csv data as the parameter. If the connection to the database fails, “DatabaseHandler” calls on the static method “ShowError”.

The export function starts by calling on the “ExportToCSV” method in the “ActivityViewModel” at the press of the “Export” button.

The view model calls on the “ExportToCSV” method in the “Simulation” class which again calls on the “ExportToCSV” method in the “ImportExport” class with the simulator dataset as the parameter. If the simulator isn’t running, it instead calls on the “GetActivityData” method in the “DatabaseHandler”.

A detailed interaction diagram for the use case can be found in Appendix D.

4.3 Iteration 3: Simulate activity data

Iteration 3 selects the “Simulate Activity Data” use case for analysis and design.

4.3.1 Analysis

Simulate activity data describes the function of reading activity data from the database and simulating it in a graphical user interface. The simulator should run in real-time, update text boxes with the simulation information and animate the movement of a person depending on the current activity. Appendix E contains the FDUCD for Simulate activity data which describes the use case in more detail.

4.3.2 Design

The use case starts by having the UI call on the “StartSimulation” method in the “ActivityViewModel” at the click of the “Start Simulation” button. This method then calls on the “StartSimulation” method in the “Simulation” class. The “Simulation” class calls on the “GetActivityData” and “GetConfigPostions” method from the “DatabaseHandler” and stores the data in lists. If there are problems connecting to the database, the “DatabaseHandler” will call on the static method “ShowError”. Once the lists have been filled, the “SetInitialPosition” method is called to place the person at the initial position and the simulation loop starts. During the simulation loop, “Simulation” calls on the “UpdateSimulationStatus” method and calls on the “MovePerson” method in the “Person” class to get the next coordinates for the person and finally calls on the “AnimateMovement” method.

A detailed interaction diagram for the use case can be found in Appendix F.

4.4 Iteration 4: Add sequence of activities

Iteration 4 selects the “Add Sequence of Activities” use case for analysis and design.

4.4.1 Analysis

“Add sequence of activities” describes the function of adding new activities to the simulation while the simulator is running. The program should be able to read sequence parameters from text boxes and insert them into the activity list which the simulator reads from. Appendix G contains the FDUCD for “Add sequence of activities” and describes the use case in more detail.

4.4.2 Design

The use case starts with the UI calling on the “AddSequence” method in the “ActivityViewMode” class with the new sequence data as the parameters when the “Confirm Selection” button is pressed. The view model then calls on the “UpdateActivityList” method within the “Simulation” class to check the validity of the new sequence data. The method calls on the “AddSequenceToList” method within the “SequenceHandler” class if the data is valid and returns a new activity list to the “Simulation” class.

A detailed sequence diagram for the use case can be found in Appendix H.

4.5 Iteration 5: Replay previous set of activities

Iteration 5 selects the “Replay previous set of activities” use case for analysis and design.

4.5.1 Analysis

“Replay previous set of activities” describes the function of rewinding to a previous time in the simulation window. This function requires that the simulation is already running. Appendix I contains the FDUCD for “Replay previous set of activities” and describes the use case in more detail.

4.5.2 Design

The use case starts with the UI calling the “StartReplay” method in the “ActivityViewModel” class with “replayTime” as the parameter. This method then calls the “Replay” method in the “Simulation” class. If the replay time is outside of the simulation boundaries, the static method “ShowError” is called. If not, “SetInitialPosition” and “UpdateSimulationStatus” is called.

A detailed sequence diagram for the use case can be found in Appendix J.

4.6 Iteration 6: Configure properties and parameters

Iteration 6 selects the “Configure properties and parameters” use case for analysis and design.

4.6.1 Analysis

“Configure properties and parameters” describes the function of the user being able to change certain properties and parameters within the application and saving these changes to the database. Appendix K contains the FDUCD for “Configure properties and parameters” and describes the use case in more detail.

4.6.2 Design

The use case starts with the UI calling on the “SaveConfig” method in the “ActivityViewModel” class with the new config parameters as the method parameter. If not all the config textboxes have been filled, the view model will call on the “ShowError” method. If all the textboxes are filled, the view model will instead call on the

“SaveConfigChanges” method in the “ConfigHandler” class. Finally, the “SaveConfigToDatabase” method is called from the “DatabaseHandler” class to save the config data. If there are problems connecting to the database, the “ShowError” method is called.

A detailed sequence diagram for the use case can be found in Appendix L.

4.7 Class Diagram

From the sequence diagrams a class diagram is created. Figure 4.2 shows the class diagram for the application. The subchapters below describe which classes are created for each iteration.

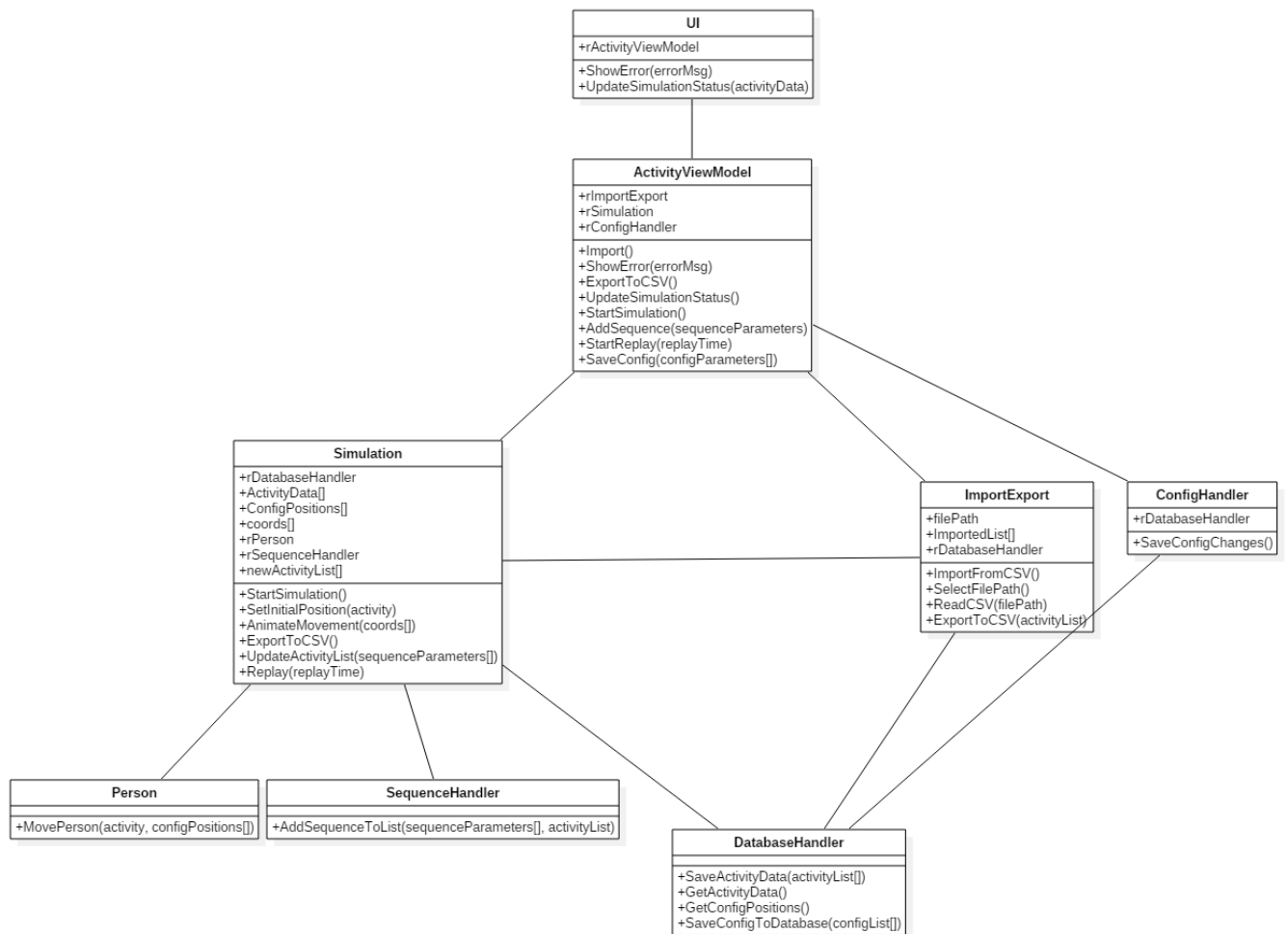


Figure 4.2: Class diagram for the Activity Simulator

4.7.1 Iteration 1: Collecting of requirements

As iteration 1 focuses on the UI prototype and the database structure, the classes created in this iteration are the UI, “ActivityViewModel” and “DatabaseHandler”.

4.7.2 Iteration 2: Import/Export

Iteration 2 involves the “Import/Export” use case. The “ImportExport” class is created during this iteration which is also used as the controller class.

4.7.3 Iteration 3: Simulate Activity Data

Iteration 3 involves the “Simulate Activity Data” use case. The classes made during this iteration are “Simulation” and “Person”. “Simulation” is used as the controller class.

4.7.4 Iteration 4: Add sequence of activities

Iteration 4 involves the “Add sequence of activities” use case. The class made during this iteration is “SequenceHandler” and the “Simulation” class is used as the controller class for this use case.

4.7.5 Iteration 5: Replay previous set of activities

Iteration 5 involves the “Replay previous set of activities” use case. No classes are made during this iteration, but rather a method within the “Simulation” class. The “Simulation” class is used as the controller class for this use case.

4.7.6 Iteration 6: Configure properties and parameters

Iteration 6 involves the “Configure properties and parameters” use case. The class made during this iteration is “ConfigHandler” which is also used as the controller class.

5 Implementation

Once a use case has gone through the design phase, they are ready to be implemented into the actual software. The classes and objects mapped out in the design phase is used as a starting point for the implementation phase. The source code for the C# application and for the database is sent to the supervisors separately as agreed upon between both parties.

5.1 User Interface

The user interface for the application is mainly designed using XAML [14] which is a markup language developed by Microsoft for use with the .NET programming model. The main page for the user interface can be found in Figure 5.1. The UI is implemented with a fixed size of 1400x720 pixels.

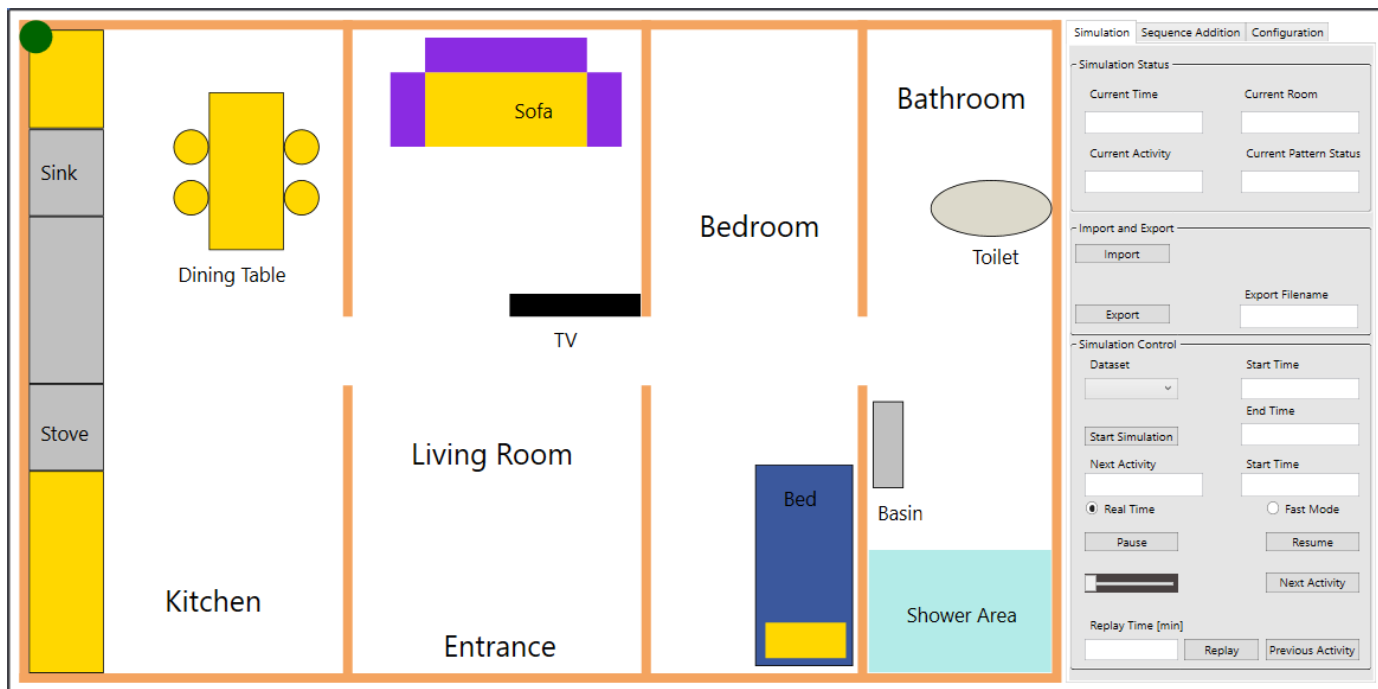


Figure 5.1: Main page for the user interface

5.1.1 Drawing Area

The drawing area of the UI is a Canvas element. A canvas is an UI element where child elements can be placed and positioned using absolute positioning [15]. This means that all canvas child elements each have their own vertical and horizontal coordinates that can be altered.

5.2 Model-view-viewmodel

The application is implemented based on the MVVM pattern [16] which purpose is to create a separation between the user interface and the rest of the application. Figure 5.2 shows the usage of the MVVM pattern on the class architecture. The classes are layered into view,

5 Implementation

model and view model. The view is the user interface, the model is the business layer, while the view model is responsible for connecting these layers together.

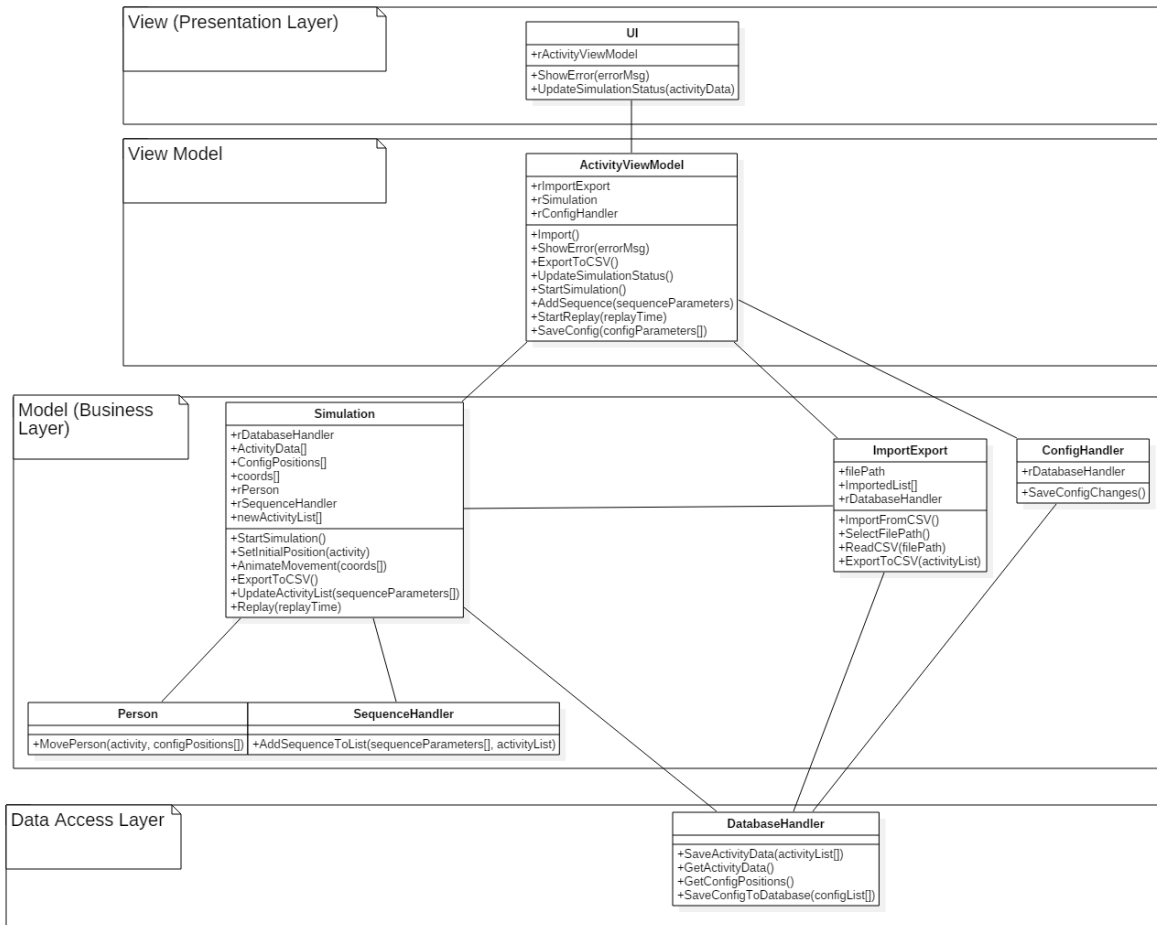


Figure 5.2: Class architecture based on the MVVM pattern

5.3 Data Context and Data Bindings

The view and the model communicate mainly through the use of data bindings. In WPF, data bindings is the process that allows the user interface to communicate with the underlying business layer [17]. If the bindings are set up correctly and the source properties within the business layer gives the UI the proper notifications, the UI can reflect changes that happens in the business layer. Reversely, changes done in the UI can be reflected in the business layer.

In order to set up the UI for connections via the data bindings, the data context [18] of the UI must be set to be the view model. The data context serves as the source for the bindings. Figure 5.3 shows a code snippet from the application where the view's data context is set to be the view model.

```
viewModel = new ActivityViewModel();
this.DataContext = viewModel;
```

Figure 5.3: Code snippet from the “MainWindow” class. The data context of the class is defined to be the “ActivityViewModel” class.

The classes with source properties to be monitored or controlled by the UI is instantiated in the view model. Figure 5.4 shows the code snippet in the view model where this is done.

```
Simulation simulation = new Simulation();
ConfigHandler configHandler = new ConfigHandler();
ImportExport importExport = new ImportExport();
0 references | SGlittum, 21 days ago | 1 author, 1 change
public Simulation Simulation
{
    get
    {
        return simulation;
    }
}
0 references | SGlittum, 21 days ago | 1 author, 1 change
public ConfigHandler ConfigHandler
{
    get
    {
        return configHandler;
    }
}
0 references | SGlittum, 21 days ago | 1 author, 1 change
public ImportExport ImportExport
{
    get
    {
        return importExport;
    }
}
```

Figure 5.4: The classes that the view should communicate with are instantiated and are accessed by the use of properties

Public properties contained in the “Simulation”, “ConfigHandler” and “ImportExport” classes can now be directly accessed from the user interface via the XAML code. Figure 5.5 shows the usage of data bindings on the “Current Activity” text box in the XAML code. For this particular data binding, the mode is set to “OneWay”, which means that the textbox value is updated whenever the model property changes. The mode can also be set to “OneWayToSource”, where the model property is updated whenever the textbox value changes, or “TwoWay”, which combines the two former modes.

```
<TextBox x:Name="txtCurrentActivity" Text="{Binding Simulation.Activity, Mode=OneWay}"
```

Figure 5.5: Data binding for the current activity text box in the user interface

When using either “OneWay” or “TwoWay” data bindings, the source properties need to raise some sort of event when the property values change to update the text boxes in the UI. This is resolved by using the “INotifyPropertyChanged” interface [19]. The method to raise the event is contained in a class called “ObservableObject” which inherits the “INotifyPropertyChanged” interface. Figure 5.6 shows the code contained within the “ObservableObject” class.

```

3 references | SGlittum, 21 days ago | 1 author, 1 change
public class ObservableObject : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    24 references | SGlittum, 21 days ago | 1 author, 1 change
    protected void OnPropertyChanged(string propertyName)
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    }
}

```

Figure 5.6: Code contained in the “ObservableObject” class

Classes which contains properties that uses the method contained in the “ObservableObject” class, inherits the class. Figure 5.7 shows the “Activity” property in the “Simulation” class. Whenever the property is set, the “OnPropertyChanged” method within “ObservableObject” is invoked and returns an indication of whether the property has changed or not. If the property has changed, the “Current Activity” textbox is updated.

```

public string Activity
{
    get
    {
        if (string.IsNullOrEmpty(_activity))
            return "Unknown";
        return _activity;
    }
    set
    {
        _activity = value;
        OnPropertyChanged("Activity");
    }
}

```

Figure 5.7: Public property “Activity” contained in the Simulation class

5.4 Database Server

The database is implemented to a Microsoft SQL Server with the help of Microsoft SQL Server Management Studio according to the logical architecture found in Appendix C.

5.5 Database Connection

The “DatabaseHandler” is the class responsible for connections to the database. The methods in the class uses a connection string to establish a connection to the server. Figure 5.8 shows the method [20] for getting the activity data from the database. The method opens a connection to the database using the connection string and reads from the database according to the SQL statement string.

```

public List<DataTypes> GetActivityData(int DataSetIndex)
{
    string connectionString = ConfigurationManager.ConnectionStrings["DatabaseConnectionString"].ConnectionString;
    List<DataTypes> activityDataList = new List<DataTypes>();
    SqlConnection con = new SqlConnection(connectionString);
    string selectSQL = String.Format("SELECT ActivityData.StartTime, ActivityData.EndTime, SensorLocation.Location, " +
        "Room.Room, Activity.Activity FROM ActivityData LEFT JOIN SensorLocation " +
        "ON ActivityData.SensorLocationId = SensorLocation.SensorLocationId INNER JOIN Room ON " +
        "SensorLocation.RoomId = Room.RoomId LEFT JOIN Activity ON ActivityData.ActivityId = Activity.ActivityId " +
        "WHERE ActivityData.DataSetIndex = {0}", DataSetIndex);
    try
    {
        con.Open();
        SqlCommand cmd = new SqlCommand(selectSQL, con);
        SqlDataReader dr = cmd.ExecuteReader();
        if (dr != null)
        {
            while (dr.Read())
            {
                DataTypes dataTypes = new DataTypes();

                dataTypes.StartTime = Convert.ToDateTime(dr["StartTime"]);
                dataTypes.EndTime = Convert.ToDateTime(dr["EndTime"]);
                dataTypes.Location = Convert.ToString(dr["Location"]);
                dataTypes.Room = Convert.ToString(dr["Room"]);
                dataTypes.Activity = Convert.ToString(dr["Activity"]);

                activityDataList.Add(dataTypes);
            }
        }
        con.Close();
    }
    catch (Exception ex)
    {
        ActivityViewModel.ShowMessageBox(ex.ToString(), "Error!");
    }
    return activityDataList;
}

```

Figure 5.8: Method in the "DatabaseHandler" class that gets the activity data from the database and stores it in a list

5.6 Import/Export

The import and export section of the UI is located in a group box. It consists of an import button, an export button and a textbox to enter the file name of the exported file. Figure 5.9 shows the “Import and Export” section of the user interface.

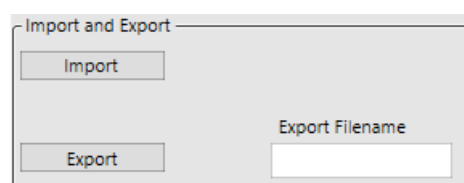


Figure 5.9: Import/Export section of the user interface

Once the user presses the import button, the “ImportFromCSV” method is called. This method also calls on the “SelectFilePath” method which lets the user select a csv file to import. The csv file must be of the same format as the csv file being exported by the export method, i.e. only semicolons as the separator. Once a file has been selected, the method saves the data to a list before it calls on a method in the “DatabaseHandler” to save the data.

```

public void ImportFromCSV(int previousDatasetIndex)
{
    DatabaseHandler dbHandler = new DatabaseHandler();
    List<DataTypes> importedList = new List<DataTypes>();
    SelectFilePath();
    try
    {
        using (var reader = new StreamReader(filePath))
        {
            while (!reader.EndOfStream)
            {
                DataTypes dataTypes = new DataTypes();
                var line = reader.ReadLine();
                var values = line.Split(';');
                dataTypes.StartTime = Convert.ToDateTime(values[0]);
                dataTypes.EndTime = Convert.ToDateTime(values[1]);
                dataTypes.Location = Convert.ToString(values[2]);
                dataTypes.Room = Convert.ToString(values[3]);
                dataTypes.Activity = Convert.ToString(values[4]);

                importedList.Add(dataTypes);
            }
        }
        dbHandler.SaveStartTimeAndEndTime(importedList, previousDatasetIndex);
        dbHandler.SaveActivityData(importedList, previousDatasetIndex);
    }
    catch (Exception ex)
    {
        ActivityViewModel.ShowMessageBox(ex.ToString(), "Error!");
    }
}

```

Figure 5.10: “ImportFromCSV” method contained within the “ImportExport” class.

The user can export the activity data to a csv file by first entering in a name for the target file and pressing the export button. The “ExportToCSV” method in the “ImportExport” class is then called with the activity list used in the simulation as a parameter. If the list is empty, i.e. if the simulator has not started, the method will instead call on the “GetActivityData” method from the “DatabaseHandler” to get the unaltered activity data. The “SelectFolderPath” method is called to allow the user to specify the folder in which the file is saved, and the data is converted to a string format with semicolons as the separator. Finally, the string is written

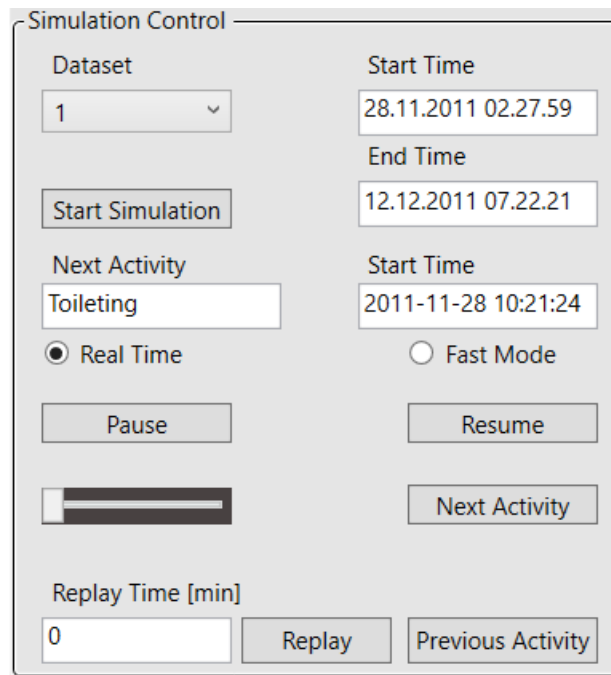
onto a csv file. Figure 5.11 shows a code snippet of the “ExportToCSV” method used to export activity data to csv format.

```
public void ExportToCSV(int dataSetIndex, List<DataTypes> activityList, string exportFileName)
{
    if(!activityList.Any())
    {
        DatabaseHandler dbHandler = new DatabaseHandler();
        activityList = dbHandler.GetActivityData(dataSetIndex);
    }
    SelectFolderPath();
    string csv = "";
    for (int i = 0; i < activityList.Count; i++)
    {
        csv = csv + string.Format("{0:yyyy/MM/dd hh:mm:ss};{1:yyyy/MM/dd hh:mm:ss};{2};{3};{4}" + "\n",
            Convert.ToString(activityList[i].StartTime), Convert.ToString(activityList[i].EndTime),
            Convert.ToString(activityList[i].Location),
            Convert.ToString(activityList[i].Room), Convert.ToString(activityList[i].Activity));
    }
    try
    {
        System.IO.File.WriteAllText(string.Format(@"{0}" + "\\{1}.csv", folderName, exportFileName), csv);
    }
    catch(Exception ex)
    {
        ActivityViewModel.ShowMessageBox(ex.ToString(), "Error!");
    }
}
```

Figure 5.11: “ExportToCSV” method contained in the “ImportExport” class

5.7 Simulate Activity Data

The program starts the simulation whenever the user presses the “Start Simulation” button. Figure 5.12 shows a screenshot of the “Simulation Control” in the user interface where the user can select a dataset to simulate, start the simulation, pause or resume the simulation, increase the simulation speed, skip to the next activity in the simulation or rewind the simulation by time or activity.

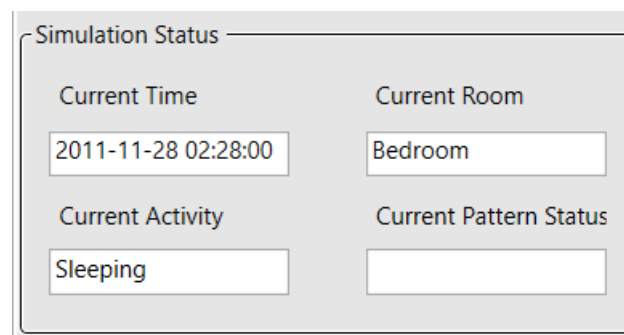


The Simulation Control UI section includes the following elements:

- Dataset:** A dropdown menu showing '1'.
- Start Time:** A text box containing '28.11.2011 02.27.59'.
- End Time:** A text box containing '12.12.2011 07.22.21'.
- Start Simulation:** A button.
- Next Activity:** A text box containing 'Toileting'.
- Start Time:** A text box containing '2011-11-28 10:21:24'.
- Real Time:** A radio button that is selected.
- Fast Mode:** An unselected radio button.
- Pause:** A button.
- Resume:** A button.
- Progress Bar:** A horizontal bar with a black fill and a white slider.
- Next Activity:** A button.
- Replay Time [min]:** A text box containing '0'.
- Replay:** A button.
- Previous Activity:** A button.

Figure 5.12: Simulation control section of the UI

The user has access to the current simulation time, current room and current activity of the person from the simulation status section shown in Figure 5.13.



The Simulation Status UI section includes the following elements:

- Current Time:** A text box containing '2011-11-28 02:28:00'.
- Current Room:** A text box containing 'Bedroom'.
- Current Activity:** A text box containing 'Sleeping'.
- Current Pattern Status:** An empty text box.

Figure 5.13: Simulation status section of the UI

A new thread is then created and started to make sure that the user can still interact with other parts of the program. Figure 5.14 shows the creation and start of the simulation thread. The thread starts the “StartSimulation” method which includes a simulation loop.


```
public void StartSimulationThread()
{
    if (simulationThreadFinished)
    {
        simulationThreadFinished = false;
        simulationCaller = new Thread(new ThreadStart(StartSimulation));
        simulationCaller.Start();
    }
    else
    {
        ActivityViewModel.ShowMessageBox("Simulation already running", "Error!");
    }
}
```

Figure 5.14: Starting of the simulation thread in the Simulation class

The “StartSimulation” method calls on the “GetActivityData” method in the database handler and saves the data in a list. It then enters the relevant data into public properties every loop iteration to make sure they appear in the user interface. The loop iterates every second with the help of a “Thread.Sleep” statement.

Whenever the activity changes in the simulation, the animation thread is called. Figure 5.15 shows the “AnimateMovement” method which the animation thread calls upon.

```

private void AnimateMovement()
{
    const int animationSpeed = 3;
    string targetActivity = activityList[listIndex].Activity;
    List<Coordinates> coordsList = new List<Coordinates>();
    coordsList = person.MovePerson(targetActivity, 1);
    for (int i = 0; i < coordsList.Count; i++)
    {
        if (coordsList[i].Dimension == "X")
        {
            while (PersonPositionX < coordsList[i].Value)
            {
                PersonPositionX++;
                Thread.Sleep((int)(Math.Round(animationSpeed / SimulationSpeed)));
            }
            while (PersonPositionX > coordsList[i].Value)
            {
                PersonPositionX--;
                Thread.Sleep((int)(Math.Round(animationSpeed / SimulationSpeed)));
            }
        }
        if (coordsList[i].Dimension == "Y")
        {
            while (PersonPositionY < coordsList[i].Value)
            {
                PersonPositionY++;
                Thread.Sleep((int)(Math.Round(animationSpeed / SimulationSpeed)));
            }
            while (PersonPositionY > coordsList[i].Value)
            {
                PersonPositionY--;
                Thread.Sleep((int)(Math.Round(animationSpeed / SimulationSpeed)));
            }
        }
    }
    animationThreadFinished = true;
}

```

Figure 5.15: “AnimateMovement” method in the Simulation class. The method gets a list of coordinates which is either specified as x or y. It then checks whether the current position of the person is correct according to the activity coordinates gotten from the database. If it is not, the method adds or subtracts the current position value until the person is at the correct position.

The animation works by data binding the vertical value and the horizontal value of a circle in the user interface and updating the values every three milliseconds to make it appear that it is moving. The x and y coordinates for the next activity is returned by the “MovePerson” method in the Person class. The “MovePerson” method takes the next activity as one of the parameters and crosschecks the activity with the activities in the “ActivityPosition” table in the database and gets the target coordinates. Figure 5.16 shows a screenshot of the “MovePerson” method.

```

public List<Coordinates> MovePerson(string targetActivity, int configId)
{
    List<ConfigPositions> configPositionList = new List<ConfigPositions>();
    DatabaseHandler dbHandler = new DatabaseHandler();
    List<Coordinates> coordsList = new List<Coordinates>();
    configPositionList = dbHandler.GetConfigPositions(configId);
    double doorVerticalPosition = dbHandler.GetDoorVerticalPosition(configId);
    double xPos = 0;
    double yPos = 0;
    for(int i=0;i<configPositionList.Count;i++)
    {
        if(targetActivity == configPositionList[i].Activity)
        {
            xPos = configPositionList[i].XPos;
            yPos = configPositionList[i].YPos;
            break;
        }
    }
    Coordinates doorCoord = new Coordinates
    {
        Dimension = "Y",
        Value = doorVerticalPosition
    };
    Coordinates xCoord = new Coordinates
    {
        Dimension = "X",
        Value = xPos
    };
    Coordinates yCoord = new Coordinates
    {
        Dimension = "Y",
        Value = yPos
    };
    coordsList.Add(doorCoord);
    coordsList.Add(xCoord);
    coordsList.Add(yCoord);
    return coordsList;
}

```

Figure 5.16: “MovePerson” method in the Person class

5.8 Add Sequence of Activities

The user is able to manually add a new sequence into the simulation list while the simulator is running. This sequence includes start time, end time, room and activity. Figure 5.17 shows the sequence addition section of the user interface. When the “Confirm Selection” button is pressed, the new sequence is added to the simulator.

Figure 5.17: Sequence Addition section of the UI

Once the “Confirm Selection” button is pressed, the program calls on the “AddSequenceToList” method in the “SequenceHandler” class with the new sequence data and the existing simulator list as parameters. Figure 5.18 shows the “UpdateActivityList” method contained in the “Simulation” class. This method checks if the new sequence data is valid and returns an error if it isn’t. If the data is valid, the method calls on the “AddSequenceToList” method with the simulation list, list index and the sequence data as the parameters. The method then inserts a new object into the list at the next index and returns a new simulation list.

```

public void UpdateActivityList()
{
    if (NewStartDate.Date + NewStartTime.TimeOfDay < CurrentTime)
    {
        ActivityViewModel.ShowMessageBox("Start time must be later than current time", "Error entering sequence");
    }
    else if (NewStartDate.Date + NewStartTime.TimeOfDay > NewEndDate.Date + NewEndTime.TimeOfDay)
    {
        ActivityViewModel.ShowMessageBox("Start time can not be later than end time", "Error entering sequence");
    }
    else if (NewActivity == "" || NewRoom == "")
    {
        ActivityViewModel.ShowMessageBox("Enter a room and activity combo", "Error entering sequence");
    }
    else
    {
        activityList = sequence.AddSequenceToList(activityList, listIndex, NewStartDate, NewEndDate,
            NewStartTime, NewEndTime, NewRoom, NewActivity);
        ShowNextActivity();
        ActivityViewModel.ShowMessageBox("Sequence added to list", "Success!");
    }
}

```

Figure 5.18: “UpdateActivityList” method in the Simulation class

5.9 Replay Previous Set of Activities

The application supports the function of replaying back to a previous time. This can either be done by entering number into the replay time textbox and pressing replay, which rewinds the simulation the specified number of minutes, or pressing the previous activity button. Figure 5.19 shows a screenshot of the replay section of the user interface.

Figure 5.19: Replay section of the UI

5 Implementation

Once the user enters a value in the replay time textbox and presses the replay time, the application calls on the “Replay” method in the “Simulation” class. This method rewinds the current simulation time the number of minutes specified and updates the current room and current activity if they were different at that time. If the resulting simulation time is outside of the simulation boundaries, an error message will appear. Figure 5.20 shows a screenshot of the “Replay” method contained in the simulation class.

```
public void Replay()
{
    DateTime tempCurrentTime = CurrentTime;
    tempCurrentTime = tempCurrentTime.AddMinutes(-ReplayTime);
    for(int i = 0; i < activityList.Count; )
    {
        if(tempCurrentTime >= activityList[i].StartTime && tempCurrentTime < activityList[i].EndTime)
        {
            listIndex = i;
            CurrentTime = tempCurrentTime;
            Room = activityList[listIndex].Room;
            Activity = activityList[listIndex].Activity;
            SetInitialPosition(activityList[listIndex].Activity);
            break;
        }
        else if(tempCurrentTime >= activityList[i].EndTime && tempCurrentTime < activityList[i+1].StartTime)
        {
            listIndex = i + 1;
            CurrentTime = tempCurrentTime;
            Room = activityList[listIndex].Room;
            Activity = activityList[listIndex].Activity;
            SetInitialPosition(activityList[listIndex].Activity);
            break;
        }
        else if(tempCurrentTime < activityList[i].StartTime)
        {
            MessageBox.Show("Replay time outside of simulation boundaries", "Error!");
            break;
        }
        else
        {
            i++;
        }
    }
}
```

Figure 5.20: Replay method in the Simulation class

5.10 Configure Properties and Parameters

There are some properties and parameters that needs to be configured in order for the simulation to run successfully. For this application, this includes the positions in which the activities take place. These positions are saved into the database for the application to get when the simulation starts. Figure 5.21 shows the section in the UI for configuring the activity positions. The parameters are saved to the database when the “Save Configuration” button is pressed.

Figure 5.21: Section for configuring the activity positions

The textboxes are connected to the “ConfigHandler” class via data bindings which means that public properties in the class changes values whenever the textbox values change. Once the “Save Configuration” button is pressed, the “SaveConfigChanges” method is called upon. This method saves the data into a list before exporting the list to the database. Figure 5.22 shows part of the method where the parameters are separated into x and y values, added to a list and finally saved to the database. The database table which the positional parameters are saved to can be seen in Figure 5.23.

As the activity positions can be changed at any time, objects may be moved within the UI drawing area or a new house design may be drawn altogether so long as the activity positions are saved at the start of the program.

5.10.1 Extending with additional parameters

Right now, the positional data of the activities is the only parameters that can be configured. In order to add additional parameters to be configured, new source properties in the “ConfigHandler” must be created and data bindings must be added to the inputs in the UI. The new parameters must then be saved to lists of their own and call on a method in the “DatabaseHandler” to save them to the database. New tables in the database may need to be created depending on the type of properties to be saved.

```

bedX = Convert.ToDouble(BedCoords.Split(separator)[0]);
bedY = Convert.ToDouble(BedCoords.Split(separator)[1]);
toiletX = Convert.ToDouble(ToiletCoords.Split(separator)[0]);
toiletY = Convert.ToDouble(ToiletCoords.Split(separator)[1]);
showerAreaX = Convert.ToDouble(ShowerCoords.Split(separator)[0]);
showerAreaY = Convert.ToDouble(ShowerCoords.Split(separator)[1]);
diningTableX = Convert.ToDouble(DiningCoords.Split(separator)[0]);
diningTableY = Convert.ToDouble(DiningCoords.Split(separator)[1]);
basinX = Convert.ToDouble(BasinCoords.Split(separator)[0]);
basinY = Convert.ToDouble(BasinCoords.Split(separator)[1]);
sofaX = Convert.ToDouble(SofaCoords.Split(separator)[0]);
sofaY = Convert.ToDouble(SofaCoords.Split(separator)[1]);
entranceX = Convert.ToDouble(EntranceCoords.Split(separator)[0]);
entranceY = Convert.ToDouble(EntranceCoords.Split(separator)[1]);
doorsY = Convert.ToDouble(DoorsCoords.Split(separator)[1]);
AddActivityCoords(bedX, nameof(bedX));
AddActivityCoords(bedY, nameof(bedY));
AddActivityCoords(toiletX, nameof(toiletX));
AddActivityCoords(toiletY, nameof(toiletY));
AddActivityCoords(showerAreaX, nameof(showerAreaX));
AddActivityCoords(showerAreaY, nameof(showerAreaY));
AddActivityCoords(diningTableX, nameof(diningTableX));
AddActivityCoords(diningTableY, nameof(diningTableY));
AddActivityCoords(basinX, nameof(basinX));
AddActivityCoords(basinY, nameof(basinY));
AddActivityCoords(sofaX, nameof(sofaX));
AddActivityCoords(sofaY, nameof(sofaY));
AddActivityCoords(entranceX, nameof(entranceX));
AddActivityCoords(entranceY, nameof(entranceY));
DatabaseHandler dbHandler = new DatabaseHandler();
dbHandler.SaveDoorsLocationToDatabase(doorsY, 1);
dbHandler.SaveConfigLocationToDatabase(configPositionList, 1);

```

Figure 5.22: Code snippet of the SaveConfigChanges method in the ConfigHandler class

	ActivityPositionId	Position	XPos	YPos	ActivityId	ConfigId
1	29	Bed	792	541	1	1
2	30	Toilet	986	192	2	1
3	31	ShowerArea	949	596	3	1
4	32	DiningTable	241	172	4	1
5	33	Basin	873	411	5	1
6	34	Sofa	519	89	6	1
7	35	Entrance	487	632	7	1
8	36	DiningTable	241	172	8	1
9	37	DiningTable	241	172	9	1

Figure 5.23: Table containing the positional data of the activities

6 Testing

This chapter describes the testing of the system and the methods that are used. The testing is based on the requirements, use case diagram and the individual FDUCDs.

The testing done can be categorized into undocumented and documented. The undocumented testing is done every time a new feature is added into the system. The feature is then tested right away without documentation. The documented testing is done by the help of test cases where two test cases is created for each of the use cases, one for the main success scenario and one for the extensions. The template used for these test cases was found online on softwaretestinghelp.com [21].

6.1 Import/Export

Testing of the Import/Export features included pressing the “Import” button, waiting for the file dialog box to open and selecting a csv file. The database was then checked to see if the new dataset had been saved. The export function was tested by entering a new file name into the textbox and pressing the “Export” button, waiting for the file dialog box to open and selecting a folder. The new csv file was then inspected to see if the dataset had been successfully exported.

The extensions were tested by shutting down the database server, pressing the “Import” button and checking if an error message appeared.

The detailed test cases for the “Import/Export” use case can be found in Appendix N and Appendix O.

6.2 Simulate Activity Data

The testing of the main success scenario for the “Simulate Activity Data” use case included pressing the “Start Simulation” button to see if the simulation status parameters started updating and to see if the person appeared at the initial activity position. The program then ran until a new activity occurred to see if the simulation status updated once more and to see if the person moved to the new activity location.

Additional features tested included pausing the simulator, resuming the simulator, fast-forwarding the simulation speed, pressing the “Next Activity” button and toggling “Fast Mode”.

Extensions for the use case was tested by shutting down the database and starting the simulation to see if an error message appeared.

Detailed test cases for the use case can be found in Appendix P and Appendix Q.

6.3 Add Sequence of Activities

The main success scenario of the use case is tested by entering in new valid sequence parameters into the textboxes and combo boxes and pressing “Confirm Selection”.

The sequence time is specified to be close to the current simulation time in order to easily check that the activity has been added to the simulation.

The randomize function is also tested by pressing the “Randomize” button and checking if the sequence parameters in the text boxes and combo boxes gives random and valid values.

The extensions are tested by entering invalid sequence parameters. This included entering a start time earlier than the current time, entering an end time earlier than the start time or leaving the room and/or activity combo boxes empty.

Detailed test cases for the use case can be found in Appendix R and Appendix S.

6.4 Replay Previous Set of Activities

The main success scenario for the use case is tested by specifying a replay time in the text box and pressing the “Replay” button. The simulation should then rewind to a previous simulation time specified by the value in the “Replay Time” text box.

The previous activity feature is also tested by pressing the “Previous Activity” button and making sure that the simulator rewinds to a time where the activity was different.

The extensions are tested by entering a replay time where the simulator would rewind to time outside of the simulation boundaries and checking if an error message appears.

Detailed test cases for the use case can be found in Appendix T and Appendix U.

6.5 Configure Properties and Parameters

The main success scenario for the use case is tested by toggling the “Activate Editing” checkbox in the configuration section and filling each of the activity location text boxes with coordinates. The “Save Configuration” text box is then pressed, and the database is checked to see if the values has been saved.

The extensions are tested by keeping any of the text boxes empty, pressing the “Save Configuration” textbox and checking if an error message appears. Error handling concerning database connection is also tested by shutting down the database and waiting for an error message.

Detailed test cases for the use case can be found in Appendix V and Appendix W.

7 Discussion

By performing a literature survey on existing systems similar to the one created in this project, it seems clear that there does not exist any systems today that fulfill the requirements of the system developed in this report.

The datasets used for the simulation comes in the form of two text files. The first being sensor data and the other being activity data. There are, however, a higher number of recorded data in the sensor data file than in the activity data file. This results in a combined dataset where some of the activity values are empty.

The simulator animation currently only supports doors where the person moves through it horizontally. These doors also have to be aligned with each other, i.e. they need to have the same vertical coordinate value. This means that if the doors are not aligned or the person have to move vertically through a door, the person may potentially walk through the walls to reach the target.

7.1 Including Additional Persons in the Simulator

Right now, the simulator only simulates the activities of one person. This person's movements and behavior is based on a dataset containing sensor- and activity data which means that any additional persons added to the simulator needs to be based on separate datasets.

The application would need to be extended where the simulation status is shown for each person. The rooms and activities would need to be updated continuously for all the persons.

For the animation, new animation threads would need to be created equal to the number of new persons added.

The sequence addition can be extended by selecting the person whose data list should be altered.

Other considerations would be to ensure that certain activities are not performed simultaneously by several persons. Showering and toileting, for example, can only be performed by person at a time, unless new toilets and showers are added to the house.

8 Suggestions for further work

This chapter suggests further work to be done on the system.

8.1 Improve Animation

Right now, the animation is somewhat limited in that only aligned doors where the person moves through them horizontally is compatible with the animation algorithm. If the animation is improved to where the door locations doesn't matter, this would allow for much more complex house layouts.

Another improvement would be to add collision to make sure the person is unable to walk through any obstructions.

8.2 Save and Load House Drawings

Right now, a developer can edit the existing house drawing in the Visual Studio IDE but is not able save or load a drawing from the application. By providing the function of saving or loading the XAML code for the drawing, a user can choose which house to simulate from.

8.3 Improve Sequence Addition

The sequence addition feature may be improved by allowing the addition of sequences by clicking on the drawing area. When the user clicks on the drawing area, the program could "guess" the closest activity to the cursor and give a confirmation prompt to the selection.

8.4 Improve Import and Export

Supporting files other than csv for importing and exporting could be an improvement to the application.

9 Conclusion

A literature survey was carried out to research similar systems to the one developed in this project. These were HOME I/O by Real Games and Smart House Online Simulation and both systems were compared to the Activity Simulator developed in this project.

The development process followed a modified version of the UP framework and used OOAD for analysis and design of the system. Development was carried out in iterations where Iteration 1 focused on collecting the requirements for the system, creating a use case diagram, creating a prototype of the UI and designing the logical architecture for the database while Iteration 2 – 6 focused on individual use cases. Each use case was analyzed and designed by creating a FDUCD for each of the use cases during the analysis phase and creating an interaction diagram for each of them during the design phase. An overall class diagram was created from the interaction diagrams.

Implementation for the simulator application was done in Microsoft Visual Studio with WPF and C# as the programming language while implementation for the database was done in Microsoft SQL Server Management Studio. The software was successfully able to get data from the database and simulate it within the simulator application. It was also able to import dataset to the database from a csv file and export simulation data to a csv file, add new sequences of activities within the simulation, replay simulation and configure properties and parameters to save in the database.

Testing was performed on each of the use cases, both documented and undocumented. The undocumented testing was performed each time a new feature was added to the system, while the documented testing took place after all the use cases had been implemented into the system. The documented testing focused on the requirements, use cases and FDUCDs for each of the use cases.

References

- [1] Real Games. (2018, 18.04.18). *HOME I/O - 3D Smart Home Automation Simulator*. Available: <https://realgames.co/home-io/>
- [2] Astea Solutions. (n.d., 18.04.18). *Cloud4all Smart House Online Simulation*. Available: <https://smarthouse.remex.hdm-stuttgart.de/#/index>
- [3] Real Games. (2017, 18.04.18). *HOME I/O Documentation*. Available: <https://realgames.co/docs/homeio/en/>
- [4] Real Games. (2017, 18.04.18). *Controls - HOME I/O*. Available: <https://realgames.co/docs/homeio/en/controls/>
- [5] Real Games. (2017, 18.04.18). *Head-Up Display - HOME I/O*. Available: <https://realgames.co/docs/homeio/en/headup-display/>
- [6] Real Games. (2017, 18.04.18). *Device Modes - HOME I/O*. Available: <https://realgames.co/docs/homeio/en/device-modes/>
- [7] Real Games. (2017, 18.04.18). *HOME I/O SDK*. Available: <https://realgames.co/docs/homeio/en/sdk-getting-started/>
- [8] Real Games. (2017, 18.04.18). *CONNECT I/O*. Available: <https://realgames.co/docs/connectio/>
- [9] Real Games. (2017, 18.04.18). *Console - HOME I/O*. Available: <https://realgames.co/docs/homeio/en/console/>
- [10] Astea Solutions. (2014, 18.04.18). *Smart House Online Simulation - YouTube*. Available: <https://www.youtube.com/watch?v=m64spcp0QoM>
- [11] F. J. Ordóñez, P. de Toledo, and A. Sanchis. (2013, 15.01.18). *Activity Recognition Using Hybrid Generative/Discriminative Models on Home Environments Using Binary Sensors*. Available: [https://archive.ics.uci.edu/ml/datasets/Activities+of+Daily+Living+\(ADLs\)+Recognition+Using+Binary+Sensors](https://archive.ics.uci.edu/ml/datasets/Activities+of+Daily+Living+(ADLs)+Recognition+Using+Binary+Sensors)
- [12] N. O. Skeie, "Object-oriented Analysis, Design, and Programming using UML and C#," Lecture Notes, 2017.
- [13] MKLab Co., Ltd. (2018, 10.01.18). *StarUML*. Available: <http://staruml.io/>
- [14] Microsoft Corporation. (2017, 15.01.18). *XAML Overview (WPF)*. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/advanced/xaml-overview-wpf>
- [15] Microsoft Corporation. (2018, 12.02.18). *Canvas Class*. Available: [https://msdn.microsoft.com/en-us/library/system.windows.controls.canvas\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.controls.canvas(v=vs.110).aspx)
- [16] Microsoft Corporation. (2018, 15.01.18). *Implementing the Model-View-ViewModel Pattern*. Available: <https://msdn.microsoft.com/en-us/library/ff798384.aspx>
- [17] Microsoft Corporation. (2017, 02.02.18). *Data Binding Overview*. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/data/data-binding-overview>

- [18] Microsoft Corporation. (2018, 02.02.18). *FrameworkElement.DataContext Property*. Available: [https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.windows.frameworkelement.datacontext(v=vs.110).aspx)
- [19] Microsoft Corporation. (2018, 02.02.18). *INotifyPropertyChanged Interface*. Available: [https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged\(v=vs.110\).aspx](https://msdn.microsoft.com/en-us/library/system.componentmodel.inotifypropertychanged(v=vs.110).aspx)
- [20] H. P. Halvorsen. (n.d., 15.01.18). *Datalogging and Monitoring, A Practical Example using SQL Server, LabVIEW and Visual Studio/C#*. Available: <https://www.halvorsen.blog/documents/technology/resources/resources/Datalogging/Datalogging%20and%20Monitoring.pdf>
- [21] Software Testing Help. (2017, 11.04.18). *Sample Test Case Template with Examples*. Available: <https://www.softwaretestinghelp.com/test-case-template-examples/>

Appendices

Appendix A: Topic Description

HSN University College
of Southeast Norway
Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Developing an activity simulator of a person living in a smart house

HSN supervisor: Nils-Olav Skeie

Co-supervisor: Veralia Gabriela Sánchez

External partner: SMART research group at USN

Task background:

Smart houses and Smart buildings may refer to any living or working environment carefully designed to assist residents in carrying out daily activities and to promote independent lifestyle. The SMART research group at USN are working to develop a system to determine normal and abnormal behaviour of a person living in such a house or building. If abnormal behaviour occurs, then the system can create an alert for the family or care worker.

People tend to follow specific patterns in their daily lifestyle. In a Smart House context, the user's daily activity generates patterns that plays an important role to predict future events in the Smart Home. The goal of the Smart House and Smart Building environment is to help the user on their daily life activities; thus, the Smart House system may find repetitive patterns in user's activities and predict the behaviour of the user for additional assistance.

The main ideas of the simulator is to let a person moving around in a set of rooms in a Smart house. The simulator should include both create, view and replay options. The person's activities should be stored in a database with minimum the following parameters: time, room, and activity of the person. The simulator should contain a graphical interface with a view of the building 1) to show the person moving (both normal and abnormally patterns), 2) to create new patterns based on user input, 3) to create new patterns based on random selection from existing activities, and 4) to replay a set of activities with a graphical representation. The system should also contain a configuration part, and the programming language is C# (or another OO programming language). This project is suitable for anyone interested in game developing.

Task description:

- Make a literature survey of similar simulators and systems,
- Use OOAD for analysis and design of the simulator,
- Create a graphical simulator for a person, based on data from an external database,
- Include a configuration section for input of parameters and properties for the system,
- Integrate a database for both the configuration data and activity data,
- Include an input module for new sequences of activities,
- Include import and export options for external data,
- Make a test plan for the system,
- Discuss the impact of including several persons to the simulator.

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

Student category: IIA

Practical arrangements: Activity data from external databases is available.

Signatures:

13-Feb-18

Student (date and signature):

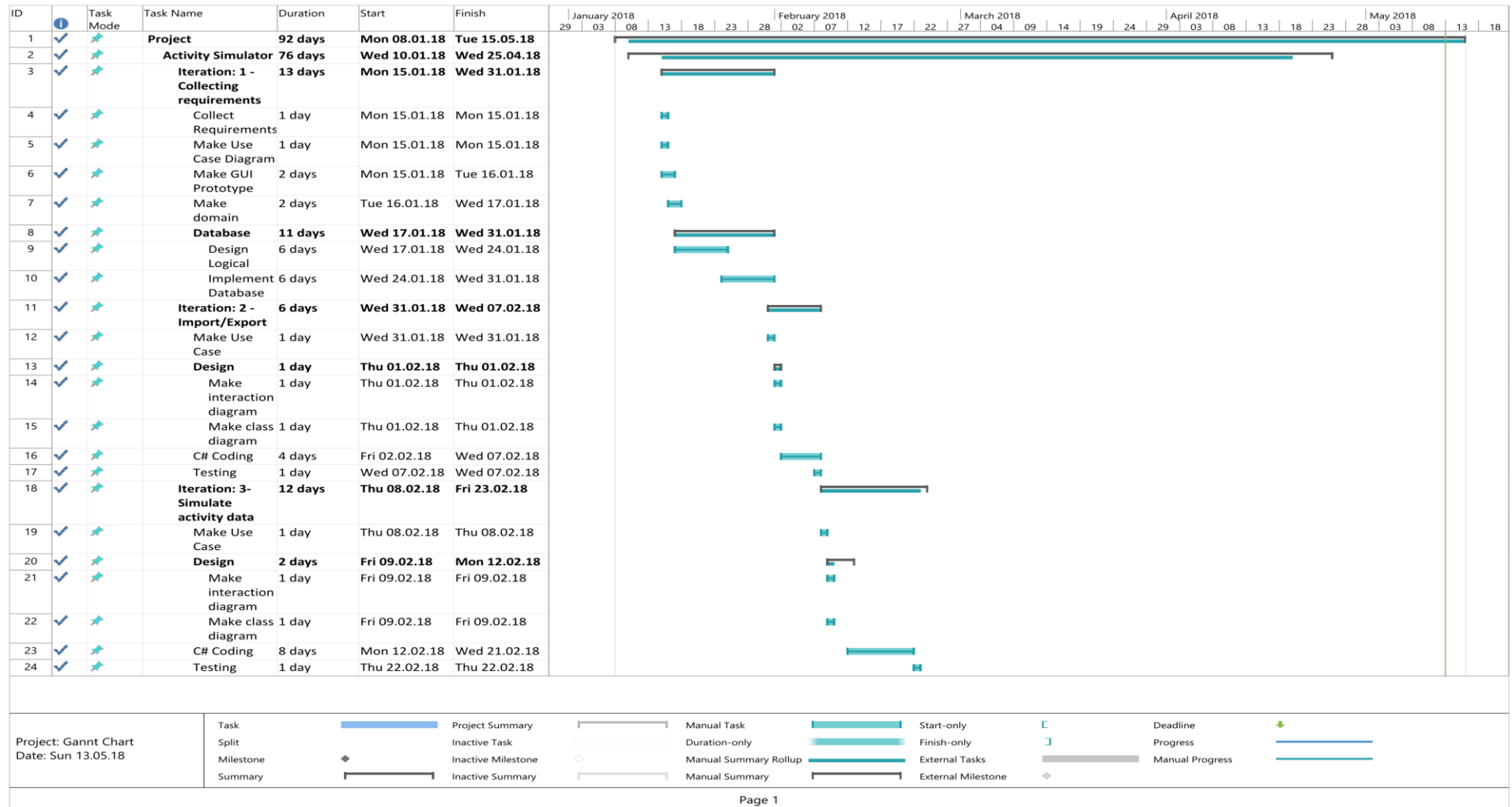
SG

(STIAN H. GLITUM)

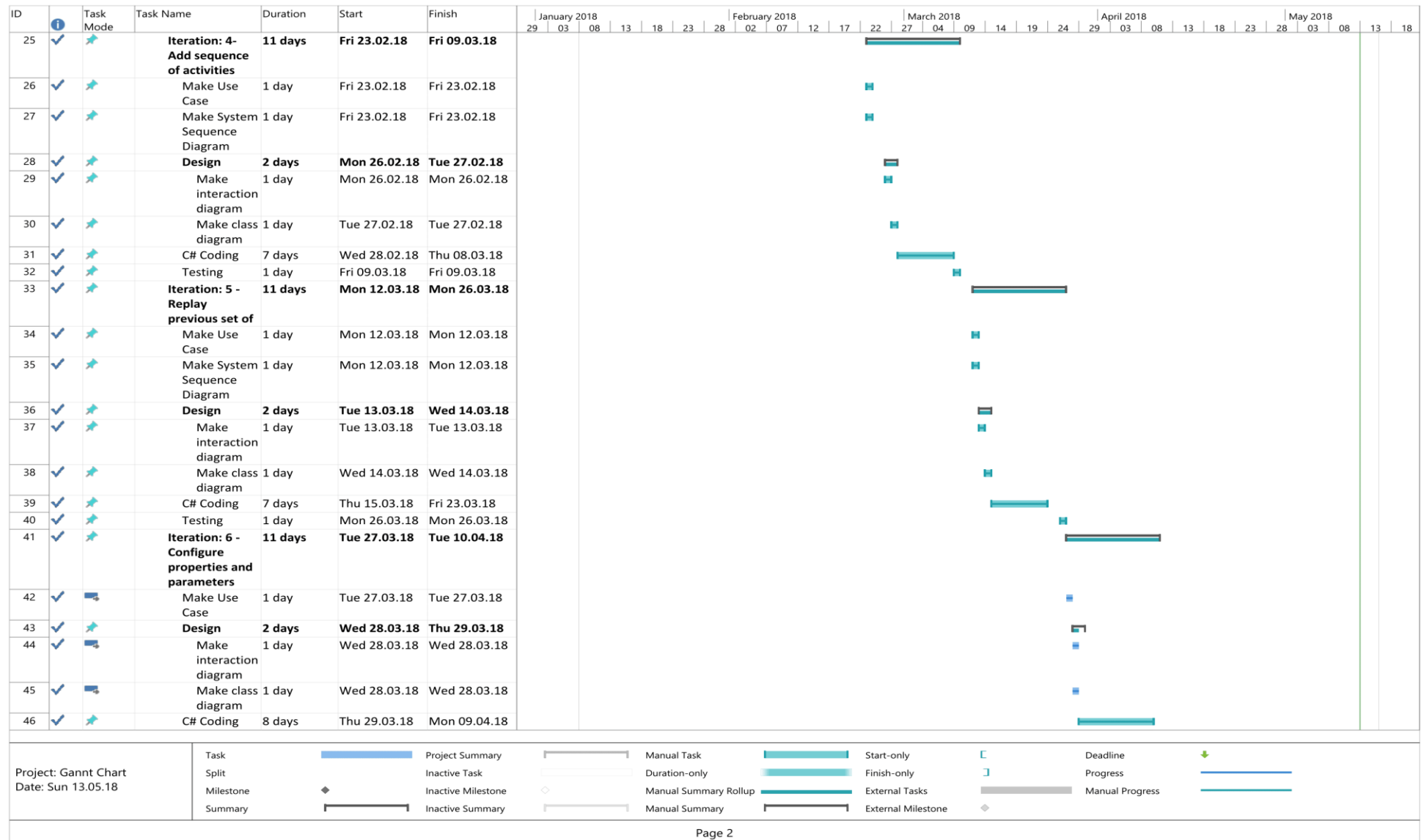
Supervisor (date and signature):

Nils J. Aune

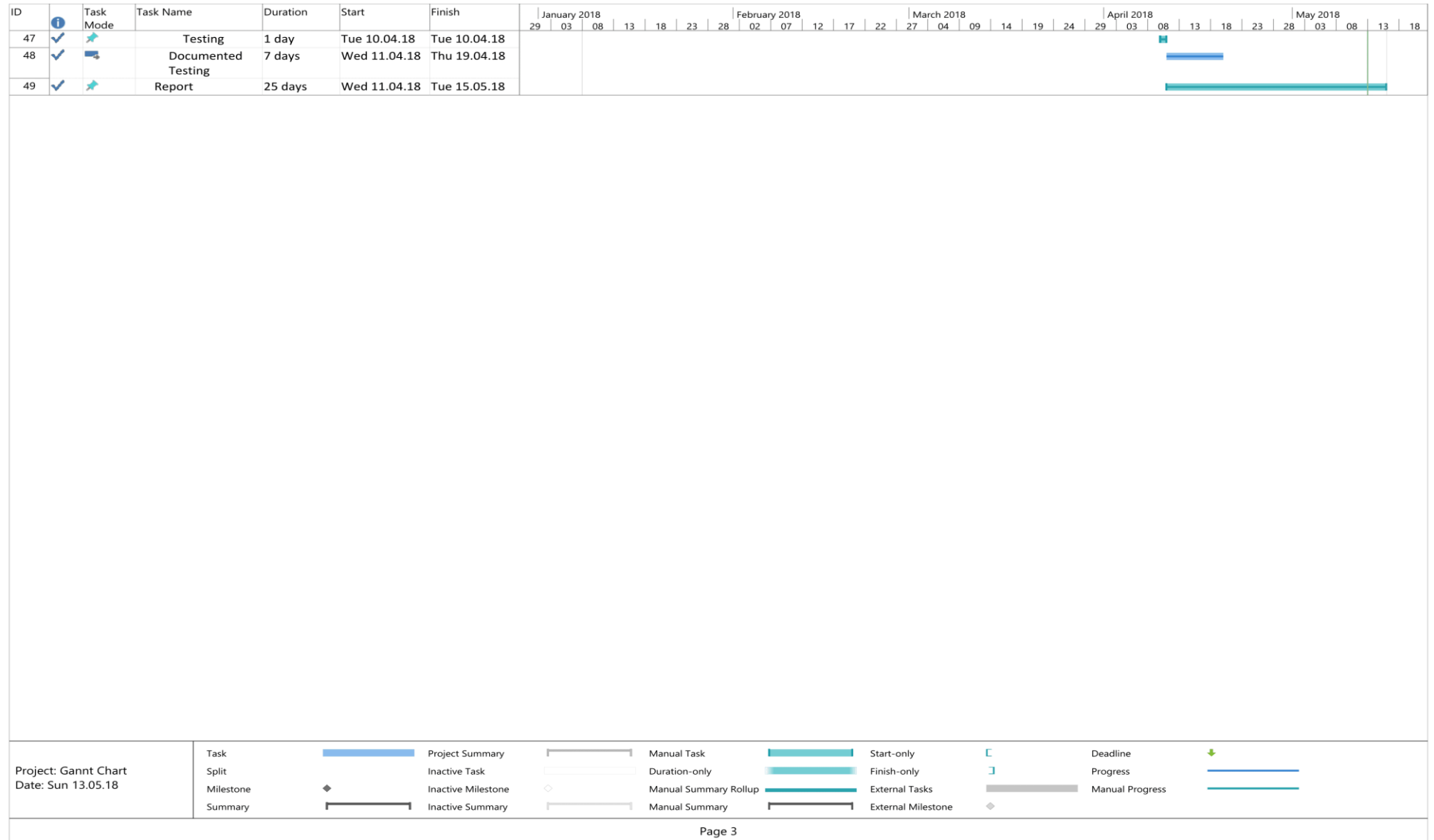
Appendix B: Work Schedule



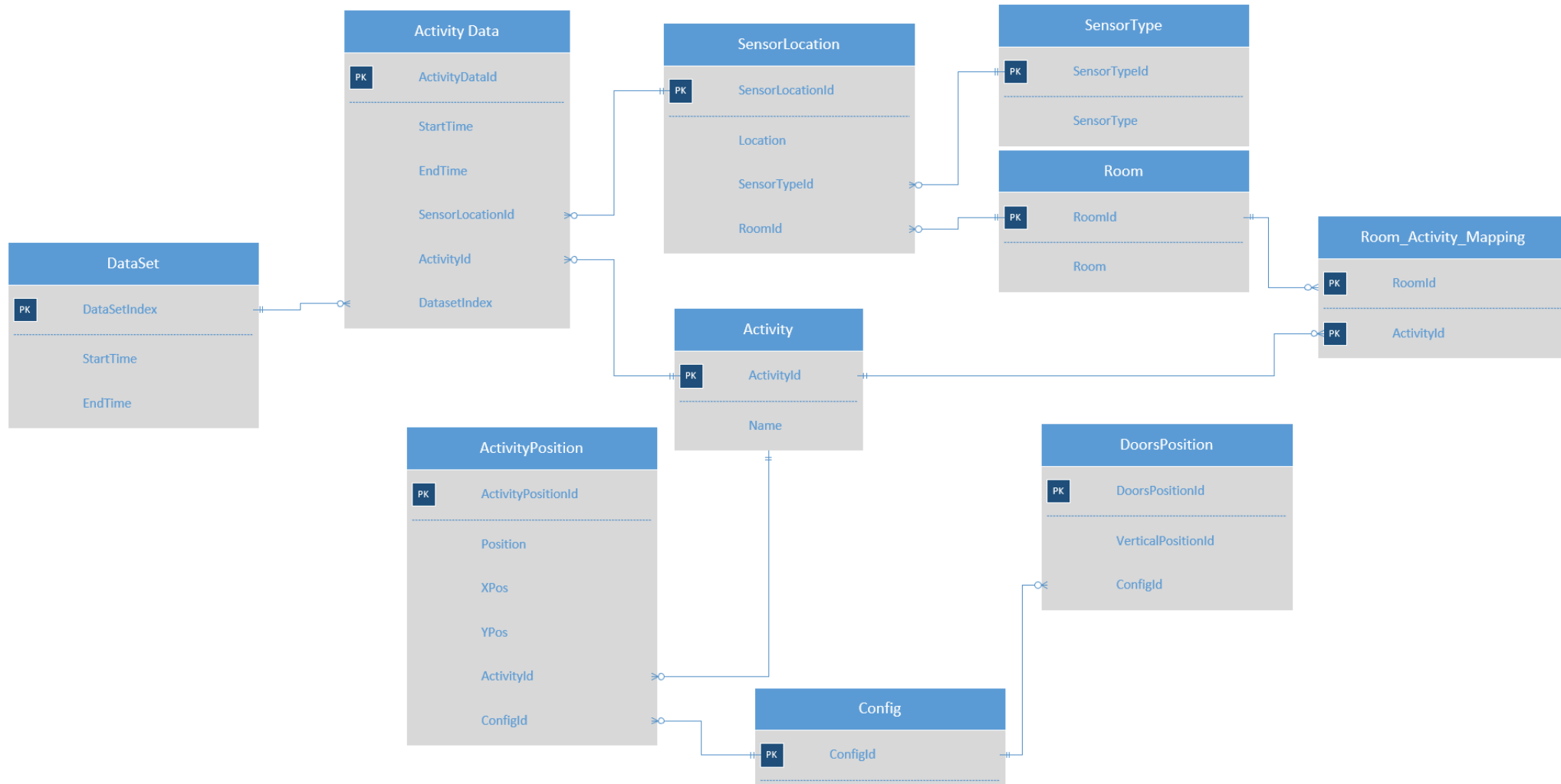
Appendices



Appendices



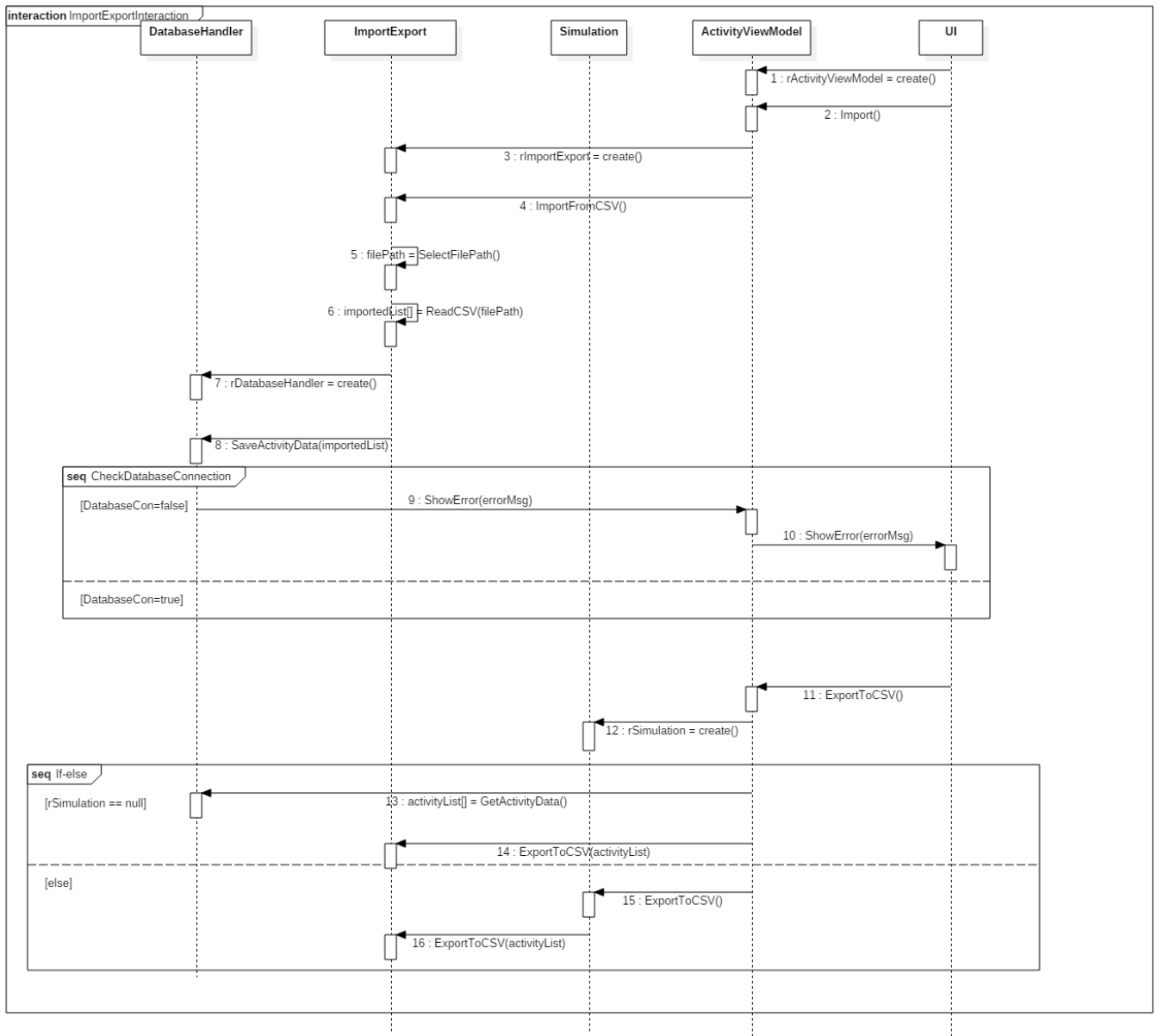
Appendix C: Database Logical Structure



Appendix D: Import/Export FDUCD

Use Case Section	Comment
<i>Use Case Name</i>	Import/Export
<i>Scope</i>	Activity Simulator
<i>Level</i>	User goal
<i>Primary actor</i>	User
<i>Stakeholders and Interests</i>	
<i>Preconditions</i>	
<i>Success guarantee</i>	Importing dataset from file and exporting dataset to file
<i>Main success scenario</i>	<ol style="list-style-type: none"> 1: Import-button is pressed 2: Open file explorer 3: Store user specified csv-file in list 4: Store list in database 5: Export button is pressed 6: Get activity list from simulator 7: Write list to csv-file
<i>Extensions</i>	<p>4a: Problem connecting to database. Show error message.</p> <p>6a: Simulator not running. Import from database instead.</p>
<i>Special requirements</i>	Connection to Operational Database
<i>Technology list</i>	
<i>Frequency of occurrence</i>	Import/Export buttons are pressed
<i>Miscellaneous</i>	

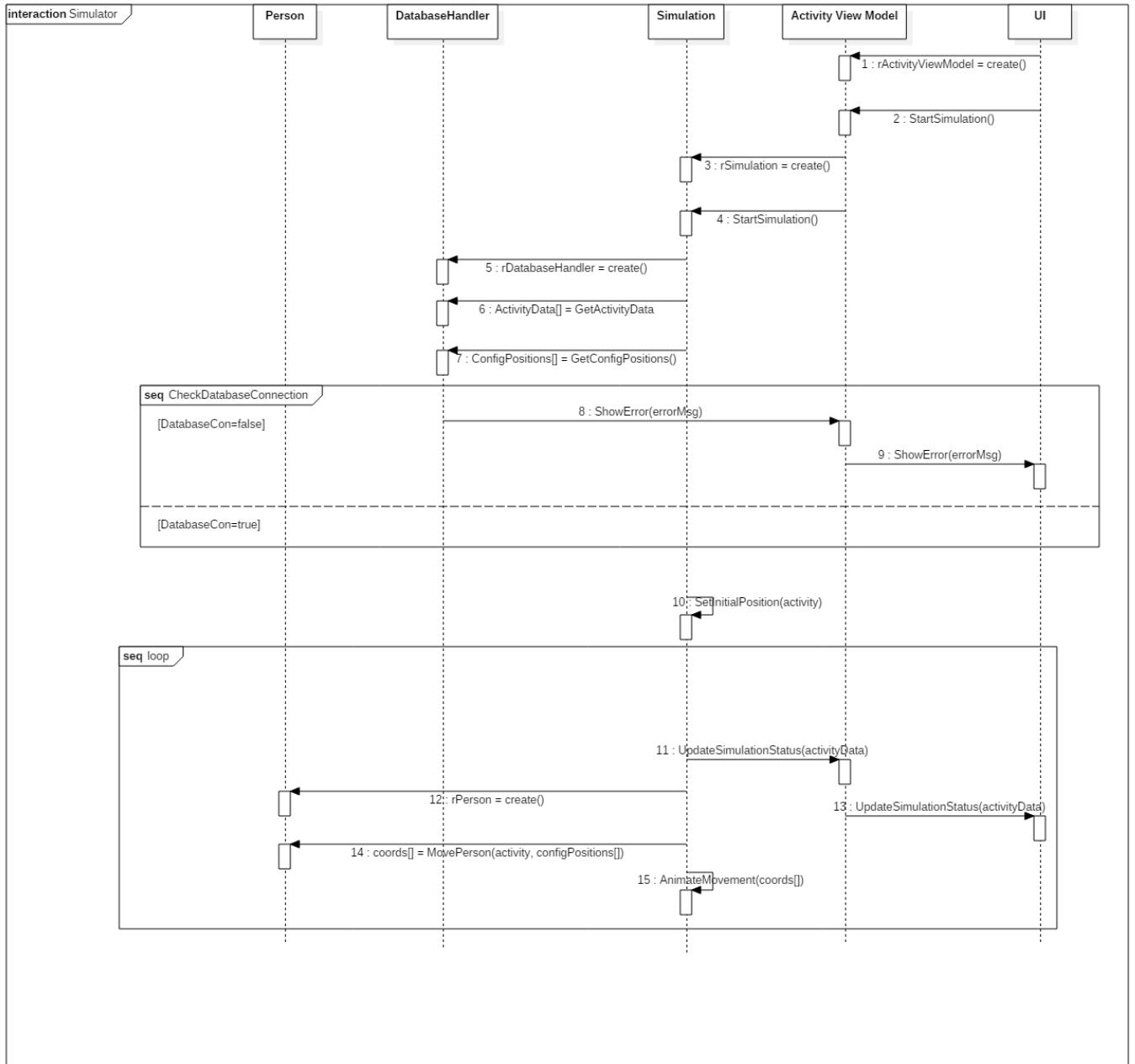
Appendix E: Import/Export Interaction Diagram



Appendix F: Simulate Activity Data FDUCD

Use Case Section	Comment
<i>Use Case Name</i>	Simulate activity data
<i>Scope</i>	Activity Simulator
<i>Level</i>	User goal
<i>Primary actor</i>	User
<i>Stakeholders and Interests</i>	
<i>Preconditions</i>	
<i>Success guarantee</i>	Simulating activity dataset
<i>Main success scenario</i>	<ol style="list-style-type: none"> 1: Get data from database and fill list 2: Place person at initial position 3: Update status textboxes 4: Move person to new location when activity changes 5: Go to 3
<i>Extensions</i>	1a: Problem importing from database. Show error message.
<i>Special requirements</i>	Operational database
<i>Technology list</i>	
<i>Frequency of occurrence</i>	When the “Start Simulation”-button is pressed
<i>Miscellaneous</i>	

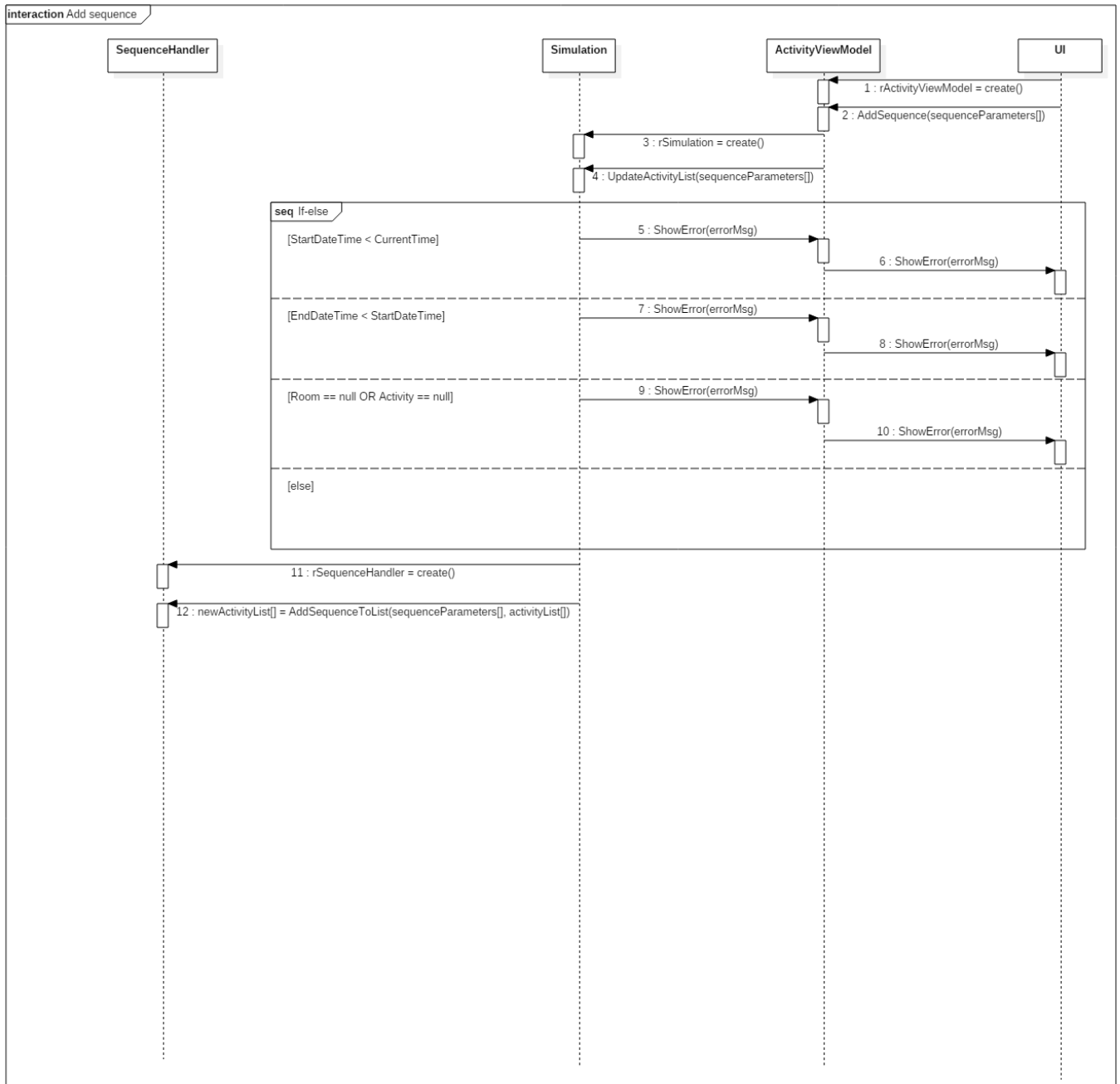
Appendix G: Simulate activity data Interaction Diagram



Appendix H: Add sequence of activities FDUCD

Use Case Section	Comment
<i>Use Case Name</i>	Add sequence of activities
<i>Scope</i>	Activity Simulator
<i>Level</i>	User goal
<i>Primary actor</i>	User
<i>Stakeholders and Interests</i>	
<i>Preconditions</i>	
<i>Success guarantee</i>	Adding activity to the simulator list
<i>Main success scenario</i>	<ol style="list-style-type: none"> 1: Read values from sequence text boxes 2: Add values to activity list 3: Update list used by the simulator
<i>Extensions</i>	<ol style="list-style-type: none"> 1a: Start date and time is earlier than current time. Show error message. 1b: End date and time is earlier than start date and time. Show error message. 1c: Room or activity is not specified. Show error message.
<i>Special requirements</i>	Simulator running
<i>Technology list</i>	
<i>Frequency of occurrence</i>	Every time the “Confirm Selection”-button is pressed
<i>Miscellaneous</i>	

Appendix I: Add sequence of activities Interaction Diagram

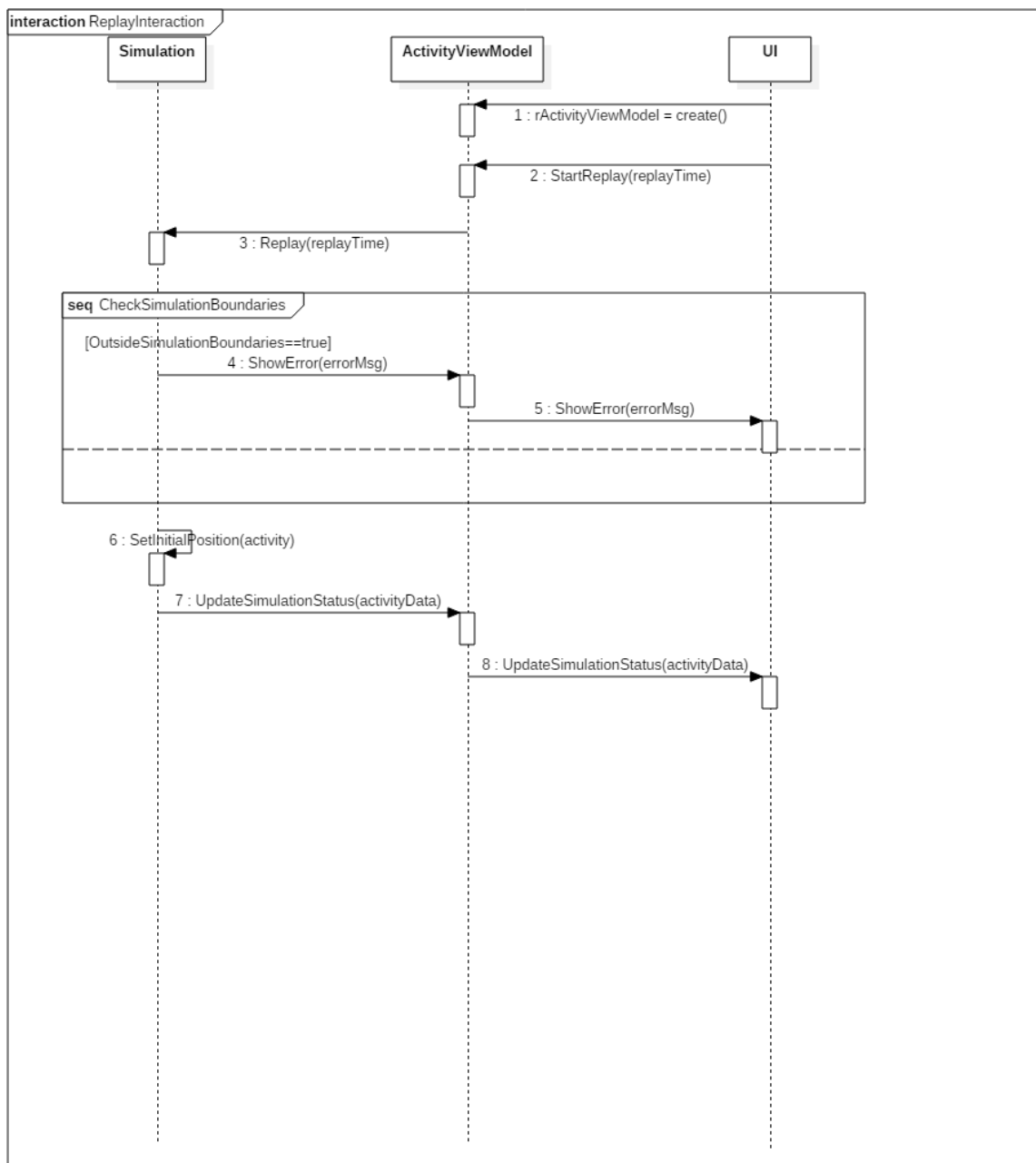


Appendix J: Replay previous set of activities

FDUCD

Use Case Section	Comment
<i>Use Case Name</i>	Replay previous set of activities
<i>Scope</i>	Activity Simulator
<i>Level</i>	User goal
<i>Primary actor</i>	User
<i>Stakeholders and Interests</i>	
<i>Preconditions</i>	
<i>Success guarantee</i>	Rewinding the Simulation
<i>Main success scenario</i>	<ol style="list-style-type: none"> 1: Read replay time textbox 2: Read activity list with current index 3: Subtract replay time from current time 4: Update simulation status
<i>Extensions</i>	1a: Replay time outside of simulation boundaries. Show error message.
<i>Special requirements</i>	Simulator running
<i>Technology list</i>	
<i>Frequency of occurrence</i>	Every time replay button is pressed
<i>Miscellaneous</i>	

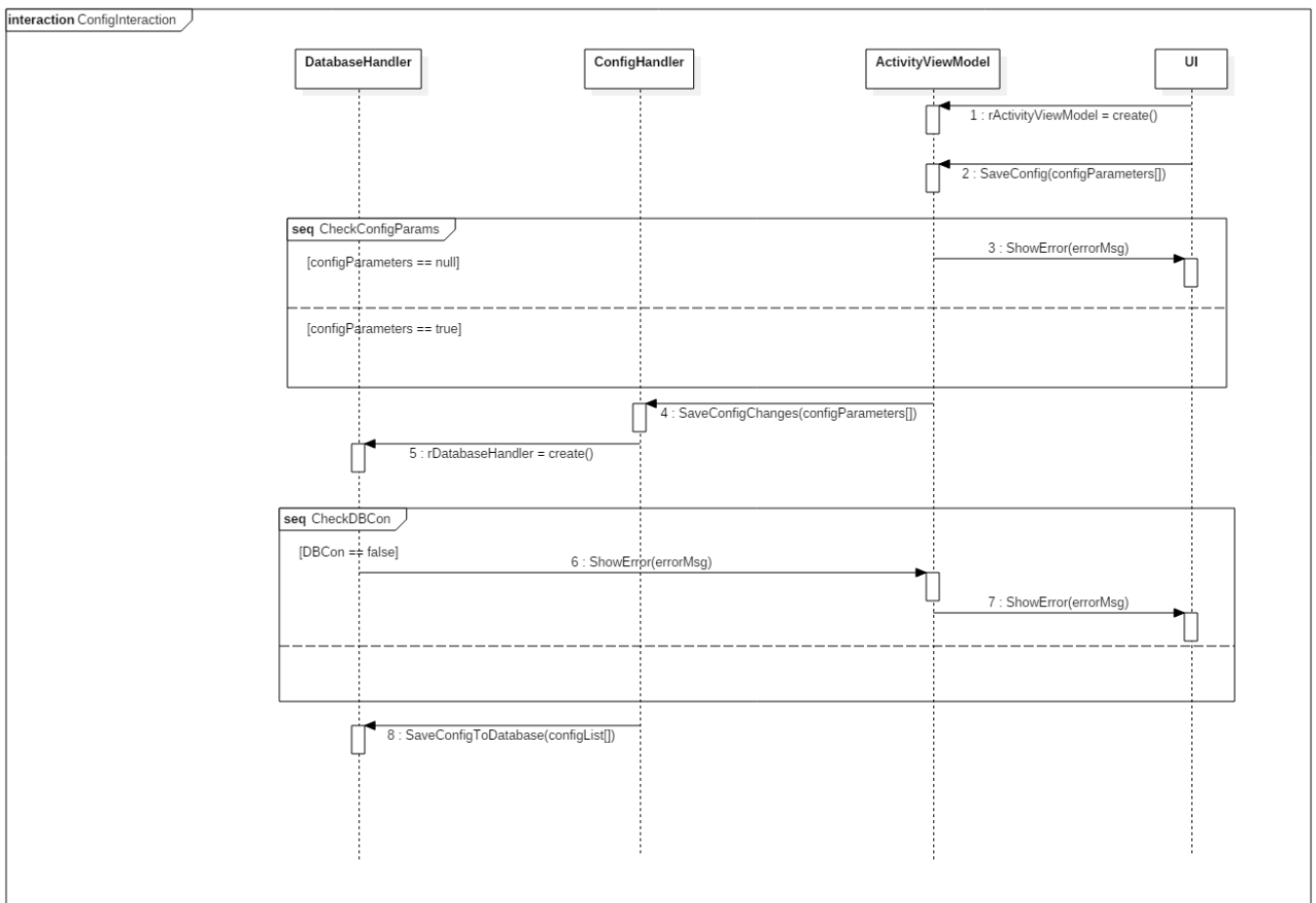
Appendix K: Replay previous set of activities Interaction Diagram



Appendix L: Configure properties and parameters FDUCD

Use Case Section	Comment
<i>Use Case Name</i>	Configure properties and parameters
<i>Scope</i>	Activity Simulator
<i>Level</i>	User goal
<i>Primary actor</i>	User
<i>Stakeholders and Interests</i>	
<i>Preconditions</i>	
<i>Success guarantee</i>	Saving properties and parameters to the database
<i>Main success scenario</i>	1: Read configuration parameters from textboxes 2: Save parameters to list 3: Save list to database
<i>Extensions</i>	1a: Not all textboxes contain a value. Show error message 3a: Problem saving list to database. Show error message
<i>Special requirements</i>	Connection to Operational database
<i>Technology list</i>	
<i>Frequency of occurrence</i>	Every time Save Config button is pressed
<i>Miscellaneous</i>	

Appendix M: Configure properties and parameters Interaction Diagram



Appendix N: Import/Export Test Case – Main

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 1

Module Name: Import/Export Main Success Scenario

Test Title: Importing data to the database and
exporting data to a csv file

Description: Test the import and export functions

Test Designed by: Stian Glittum

Test Designed date: 11.04.18

Test Executed by: Stian Glittum

Test Execution date: 11.04.18

Pre-conditions: Connection with Operational Database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Pressing of “Import” button		Opening of file dialog box	File dialog box opens	Pass	
2	Selection of a valid csv file	NewDataset.csv	New activity dataset in the database	A new dataset appears in the database	Pass	
3	Pressing of “Export” button		Opening of file dialog box	File dialog box opens	Pass	
4	Selection of folder		New csv file appearing in the selected folder	A new csv file appears in the selected folder	Pass	

Post-conditions:

A new activity dataset appears in the database when importing. A new csv file appears when exporting.

Appendix O: Import/Export Test Case – Extensions

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 2

Module Name: Import/Export Extensions

Test Title: Importing data to the database and
exporting data to a csv file: Extensions

Description: Test the import and export extensions

Test Designed by: Stian Glittum

Test Designed date: 11.04.18

Test Executed by: Stian Glittum

Test Execution date: 11.04.18

Pre-conditions: Connection with Operational Database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Turning off database and importing a file	NewDataset.csv	Appearance of an error message	An error message appears, indicating no connection to the database	Pass	
2	Pressing "Export" before turning on the simulator		Export of data directly from the database	The csv file created contains data directly from the database	Pass	

Post-conditions:

Appearance of an error message, signifying that the importing failed. New csv file with data from the database.

Appendix P: Simulate Activity Data Test Case – Main

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 3

Module Name: Simulate Activity Data

Test Title: Simulate activity data from a database

Description: Test the simulation functions

Test Designed by: Stian Glittum

Test Designed date: 12.04.18

Test Executed by: Stian Glittum

Test Execution date: 12.04.18

Pre-conditions: Connection with Operational Database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Pressing of “Start Simulation” button	Activity Dataset	Simulation status parameters updates, person appears at activity location	Simulation status parameters updates, person appears at activity location	Pass	
2	Waiting until activity changes	Activity Dataset	Simulation status parameters updates, person moves to next activity location	Simulation status parameters updates, person moves to next activity location	Pass	
3	Pressing of pause button	Activity Dataset	Simulation pauses	Simulation pauses	Pass	
4	Pressing of resume button	Activity Dataset	Simulation resumes	Simulation resumes	Pass	
5	Sliding of fast-forward slider	Activity Dataset	Simulation speed increases	Simulation speed increases	Pass	
6	Pressing of “Next Activity” button	Activity Dataset	Simulator jumps to the next activity, moving the person	Simulator jumps to the next activity, moving the person	Pass	
7	Toggling “Fast Mode”	Activity Dataset	Simulator jumps to next activity every five seconds	Simulator jumps to next activity every five seconds	Pass	

Post-conditions:

Continuously simulating the activities of a person, all functions related to the simulator reacts appropriately when interacted with.

Appendix Q: Simulate Activity Data Test Case – Extensions

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 4

Module Name: Simulate Activity Data Extensions

Test Title: Simulate activity data from a database: Extensions

Description: Test the simulation extensions

Test Designed by: Stian Glittum

Test Designed date: 12.04.18

Test Executed by: Stian Glittum

Test Execution date: 12.04.18

Pre-conditions: Connection with Operational Database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Turning off database and starting simulator	Activity Data	Appearance of an error message	An error message appears, indicating no connection to the database	Pass	

Post-conditions:
 Appearance of an error message, signifying that the simulator failed to get the activity data.

Appendix R: Add Sequence of Activities Test Case – Main

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 5

Module Name: Add Sequence of Activities

Test Title: Adding activity sequences to simulator list

Description: Test the sequence functions

Test Designed by: Stian Glittum

Test Designed date: 13.04.18

Test Executed by: Stian Glittum

Test Execution date: 13.04.18

Pre-conditions: Simulation running

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Entering sequence parameters and pressing “Confirm Selection”	Activity Dataset	New sequence adds to the simulator list	New sequence added to the simulator list, “Next activity” and “Next Start Time” textboxes indicates the success	Pass	
2	Pressing “Randomize” button	Activity Dataset	Generates a set of random valid sequence parameters	Random values appear in the text boxes	Pass	

Post-conditions:

New simulator list includes the newly added sequence.

Appendix S: Add Sequence of Activities Test Case – Extensions

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 6

Test Designed by: Stian Glittum

Module Name: Add Sequence of Activities Extensions

Test Designed date: 13.04.18

Test Title: Adding activity sequences to simulator list: Extensions

Test Executed by: Stian Glittum

Description: Test the sequence function extensions

Test Execution date: 13.04.18

Pre-conditions: Simulation running

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Setting new start time as earlier than current simulation time	Activity Dataset	Error message	Error message specifying the new start time must be later than the current time	Pass	
2	Setting new end time as earlier than new start time	Activity Dataset	Error message	Error message specifying the new end time must be later than the new start time	Pass	
3	Leaving new room and new activity empty	Activity Dataset	Error message	Error message specifying that new room and new activity must have values	Pass	

Post-conditions:
 No new sequence is added.

Appendix T: Replay Previous Set of Activities Test Case – Main

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 7

Module Name: Replay Previous set of Activities

Test Title: Replaying to a previous simulation time

Description: Test the replay function

Test Designed by: Stian Glittum

Test Designed date: 14.04.18

Test Executed by: Stian Glittum

Test Execution date: 14.04.18

Pre-conditions: Simulation running

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Pressing “Replay” button after specifying replay time	Activity Dataset	Simulation rewinding to a previous time specified by the replay time	Simulation rewinds to a previous time. Activity and person position changes as the previous simulation time also includes a different activity	Pass	
2	Pressing “Previous Activity” button	Activity Dataset	Rewinding the simulation to a time where the activity is different	Simulation rewinds to a time where the activity is different	Pass	

Post-conditions:

Simulation time is rewinded. Simulation status textboxes updates. New person position if different activity.

Appendix U: Replay Previous Set of Activities Test Case – Extensions

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 8

Test Designed by: Stian Glittum

Module Name: Replay Previous set of Activities Extensions

Test Designed date: 14.04.18

Test Title: Replaying to a previous simulation time: Extensions

Test Executed by: Stian Glittum

Description: Test the replay extensions

Test Execution date: 14.04.18

Pre-conditions: Simulation running

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Specifying a replay time that rewinds the simulation time to be outside the boundaries	Activity Dataset	Error message	Error message that indicates the target simulation time is outside of simulation boundaries appears	Pass	

Post-conditions:
Simulation time is not rewinded.

Appendix V: Configure Properties and Parameters Test Case – Main

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 9

Test Designed by: Stian Glittum

Module Name: Configure Properties and Parameters

Test Designed date: 15.04.18

Test Title: Saving configurations to database

Test Executed by: Stian Glittum

Description: Testing the possibility of saving configurations to database

Test Execution date: 15.04.18

Pre-conditions: Connection to database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Toggling “Activate Editing” checkbox, focusing a textbox and clicking on the drawing area		Coordinates of mouse to appear in the text boxes at click	Coordinates appear in the text boxes at the click of the mouse	Pass	
2	Pressing “Save Configuration” once all coordinates textboxes are filled		Database tables to be updated with the new coordinates	New coordinates are saved in the database	Pass	

Post-conditions:
Coordinates are saved to the database.

Appendix W: Configure Properties and Parameters Test Case – Extensions

Project Name:

Developing an activity simulator
of a person living in a smart house

Test Case

Test Case ID: 10

Test Designed by: Stian Glittum

Module Name: Configure Properties and Parameters Extensions **Test Designed date:** 15.04.18

Test Title: Saving configurations to database: Extensions **Test Executed by:** Stian Glittum

Description: Test the configuration extensions **Test Execution date:** 15.04.18

Pre-conditions: Connection to database

Dependencies:

Appendices

Step	Test Steps	Test Data	Expected Result	Actual Result	Status (Pass/Fail)	Notes
1	Leaving one or more of the coordinate textboxes empty		Appearance of error message	An error message appears indicating that all the textboxes must have values	Pass	
2	Turning off database and pressing "Save Configuration"		Appearance of error message	An error message appears, indicating no connection to the database	Pass	

Post-conditions:
 No new coordinates are saved to the database.

