

# A Smart Home Simulation Tool

*The Development of a Simulation Tool for Measuring  
the Impact of a Smart Grid on a Private Home*

Michael Nysteen & Henrik Mynderup

Kongens Lyngby 2012

IMM-M.Sc.-2012-117

## A Smart Home Simulation Tool

### The Development of a Simulation Tool for Measuring the Impact of a Smart Grid on a Private Home

#### Author(s):

Henrik Mynderup, s062373

Michael Nysteen, s062413

#### Supervisor(s):

Bjarne Poulsen (Associate Professor, DTU Informatics)

Department of Informatics and Mathematical Modelling

Technical University of Denmark

Asmussens Allé, building 305

DK-2800 Kgs. Lyngby, Denmark

[www.imm.dtu.dk](http://www.imm.dtu.dk)

Phone: (+45) 45 25 33 51

Fax: (+45) 45 88 26 73

E-mail: [reception@imm.dtu.dk](mailto:reception@imm.dtu.dk)

Release date: 10<sup>th</sup> September 2012

Class: 1 (public)

Edition: 1<sup>st</sup> edition

Comments: This report is a part of the requirements to achieve Master of Science in Engineering (MSc) at Technical University of Denmark.

The report represents 30 ECTS points.

Rights: © Henrik Mynderup, Michael Nysteen, 2012

# Abstract

---

The power grid today is highly affected by the variations in the use of energy during a day which peaks during the afternoon and evenings. Providing the energy required during peak hours is already a challenge and will become increasingly difficult with the increasing amount of unpredictable energy production from renewable energy sources. Therefore it is necessary to be able to shift the energy consumption to match the energy being produced in any given period. One way to control the energy consumption of private homes is to use energy prices that vary during the day based on the energy available. In order to motivate people to shift their consumption, initiatives must be taken to make them aware of which effects it has on their home.

This thesis focuses on how software simulation tools can be used to make consumers aware of these effects. Existing simulation tools of smart homes are examined and it is found that very little work has been done in the area of simulating the effects of varying energy prices in private homes. Therefore this thesis develops a prototype of a simulation tool that enables the users to create accurate models of their homes and simulate how the energy consumption can be scheduled based on energy prices and power generation from attached distributed energy resources such as wind turbines and solar panels.

The key focus areas of the developed simulation tool are *usability* to make it user-friendly and attractive to use for the consumer; *customizability* to make it easy to create accurate personalized models; and *extensibility* to let device manufacturers easily add their energy consuming devices to the application and to let future developers easily add new functionality.

Case studies of two real households are used to examine the accuracy of the simulation and to show how the application can be used to present the effects of using intelligent scheduling of the energy consumption in the home.

# Resumé

---

El-nettet i dag er stærkt påvirket af variationer i brugen af energi i løbet af en dag, der topper i løbet af eftermiddagen og aftenen. Det er allerede en udfordring at levere den nødvendige energi under spidsbelastning og dette vil blive stadig vanskeligere med den stigende mængde af uforudsigelige energiproduktion fra vedvarende energikilder. Det er derfor nødvendigt at kunne forskyde energiforbruget så det svarer til energien, der produceres i et givet tidsrum. En måde at styre energiforbruget i private hjem på er at anvende energipriser, der varierer i løbet af dagen baseret på den tilgængelige energi. For at motivere folk til at flytte deres forbrug, skal der tages initiativer til at gøre dem opmærksomme på, hvilke effekter det har på deres hjem.

Denne afhandling fokuserer på, hvordan software-simuleringsværktøjer kan bruges til at gøre forbrugerne opmærksomme på disse effekter. Eksisterende simuleringsværktøjer for intelligente hjem er undersøgt, og det viser sig, at meget lidt arbejde er blevet gjort indenfor simulering af varierende energiprisers virkninger i private hjem. Derfor vil denne afhandling udvikle en prototype af et simuleringsværktøj, der giver brugerne mulighed for at skabe præcise modeller af deres hjem og simulere, hvordan energiforbruget kan planlægges ud fra energipriser og el-produktionen fra tilsluttede distribuerede energiressourcer såsom vindmøller og solfangere.

De vigtigste fokusområder i det udviklede simuleringsværktøj er at være *brugervenligt* for at gøre det attraktivt at anvende for forbrugeren; *brugerdefinerbart* for at gøre det nemt at oprette præcise personlige modeller, og *udvidelsesklart* for at lade fabrikanter af udstyr nemt tilføje deres energiforbrugende apparater til programmet og at lade fremtidige udviklere nemt tilføje ny funktionalitet.

Case studies af to reelle husholdninger bruges til at undersøge nøjagtigheden af simuleringen og vise, hvordan programmet kan bruges til at præsentere konsekvenserne af brugen af intelligent planlægning af energiforbruget i hjemmet.



# Preface

---

This thesis has been prepared at the Department of Informatics and Mathematical Modelling at the Technical University of Denmark in the period from February 2012 to September 2012. The thesis is a part of the requirement for acquiring the Master of Science degree in Computer Science and Engineering.

The thesis deals with how to develop a simulation tool that can simulate the effects of using variable energy prices along with intelligent scheduling of energy consumption in a smart home in a smart grid setup. A significant part of the thesis was to design and implement a prototype of such a simulation tool.

A paper is being prepared based on the thesis and will be submitted to *The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies* that takes place in Lisbon, Portugal on the 24<sup>th</sup> – 29<sup>th</sup> of March 2013. A preliminary version of the paper is provided in Appendix J of this thesis.

We would like to thank Associate Professor Bjarne Poulsen who has guided us in the development process of this thesis and provided us with contacts and material that have formed the basis of the analysis of the smart grid and smart home concepts. Additionally we would like to thank Associate Professor Esben Larsen who has provided us with insight into the electrical aspects of the energy consumptions of home appliances and general considerations of the smart grid. Finally we would like to thank the Danish developers of intelligent energy management systems, GreenWave Reality, for taking the time to meet with us and provide insight into their approach to handling energy management in homes and for providing valuable feedback of the developed prototype of the simulation tool.

The work carried out in this thesis has been divided equally between the two authors and each section of the written thesis is composed of equal contributions from both authors. Consequently, both authors are responsible for all sections of the written thesis.

Michael Nysteen

Henrik Mynderup

Lyngby, September 2012

# Table of Contents

---

Abstract .....	i
Resumé .....	ii
Preface .....	iii
1. Introduction.....	1
1.1. Background and Motivation .....	1
1.1.1. The Existing Power Grid.....	2
1.1.2. The Smart Grid.....	4
1.1.3. Smart Home.....	6
1.1.4. Home Area Network.....	8
1.2. Vision .....	9
1.3. Thesis Definition .....	9
1.4. Scope .....	10
1.5. Development Methodology .....	10
1.6. Report structure .....	12
1.7. Conclusion .....	13
2. Analysis.....	15
2.1. State of the Art Simulation Tools .....	15
2.1.1. EnergyPlus .....	15
2.1.2. Academic Approach to Home Automation Systems .....	15
2.1.3. PowerMatcher Simulation Tool.....	16
2.1.4. GridLab-D.....	17
2.1.5. Status of the Current Solutions .....	17
2.2. Home Area Network.....	18
2.2.1. A Part of Smart Grid.....	18
2.2.2. Smart Meters.....	19
2.2.3. Types of Home Area Networks.....	21
2.2.4. HAN Communication Technologies.....	22
2.2.5. Controllers for Home Automation.....	23

2.3.	Consumer Benefits .....	27
2.4.	The Societal Impact of the Smart Home.....	28
2.5.	Conclusion .....	29
3.	Requirements .....	31
3.1.	Target Audience.....	31
3.1.1.	The Consumer.....	31
3.1.2.	The Advisor/Salesman .....	32
3.1.3.	The Educator/Educatee .....	33
3.1.4.	Comparison of Potential Target Audiences.....	33
3.2.	Main Purposes of the Program.....	34
3.3.	Domain Model .....	35
3.4.	Modeling Components .....	36
3.5.	Realism in Simulations.....	37
3.6.	Non-Functional Requirements .....	38
3.6.1.	Customizability .....	38
3.6.2.	Extensibility.....	39
3.6.3.	Report-Oriented Result .....	39
3.6.4.	Performance .....	39
3.6.5.	Usability .....	39
3.7.	Functional Requirements .....	42
3.8.	Behavioral Requirements (Use Cases).....	44
3.9.	Graphical User Interface Mock-ups.....	45
3.9.1.	Main Window .....	45
3.9.2.	Navigation (Ribbons) .....	47
3.9.3.	Reporting Window.....	50
3.9.4.	Settings Window.....	51
3.10.	Conclusion .....	52
4.	Design .....	53
4.1.	Overall Design.....	53
4.1.1.	Architecture.....	53
4.1.2.	Design Patterns.....	54
4.1.3.	Frameworks and Toolkits .....	56
4.1.4.	Modules.....	58

4.1.5.	Interfaces and Abstract Classes .....	61
4.2.	Detailed Design.....	62
4.2.1.	General Functionality .....	62
4.2.2.	Modelling.....	64
4.2.3.	Data Handling .....	74
4.2.4.	Simulation.....	77
4.2.5.	Data Visualization .....	79
4.3.	Conclusion .....	82
5.	Implementation and Presentation .....	85
5.1.	General .....	85
5.1.1.	General Implementation .....	86
5.2.	Modelling.....	93
5.2.1.	Toolbox and Components.....	93
5.2.2.	Canvas.....	95
5.2.3.	Undo/Redo .....	99
5.3.	The Summary Area .....	101
5.3.1.	Implementation of the Summary Module.....	102
5.4.	Input Data.....	103
5.4.1.	Implementation of Input Data.....	104
5.5.	Simulation.....	105
5.5.1.	Implementation of the Simulation .....	106
5.6.	Reporting.....	114
5.6.1.	Implementation of the Reporting Module .....	118
5.7.	Projects.....	122
5.8.	Usability Adherence.....	124
5.9.	Conclusion .....	126
6.	Test .....	129
6.1.	Unit Tests.....	129
6.1.1.	Sorting Performance.....	130
6.2.	Manual Tests .....	131
6.2.1.	GUI Tests.....	131
6.2.2.	Chart Performance .....	131
6.2.3.	Simulation Tests.....	132

6.3.	Case Studies.....	135
6.3.1.	Case Study: House .....	136
6.3.2.	Case Study: Apartment.....	138
6.4.	Demonstration and Feedback .....	139
6.5.	Conclusion .....	140
7.	Conclusion .....	141
7.1.	Conclusion on the Thesis Goals .....	141
7.2.	Overall Conclusion.....	142
8.	Future Work.....	145
	References.....	149
	Appendix A: Glossary.....	A
	Appendix B: HAN Communication Protocols.....	B
	Appendix C: Use Case Diagram.....	C
	Appendix D: Detailed Use Case Descriptions .....	D
	Appendix E: ISO 9241-171 Conformity .....	E
	Appendix F: Unit Test Results .....	F
	Appendix G: Case Studies – House .....	G
	Appendix H: Case Studies – Apartment.....	H
	Appendix I: Installation Guide .....	I
	Appendix J: Paper of the Thesis.....	J



# 1. Introduction

---

This chapter provides an introduction to the thesis by first looking at the motivation and background for the thesis, then it describes the vision for the project and lastly it presents the thesis definition. The thesis definition lists a number of specific goals that should be fulfilled by this thesis. After the thesis definition has been presented the report structure will be described and the scope of this thesis will be defined.

A glossary explaining the terms used in the thesis is provided in Appendix A and a guide to how to install the simulation tool developed in this thesis is provided in Appendix I.

## 1.1. Background and Motivation

The current power grid is a great engineering masterpiece that started with Thomas Edison who in 1882 opened the first power plant. From then on the development of the big power grid known today went fast, but the information technology has since then overtaken the grid and is now providing a lot of possibilities that society is not using efficiently. In the not so distant future, energy will be in short supply as more and more electrical devices are developed. This is intensified by an increasing number of countries developing a high need for energy. Additionally, today's main sources of energy such as coal, gas and oil are finite and will eventually be depleted. Therefore it is essential that energy can be used more intelligently and that an increased amount of renewable energy is incorporated into the grid. A challenge with relying on renewable energy resources is that the time the energy is generated (and the amount) is unpredictable as they rely on weather conditions. Therefore it will be highly useful to be able to shift the energy consumption such that it matches the time of the energy production. A solution to this could be an upgraded version of the power grid that incorporates the consumer and motivates them to shift their energy consumption and thereby be an active part in the environmental challenges that will be faced in the future. This concept is called *the smart grid*.

This area has attracted a lot of attention from businesses all over the world that seek to develop technologies and devices that can help the implementation of intelligence into the power grids. This ranges from huge well-established companies such as Microsoft [1: Microsoft], Belkin [2: Belkin] and Google [3: Google] to new companies founded with this specific area in mind such as GreenWave Reality [4: GreenWave Reality] in Denmark.

Informing consumers about the economic benefits of shifting their energy usage and generating their own energy inside their home will encourage the consumer to participate in a decentralization of the power grid and thereby increase the efficiency at which society uses energy. This thesis will describe the development of a simulation tool that can help inform the consumers of the effects of intelligent energy consumption in their homes in a smart grid scenario and thereby help the transition to smart grid as described by *Smart Grid Netværkets* main suggestions to how to improve this transition [5: Klima- Energi- og Bygningsministeriet] (see 1.2 Vision).

### 1.1.1. The Existing Power Grid

In order to understand the concepts of *smart grid* and *smart homes* that are essential to this thesis one must first understand how the power grid works today. The power grid in Denmark currently consists of a number of elements and actors. It can be divided into two parts: The power network and the power market. Each of these parts will be described in the coming sections.

#### 1.1.1.1. The Power Network

In the power network the producers are responsible for producing the power in power plants and wind turbines etc. This power is transferred through the transmission network over longer distances and through the distribution network the final distance to the consumer. The transmission network also allows for sending or receiving power to and from neighboring countries, i.e. Norway, Sweden, Germany and The Netherlands. In Denmark the transmission network is owned and managed by Energinet.dk and the distribution network is owned by the network companies. *DONG Energi City Elnet A/S* and *NRGI Net A/S* are examples of such network companies.

#### 1.1.1.2. The Power Market

The generated power is sold by the producers in the Nordic wholesale power market, NordPool. Here the power trading companies buy the power in bulk and sell it to the consumers (see Figure 1). Examples of power trading companies (also known as power providers) are *Dong Energy City Forsyning A/S* and *NRGI Elsalg A/S*, not to be confused with the above mentioned network companies.

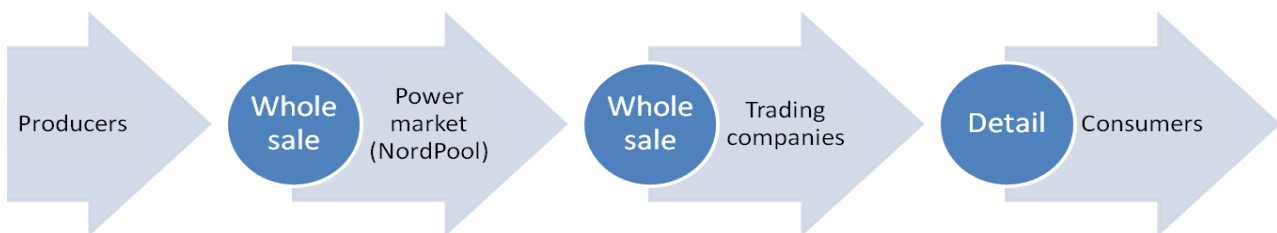


Figure 1: An illustration of the existing power grid in Denmark.

In Denmark the consumer has a right to choose his own power provider. If he does not actively choose one he will receive power from the power provider that has the *providing duty* in the region in which the consumer lives. The providing duty is a contract provided by the central authorities to a power provider in a region requiring them to provide power to consumers that have not chosen another provider. Research by the Danish Competition Authority in 2008 showed that 94% of the consumers get power from the provider with the providing duty in the region [6: Konkurrencestyrelsen].

The consumer can be billed for their energy use in one of two ways: Time-of-use in which you pay a price for each kWh depending on when you use it (also called varying prices) or Fixed-rate in which you pay a fixed price for every kWh (also called flat-rate prices). Consumers that use the provider's providing duty automatically gets flat-rate pricing and therefore this is by far the most common billing form. Additionally time-of-use billing is charged an extra fee each year, making it less viable for private homes. Time-of-use billing, however, is a prerequisite for using the power prices to control energy consumption patterns in a smart grid.



The power generated in Denmark today is very dependent on fossil fuels. As seen in Figure 2 the power is mostly generated by centralized plants which are typically powered by coal, oil or gas. In 2011 around 29% of the power came from renewable energy resources which are almost exclusively massive wind turbines.

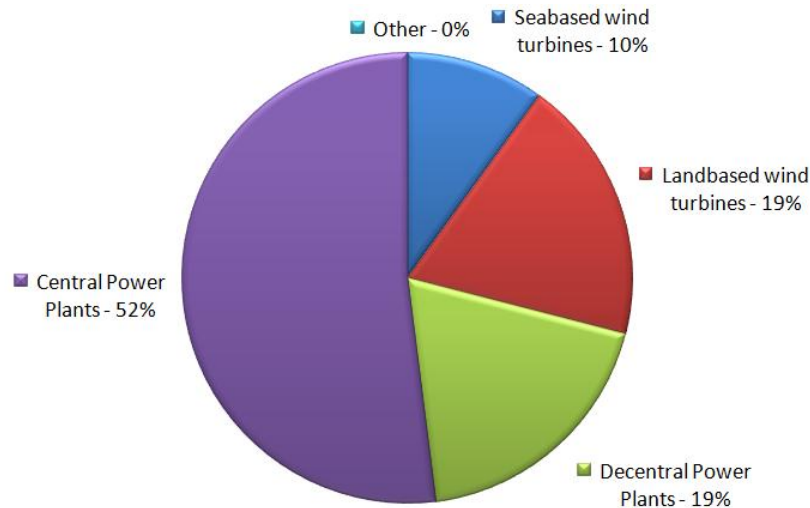


Figure 2: The sources of the power generated in Denmark in 2010. Data source: [7: Energinet.dk]

The government, however, aims to let 50% of the country's power supply come from wind turbines by 2020 [8: Klima- Energi- og Bygningsministeriet] and by 2050 Denmark should be independent of fossil fuels [9: Schaldemose]. This presents a problem as it is not possible to control when the wind turbines are generating power or how much they are generating as this depends on the weather. There are a few techniques for storing power for later use, such as flywheels, but these techniques are not very efficient and therefore the power should ideally be used when it is generated. Additionally, the power supply and demand should be balanced at all times since excess power will cause an overload in the network which results in blackouts, and similarly too little power obviously results in blackouts as well.

Balancing the supply and demand is not a huge problem when most of the power comes from consistent, centralized sources such as coal-driven power plants as you can just (although expensive) burn some more coal or oil in times of need, but the more power the grid get from renewable energy sources the harder it becomes to balance it [10: Ricci, Vinerba].

#### 1.1.1.3. Daily Peaks in Energy Demand

The power grid today is highly affected by variations in the use of energy during a day. The energy use during the night is very low, but the use of energy increases significantly during a couple of hours in the afternoon and evening (called peaks hours) when people come home from work. This is because people turn on the lights, TVs, computers, start cooking etc. which all consume energy (see Figure 3).

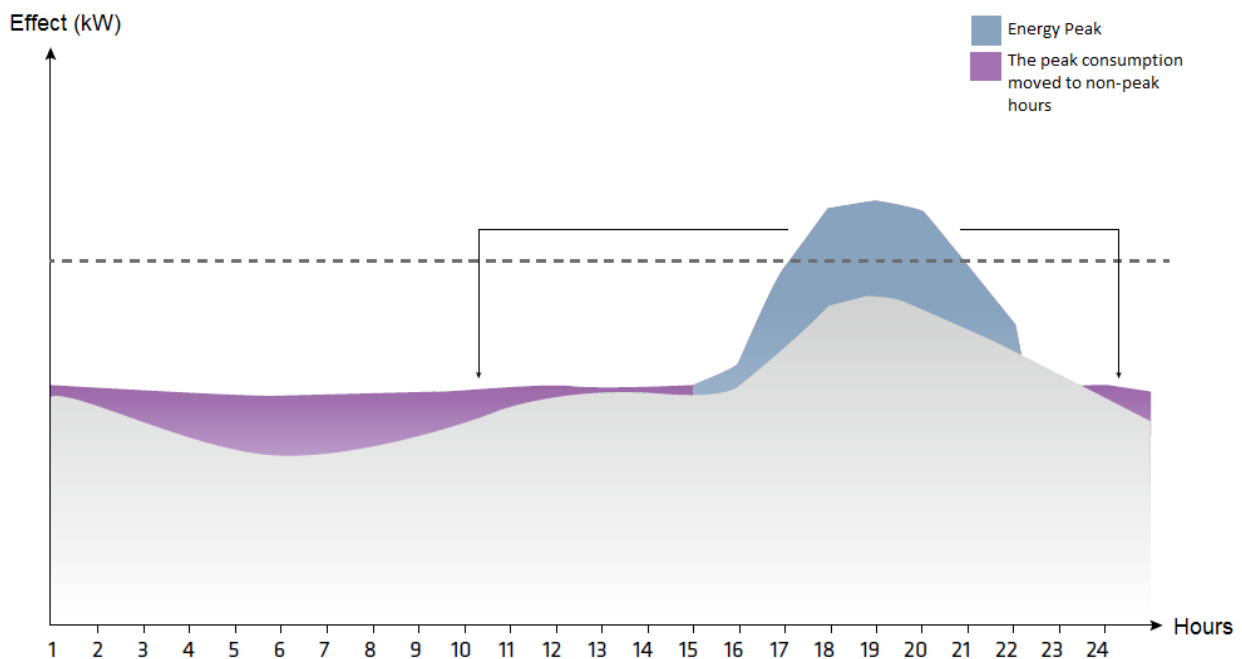


Figure 3: Energy use during the day displaying the afternoon peak hours and how the consumption can be spread out to less intense periods. Source: [5: Klima- Energi- og Bygningsministeriet]

Today the energy demand in these peaks is met by powering up so called *peaker plants* that can be started rapidly when more power is required [11: Haas]. These typically run on oil or natural gas and are very expensive to run and produce a significant amount of pollution [12: Illinois Environmental Protection Agency]. They are, however, needed as energy companies cannot rely solely on energy from renewable energy resources during peak hours as it may be that there is no wind for the wind turbines or sun for the solar cells etc. Another issue with renewable energy resources is that much of the power generated by renewable energy resources will be wasted if generated outside of the peak hours as it cannot be stored efficiently which means that it is difficult to get the most out of them.

As an increasing amount of the energy production comes from renewable energy resources the above-mentioned challenges become increasingly important. In order to solve these issues a more intelligent power grid is needed: The Smart Grid.

### 1.1.2. The Smart Grid

Adding intelligence to the existing power grid will make it possible to monitor the energy use and thereby know how and when the energy will be used. By having a two way communication between all parts of the power grid it will be possible to control the energy generation and consumption more efficiently. This way of handling and using the energy in a smarter way is referred to as a “*smart grid*”.

As mentioned in the previous chapter the Danish government has decided that in 2020 50% of the power production has to come from wind turbines. With a smart grid you can overcome the issue of inconsistent power generation from renewable energy sources through *peak shaving* which is the process of moving consumption from a peak period to a non-peak period (see Figure 3). The peak shaving can be done in various ways, but the manner in which it is achieved is irrelevant to the definition of a smart grid. One

popular way of moving the consumption, however, is using price regulations to motivate people to change their consumption habits.

As seen in Figure 4 each actor in the smart grid is connected to a network. Information about generated and consumed power is collected and based on this the total generation or consumption can be regulated by e.g. changing the prices for the energy accordingly in order to motivate people to use more or less energy at the given time.



Figure 4: An illustration of the overall structure of the smart grid [13: Electric Power Research Institute]

The existing power grid has developed from being centralized with big power plants, to being distributed on many of smaller power production plants. The vision of the smart grid is to develop a completely distributed production grid where the consumers can also contribute. By installing Distributed Energy Resources (DER) on their property, consumers can produce power for themselves or sell it to the grid if they produce more than they need in certain periods. The smart grid will allow users to send surplus power into the global power grid. This way the consumer can help with leveling out the production of power from non-renewable energy sources and lowering their own power bill. If a consumer wishes to sell energy directly at the Nordic power market they must produce a substantial amount of energy each year which is beyond the capabilities of a single household. Therefore they should participate in a Virtual Power Plant (VPP). A VPP is a grouping of consumers that have DERs installed on their property, and in the eyes of the power grid the VPP works as a single power plant, but with all the generated power comes from the consumers. Additionally, because the VPP produces more energy than one private person can the VPP can sell the power at a higher price on the power market than one single consumer could. An alternative to VPPs is *Microgrids*. A microgrid is, like a VPP, a group of entities containing DERs, but where VPPs focus on providing combined power from the DERs to the power market the microgrid focuses on creating a self-sustainable unit in the smart grid that, in the eyes of the smart grid, is one unit [14: Xu and Wu]. While microgrids is an interesting aspect of the smart grid and tightly connected to smart homes this thesis focuses on smart homes as isolated units and will therefore not consider microgrids .

### 1.1.3. Smart Home

A *smart home* is a term used about homes that contain a computer device that can control the energy consumption in the home [15: Oxford Dictionaries]. Additionally, smart homes that are connected to a smart grid can adjust the consumption based on external parameters such as the load of the grid [16: Kok, Karnouskos].

Each smart home will be connected to the smart grid through a *smart meter*. The smart meter is basically an intelligent electricity meter that receives the current energy prices from the smart grid at regular intervals (see 2.2.2 *Smart Meters*). It also keeps track of how much power is being used in the house and how much is being generated by the house's Distributed Energy Resources (DERs) and this information is sent back into the grid to the utility companies. The smart meter can also communicate the varying energy prices to other devices such as electrical appliances or a controller unit to allow them to determine when it is most beneficial to start.

In 2009 the European Parliament settled on an agreement about the installment of smart meters in private homes. They agreed that 80% of the homes in EU should have a smart meter installed in 2020 if it was economically responsible [6: Konkurrencestyrelsen]. This created a deadline for the completion of standards and settlements between the actors involved in the smart grid on the European market.

#### 1.1.3.1. Distributed Energy Resources (DERs)

An important aspect in the smart home is the DERs. They allow the consumer to generate part of their power supply themselves or sell excess power to the grid. DERs come in many varieties. A few examples will be examined here.

##### 1.1.3.1.1. Wind turbines

A number of wind turbines have been created in sizes that are suitable for private homes. These wind turbines typically have wattage of less than 10 KW. The smaller ones have wattage of less than 1 KW and can be mounted on roof tops. These can typically be bought for less than €2500. Larger wind turbines can be placed on poles in fields, but are mostly suitable for rural areas where they are not obstructed by other objects. These come in the price range of €18,500 to €29,000 [17: Energy Saving Trust].

##### 1.1.3.1.2. Photovoltaics (Solar Panels)

Photovoltaics is the process of converting solar energy to direct current. The most common application of this technique is solar cells grouped into larger solar panels. The direct current produced when the sun's rays hit the silicon and the photons are knocked loose needs to be transformed into alternating current using an inverter. In this process a significant amount of the energy is lost resulting in effects less than 100%. Modern solar panels can convert about 12-18% of the solar energy to electrical power, but some experimental solar panels have displayed an efficiency of just over 40% [18: Wesoff]. It currently takes an average of 10 years for a solar panel to repay its price, but as the efficiency goes up with improved technology the time will decrease [19: BBC News].

##### 1.1.3.1.3. Micro-Combined Heat and Power unit ( $\mu$ CHP)

Micro-combined heat and power units ( $\mu$ CHP) are home-sized versions of the larger CHP systems used in power plants providing up to 50 kW. They are typically generating heat as their main product and electricity

is generated as a by-product. This allows the consumer to get “free” power when using a CHP to heat his house.  $\mu$ CHPs can be fuelled in a number of ways including biogas, solar thermal and natural gas.

### **1.1.3.2. Batteries**

When the DERs produce more power than required at the given time the power can either be fed back into the power grid or it can be stored in a local ion-battery. Such batteries can be stored in the home and be used for peak shaving by letting the house consume the energy stored in the battery during peak hours.

#### **1.1.3.2.1. Electric Vehicles (EVs)**

Since electric vehicles (EV) contain a rechargeable ion-battery it can be used to store energy and perform peak leveling as described above. When more energy is required in the smart grid the EV can deliver energy from its ion-battery to the grid. Then when the peak hours are over and there is a surplus of energy available the EV can recharge its battery thereby helping to level out the peaks. In the EV infrastructure made by BetterPlace in Denmark the EV charging is not connected to the individual house’s power lines and therefore the management of the energy in the battery is separated from the home [20: BetterPlace]. This means that the home cannot use the energy from the car, nor will they get money for the energy supplied to the network. Instead the owner pays a fixed fee each month that allows him to drive a certain distance each year and BetterPlace then administers the required power for the car.

### **1.1.3.3. The Effect of the Current Energy Pricings on Smart Homes**

A financial effect is likely to be achieved in smart homes if all energy sources and appliances are controlled based on dynamic prices [21: Flaim and Levy]. This especially includes electrical-powered heating as such heating is responsible for a significant amount of the energy consumption and it is relatively flexible in terms of when to use energy. It is already possible to control electrical heating based on dynamic prices if the consumers use electrical heating [22: Control4]. Similarly electrical cars can be incorporated in the home to charge them when it is most beneficial, but in order to make EVs and electrical heating viable for the consumers in Denmark the taxes on electricity have to be reduced. According to *Dansk Energi* the high taxes on electricity compared to gas and oil are holding back the development of smart homes [23: Dansk Energi].

Compared to the countries in the rest of the European Union, Denmark has by far the highest total price on electricity, but interestingly there were in 2009 13 countries with a higher raw electricity price than Denmark before taxes [24: Det Energipolitiske Udvalg]. As described by The Danish Political Energy Committee [24: Det Energipolitiske Udvalg] electricity prices were in 2009 in Denmark 0.922 DKK/KWh before taxes and 1,988 DKK/KWh after taxes. This shows that more than half of the price is actually taxes. Power companies are restricted in how much profit they are allowed to make on selling energy to consumers, but the public is not and in 2012 over 0.8 DKK/KWh was public taxes [25: Go Energi]. Based on the 2012 prices from *Elpristavlen* of 2.34 DKK/KWh [26: Elpristavlen] this is over 34% of the total price. In addition to that are the tariffs to the power and distribution companies which results in over half of the price being taxes. The taxes do not scale with the raw electricity price which means that using varying prices as the situation is at the moment will not have much of economic effect as over half of the price will remain the same at all times.

As a consequence of this the public taxes on electricity must be reduced or changed to scale with the raw energy price at the given moment if consumers are to be encouraged to switch to electrical heating, EVs etc. and thereby increase the benefits of smart homes [5: Klima- Energi- og Bygningsministeriet].

#### **1.1.3.4. Flexibility**

As mentioned earlier the smart grid relies on moving the energy consumption to periods where a lot of energy is available. Smart homes can do this to different extents based on the components available in the home. Homes with DERs, washing machines, heat pumps, EVs etc. have a significant potential energy consumption that can be moved without the consumer noticing it, while other homes such as many apartments without washing machines and who typically have gas-driven heating have less potential to do so. This potential to move energy consumption based on the requirements of the smart grid is often referred to as the *flexibility* of the home. This flexibility is by some people considered a potential way of charging customers of their energy consumption such that the higher the home's flexibility is the lower energy price they have to pay [GreenWave Reality, personal communication, 2012]. This would motivate people to increase their home's flexibility and thereby be more beneficial for the smart grid.

#### **1.1.4. Home Area Network**

A Home Area Network (HAN) is a Local Area Network within a household. It uses a wired or wireless communication between the devices connected to it. HAN is also the backbone of a smart home as it allows the appliances within the household to communicate with each other and with the smart meter. A lot of research into standards for communication within a HAN is currently being conducted, but this will be examined in detail in the Analysis chapter in section 2.2.4.

Two main HAN solutions are being tested in different countries to find out which would be more beneficial to the consumer and product manufacturers. The first approach is a peer-to-peer solution where the network is constructed by a number of connected self-controlled units that communicate directly with the smart meter and then decide individually what to do. The second solution is a gateway based solution where the appliances are controlled by a *controller unit* that communicates with the smart meter and then tells the appliances what to do and when to do it.

The devices connected to a HAN can be any electric devices such as PCs, Dishwashers and Washing machines etc. To be able to use a home appliance on a HAN it will need to be smart grid ready, which means that the appliance needs to have a remote control interface that allows for remotely adjusting and/or starting and stopping the device.

##### **1.1.4.1. Controller Units**

The controller unit's job is to monitor the energy prices received from the smart meter, the usage tendencies and the consumer defined choices. Based on these variables it must then decide when the appliance needs to run or when it should turn down the energy consumption. A controller unit can be placed in three different places; in a HAN gateway within the house, as a cloud based service on the internet or inside each of the home appliances. For more information about controller units see section 2.2.5.

## 1.2. Vision

Smart homes and smart grid, as described in the previous sections, are currently hot topics in Denmark and the rest of the World. In October 2011 the Danish workgroup, *Smart Grid Netværket*, proposed nine main suggestions to be implemented to smooth the transition to smart grid. Suggestion 7 is to increase the knowledge of the smart grid technology with the consumers [5: Klima- Energi- og Bygningsministeriet]. The vision of this thesis is to create a tool that can help people realizing the effects of integrating a private home with smart grid technologies and thereby the thesis fits well into *Smart Grid Netværkets* main suggestion 7. It should, in a simple way, give the user tools to compare different home setups with and without a smart grid and thereby seeing the differences between them. Additionally the tool can be used to compare various *configurations* of the home which can help in the process of purchasing new components to the house.

## 1.3. Thesis Definition

This thesis will describe the creation of a simulation tool that examines the economic impact of a smart grid on the private household when using intelligent control of the energy consumption based on varying energy prices. In order to examine this, the simulator allows the user to create a virtual household. The user will be able to add electric appliances (washing machines, dishwashers, refrigerators etc.) as well as local energy resources (such as wind turbines, solar cells etc.) to the household and see how this affects the power bill.

In order to simulate this, the application will use environmental variables which includes weather information (affecting power generated by wind turbines, solar cells etc.) and the energy prices from external energy sources (power plants) that will be used to schedule when the energy should ideally be consumed. These variables will be importable from real sources as well as manually configurable. The application should support the creation of multiple scenarios, i.e. configurations of a home, and allow for comparisons of these. The scenarios can also be executed without intelligent scheduling and thereby allowing the user to compare the scenarios with and without smart-grid scheduling enabled.

The possible outputs of the simulation for each scenario could be a price for the total energy consumption with the given setup, a graph of the energy consumption over time in the given time interval and a schedule of when each electrical appliance was used. These results can be compared for each scenario, allowing the user to identify how the parameters that have changed between the scenarios affect the home.

The thesis will utilize the UML design principles to design the simulation tool. This will result in a number of UML deliverables that combined will describe the intended structure and behavior of the application. The simulation tool will be implemented as a stand-alone application using C#.NET, more specifically the Windows Presentation Foundation (WPF).

The main goals of thesis and thereby also the learning objectives are as follows:

1. Present the required background knowledge about the existing power grid, the Smart Grid, the power market, Virtual Power Plants (VPP) and Distributed Energy Resources (DER)
2. Analyze the concepts of a smart meter and Home-Area-Network (HAN)



3. Examine the existing approaches for communication within the smart home
4. Survey existing state-of-the-art controller units and projects for home automation
5. Survey existing state-of-the-art simulation solutions
6. Present the target audience of the application
7. Identify aspects of a private home that are relevant for the simulation
8. Identify functional and non-functional requirements for the simulation tool
9. Find an appropriate design architecture for the system
10. Design the simulation tool with extensibility in mind
11. Adhere to chosen standards of user interface design
12. Implement a prototype of the designed simulation tool as a proof of concept
13. Use existing design patterns to guide a best practice implementation of the tool
14. Create case studies that show how the application can be used to discover the effects of a smart grid on a private home

## 1.4. Scope

As this thesis aims to create a prototype rather than a full version of a simulation tool there are certain things that will not be included. First of all the tool will focus on creating an easily extendable framework that can use many types of energy consuming devices rather than focusing on extremely accurate simulations of the individual device. Such implementations can be added later. Additionally, the tool will be using historical data rather than forecasts for both weather and energy prices as these are readily available. Price forecasts only exist one day ahead and weather forecasts are very difficult to get in a usable format for free. Ideally the simulator should work in any country, but this prototype will only be using data from Danish power providers and weather services. There is no common standard for how the weather data and power data being integrated is provided by the sources, but the prototype will still attempt to allow developers to integrate any power and weather data source in the future by extending the existing application, but without having to modify the existing code.

The tool will focus on the financial effects on the home and it will not simulate the effects on the national power grid. The prices of the energy are, however, expected to be at their lowest when increased energy consumption is most beneficial for the grid and therefore the cheapest use of energy for the consumer should also be the most beneficial use of energy for the grid as a whole. Additionally the tool assumes that the varying energy prices (time-of-use pricing) will be commonly used in the future as this is a fundamental prerequisite for consumer's financial gain in smart grids and therefore for this tool.

The tool will not examine heating from non-energy consuming components and it will not examine heat production from components that are not dedicated heating units such as computers and TVs.

Lastly, the simulation tool will not be simulating the effect of inserting sensors that turn off the light etc. based on human behavior. It will only simulate the effects the varying energy prices have on the home.

## 1.5. Development Methodology

Developing software can be done using a variety of different development methods, each with their own strengths and weaknesses. A few of these development methods are the linear sequential model also known as the waterfall model, prototyping, incremental development, spiral development, scrum and agile



development [27: Software Development Methodologies]. These approaches are not mutually exclusive and some of the mentioned methods will include aspects of the other methods such as scrum being both iterative, incremental and an agile development method [28: Schwaber and Sutherland].

This thesis will be using an incremental development approach in which the waterfall model is combined with iterative prototyping. The analysis, requirements and overall design will be performed using the waterfall approach such that each step is finished before the next is undertaken. This way a full list of required features and use cases can be developed before the design is begun and thereby help guiding the design. Completing the overall design before beginning on the detailed design ensures that there is a solid foundation on which the rest of the application can be built. This includes all the basic design decisions such as which frameworks and design patterns to use as well as creating the overall structure of the application. The use cases developed in the requirements chapter will be grouped into batches based on importance and each batch will be the foundation of the steps to follow. The rest of the development, i.e. the detailed design, implementation and testing will be performed in an iterative process. Each iteration will design the details of the functionalities in a batch, implement the functionalities and test them resulting in a prototype containing the batch of functionalities as well as the ones from previous iterations. Iterations will be performed until all functionality has been implemented or the deadline is reached.

By doing it this way the most important functionality (and thereby the functionality that provides most value for the application) is ensured to be implemented and tested before less important features are implemented. This ensures the best use of the time, maximizing the value gained from the time spent. Since this is a prototype it is likely that the final result will not contain all features that were originally planned and this development method ensures that it is the least important features that are left out. The development methodology used can be seen in Figure 5.

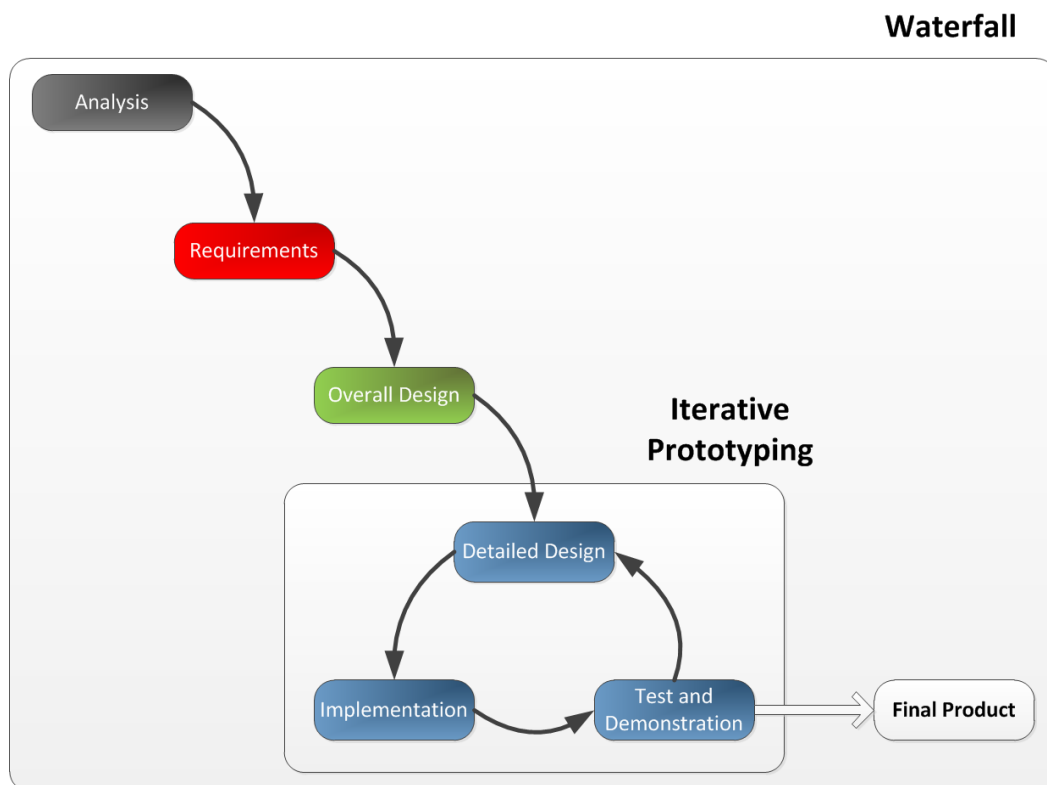


Figure 5: The incremental development methodology used in this thesis.

In order to ensure that the time is managed properly an overall project plan has been created that describes the phases to be worked through and when each phase should be finished. The project plan can be seen in Figure 6. The plan will be used to ensure that all tasks in the development of the thesis are given an appropriate amount of time and to ensure that multiple iterations will be performed by setting a deadline for each task.

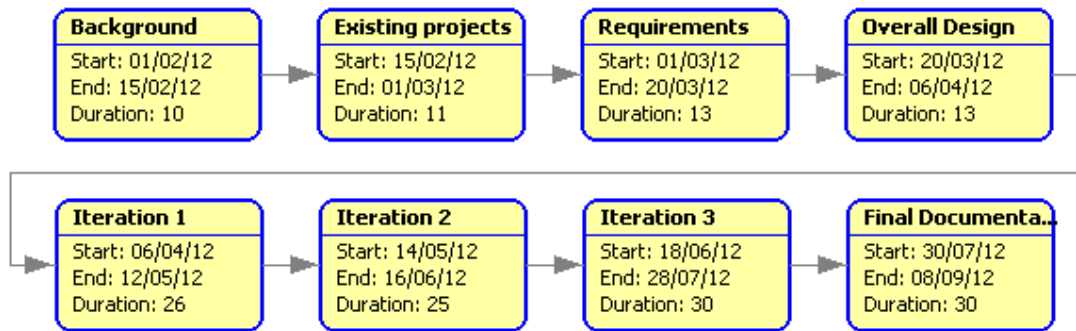


Figure 6: Project plan used in the development of this thesis.

## 1.6. Report structure

This section outlines the structure of the report by describing the scope and content of the remaining chapters. For each chapter the covered thesis goals (defined in 1.3 *Thesis Definition*) are listed. Chapter 1 covered the thesis goals 1 and partially goal 2.

### Chapter 2 – Analysis

In this chapter an analysis of some of the important aspects of the smart home is performed. First the chapter looks a current state-of-the-art simulation tools. Then it describes the core concepts of the Home-Area-Network (HAN) by describing the role of the HAN, outlining the smart meter concept, introducing different versions of HANs and presenting and comparing the communication protocols and controllers within a HAN.

This chapter covers goals 2, 3, 4 and 5.

### Chapter 3 – Requirements

This chapter describes the requirements for the application that are identified by first defining the target audience and their preferences. Based on this the purpose of the program is defined which is then used to identify a set of functional and non-functional requirements. A domain model is presented that shows the relationships between the concepts of the domain and use cases are defined that identify all the scenarios that the application should handle. Finally a set of graphical user interface mockups are shown based on the identified requirements that provide guide lines for the graphical development of the application.

This chapter covers goals 6, 7, 8 and partly 11.

### Chapter 4 – Design

Based on the requirements defined in chapter 3, this chapter presents the important design decisions for the simulation tool. A number of important decisions are addressed such as the choice of a suitable architecture and which design patterns to be use based on the purpose of the application. This chapter also

defines which technologies to use when implementing the simulation tool. Finally the chapter defines which modules the application is built of and how they are allowed to communicate with each other.

This chapter covers goals 9, 10 and 13.

### **Chapter 5 – Implementation and Presentation**

This chapter presents the implemented application and describes some of the most crucial implementation details. The chapter goes through each section of the application and introduces the features and how they are implemented.

This chapter covers goals 11 and 12.

### **Chapter 6 – Test**

In this chapter the test methodology is described and each type of testing that has been performed is documented with results. Additionally, case studies that have been performed to show how the application can be used to discover the effects of the smart grid on a private home are presented.

This chapter covers goal 14.

### **Chapter 7 – Conclusion**

This chapter sums up the results of the thesis. It describes how each thesis goal has been covered in the report and finally it sums up all the findings and conclusions made throughout the report and provides an overall conclusion of what has been achieved.

### **Chapter 8 – Future Work**

Future work in relation to extending the application will be introduced and described and can be used as a starting point for future studies and development of the simulation tool.

## **1.7. Conclusion**

The key motivating factor of this thesis is to inform consumers of the effects of the smart grid and motivate them to take actions to enable better energy consumption in their homes. There is a lot of development and research being done in the area of smart grids and smart homes, but the success of the smart grid relies highly on the participation of the consumers. Therefore a tool that can present the effects of the smart home can be a key factor in the future enrollment and acceptance of the smart grid.

This chapter has provided the necessary background information about this subject and based on this the definition for this thesis has been conducted. A number of learning objectives were defined and the project scope was presented. An incremental development approach for developing the simulation tool has been chosen that consists of a mix of a waterfall and iterative prototyping methodology, which suits the thesis scope perfectly.



## 2. Analysis

---

This chapter is going in depth with the core concepts of a smart home. First the existing simulation tools will be analyzed in order to see what functionality currently exists and what is needed of a new simulation tool. Secondly, a thorough description of the Home Area Network (HAN) will be presented to explain how components communicate within a smart home. Finally, some of the existing controller units that exist for smart homes will be examined to see if the choice of controller unit is likely to affect the result of the energy consumption and therefore be relevant for a simulation tool.

### 2.1. State of the Art Simulation Tools

A number of simulation tools have been proposed to simulate the energy consumption in a home or building. Some of these have been proposed in academic papers [29: Conte, Scaradozzi] while others are commercially created tools such as EnergyPlus [30: EnergyPlus] and PowerMatcher [31: PowerMatcher Smartgrid Technology] and others still are tools developed by an open source community such as GridLab-D [32: GridLAB-D]. Most of these do not simulate the scheduling of devices based on varying prices through the day and do therefore not show the effects of a smart grid, but may still prove useful to this thesis as they describe different approaches to simulating the energy consumption. This section will therefore analyze the tools and evaluate their strengths and weaknesses with a focus on how this thesis can improve the field.

#### 2.1.1. EnergyPlus

EnergyPlus is a simulation engine that simulates the heating and cooling loads required to keep a building at a specific temperature. It is designed for architects and building engineers to ensure that their buildings are thermally efficient before they are build. The calculations can take a number of parameters into consideration such as weather forecasts, heat conduction through the walls (and windows), blinds in front of the windows etc., and it can also simulate the atmospheric pollution produced by heating/cooling of the building. EnergyPlus does not provide any GUI, but instead uses a number of input/output files in pure ASCII text [30: EnergyPlus] .

While heating simulation is not the focus of this thesis, it is a relevant aspect to consider. The more energy required heating the house, the less energy is available for selling to the market (or the more energy it is necessary to buy) and thereby increasing the energy bill for the household. It is therefore useful to know the heating requirements for the home in this simulation. Using the EnergyPlus engine could be a possible way of obtaining such information, but will not be implemented in the current prototype due to it being outside of the scope due to time restrictions.

#### 2.1.2. Academic Approach to Home Automation Systems

[29: Conte, Scaradozzi] from Università Politecnica delle Marche in Italy have proposed an approach to scheduling the use of home appliances based on limited resources. Rather than considering at which point

in time the energy is cheapest they instead consider how to use the devices such that a threshold of available energy at any point in time is not exceeded. While this is not the goal of this thesis their approach is interesting and should be considered.

They use the Multi Agent Theory (MAS) to model the scenario using each appliance as a separate agent. The agents communicate through variables on a global stack that allows the agents to run on separate computers and real devices can be plugged in.

While this thesis does not directly operate with limited resources it may be necessary to consider it, as running all the appliances at once may result in unnecessary load on the network and possibly blowing a fuse. Additionally the concept of describing each appliance as a separate agent is a design pattern worth considering as it is an effective way of increasing scalability and emulating the actual behavior of a *decentralized* control system (see 2.2 Home Area Network).

### 2.1.3. PowerMatcher Simulation Tool

PowerMatcher Smartgrid Technology is a company that develops units for controlling the energy consumption in a building in a smart grid. They have created a simulation tool [31: PowerMatcher Smartgrid Technology] that serves as a demonstration of their technology. Therefore the simulation tool is highly similar to the actual technology, so examining the technology used is in order. The PowerMatcher technology is based on the concepts of Multi Agent Systems and microeconomics. Each device in the building (DER or appliance) is managed by a *device agent*. These agents attempt to optimize the use of the device by buying and selling electricity to the *auctioneer agent* based on the economic principles of microeconomics. The auctioneer agent broadcasts prices at a given interval to the devices and adjust these prices to attempt to reach equilibrium between energy consumption and production. This behavior can be changed by inserting an *objective agent* that buys and sells from the auctioneer agent at a given rate in order to either increase production or decrease production. An example of the use of such an agent is if the system should function as a virtual power plant and therefore has to generate more power than it is consuming [33: PowerMatcher Smartgrid Technology].

The simulation tool mimics the above mentioned behavior and allows the user to configure the agents as well as setting prices and simulation speeds. The simulation tool allows multiple households to be connected to the same auctioneer agent which allows the user to mimic a VPP. In addition to the already described agents the simulator also contains agents for wind parks, diesel generators and other energy producing units. These can be connected to the auctioneer agents to act as the source of the energy prices.

When the simulation is run graphs can be produced showing the results over time. This includes the prices from the auctioneer, the demand of each unit over time and the generation of each unit over time.

PowerMatcher Simulation Tool is the tool most closely resembling the objective of this thesis. PowerMatcher does not provide a strong graphical modeling framework for creating the scenarios, but instead relies on graphically adding nodes to a tree-structure. The simulation tool only supports adding DERs as well as a total demand-node representing the entire energy demand of all appliances in the household and does therefore not provide any way of examining the consumption over time for different components except for the DERs.

## 2.1.4. GridLab-D

GridLab-D is another command-line based simulation engine. It is developed by an open source community and is meant for a wide audience including utility engineers, regulators, various stakeholders, and consumers [32: GridLAB-D]. The purpose of the engine is to simulate any aspect of the distribution net including aspects such as energy markets, DERs and home appliances. The user can model the scenarios in an agent-based manner like in the PowerMatcher Simulation Tool. Each class is defined in a module (a .dll file) and specific objects of these classes are used to define the actual scenarios. The simulation lets each agent (device) adapt over time based on the other objects until an equilibrium is reached and at this point in time the simulation is stopped. Equilibrium is sought for in the network to ensure that the production and consumption are matched at all times.

GridLab-D is intended as a simulation engine that provides data to third-party analysis tools and therefore the inputs and outputs are handled only through xml-based text files. GridLab-D covers a huge part of the smart grid network, each part being represented by a module in the modeling part. One of these modules is the *residential module* that covers a private home. This is the part that is relevant for this thesis. It appears, however, to only examine thermal loads and even though they support the addition of a number of appliances it seems to be in the context of thermal load only (heat generation) [34: GridLAB-D].

GridLab-D therefore does not appear to support calculations of energy generation and consumption in a household or scheduling of appliances based on varying prices which is the main focus of this thesis. It is, however, a possible engine to use for thermal generation (like EnergyPlus) as it is less straining on the CPU, but also less advanced algorithms. This will, however, not be implemented in this prototype.

## 2.1.5. Status of the Current Solutions

The existing solutions vary in their focus as well as implementations. Table 1 compares the solutions based on six areas.

Simulation Tool	Focus	Target Audience	Type	Interaction	Scope	Scheduling
EnergyPlus	Find thermal load	Engineers	Engine (No GUI)	Input / output text files	A physical building	None
Conte et. al.	Energy load balancing w. limited resources	Academic circles	Several programs	Settings files. Can plug in real devices	A household	Agents based on load threshold
PowerMatcher	Energy generation / consumption balancing	Demonstration of their technology	Stand-alone (GUI)	Graphical. Can plug-in external agents	One or more combined households	Agents based on equilibrium
GridLab-D	Thermal load Balancing to minimize consumption	Engineers	Engine (No GUI)	Input / output text files	Any	Agents based on thermal load

Table 1: Comparison of the current simulation tools

By examining the focus and target audience of the current simulation tools in Table 1 it can be seen that the current tools focus on professionals as target audience rather than the end consumer. This also means that the consumer's interests are not taken into account as they could be. This again is seen in the scope

and scheduling categories where none of the tools examine economic benefits or attempt to personalize the modeling experience for the consumer by letting them model their own home.

Additionally the current simulation tools do not cover certain areas that could be of importance in a Home Area Network (HAN). Therefore the rest of this chapter will analyze the following areas in order to determine their relevance to a simulation tool.

- Home Area Networks
  - Smart Meters' effect on a smart home
  - Communication technologies used in HANs
  - Which controllers units exist and how they control the home
- Consumer benefits
  - Economic benefits of using smart homes over regular homes
  - Personalization: The benefits of modeling a specific home as precisely as possible

## 2.2. Home Area Network

As mentioned in the introduction the Home Area Network (HAN) is a local area network within the private household. It is responsible for the communication between the intelligent appliances that use energy to perform their tasks. This section will examine the HAN in closer detail.

### 2.2.1. A Part of Smart Grid

The smart grid is based on a tiered architecture of a number of networks that connects all the parts of the smart grid, from power plants to the private home. The communication goes both ways which in theory allows all parts of the smart grid to communicate with each other. The smart grid can be divided into three main networks:

- **Home Area Network (HAN)** is as previously mentioned the network within the consumer's property. The outer communication point to the HAN is the smart meter, but the communication within the property is still unstandardized. The existing communication protocols will be examined in 2.2.3 *Types of Home Area Networks* later in this chapter. The data from the HAN is sent to a unit collecting data from multiple homes in the area.
- **Neighborhood Area Network (NAN)** is the communication between the data collection units and each of the consumers (houses) in the area. As there has not been agreed upon a common standard way, this communication can be either structured as a peer-to-peer setup between each consumer or as a normal wireless network with access points that each consumer communicates to. In the United States of America they are currently testing both solutions; in Texas they are testing the access point approach and the company Silver Spring Networks [35: Silver Spring Networks] are using the peer-to-peer setup in their solutions.

The communication point from the network to the house is the smart meter. Originally the intension was for utility companies to be able to read the values of the electricity meters remotely (called Advanced Meter Reading (AMR)), but now it has developed into a richer communication form that also contains the information about the energy prices being sent to the smart meter. This way of communicating to the smart meters is referred to as Advanced Metering Infrastructure (AMI) [36: Farhangi].



- **Wide Area Network (WAN)** is a network between the power plants, the producers (wind turbines etc.) and the distribution facilities and is used to transmit the consumer data from the NANs to the power suppliers. The smart grid can then regulate the power production based on the collected data. The collected data originates from the HAN network in the homes and is passed through and gathered in the Neighborhood Area Network.

The descriptions above show that the NAN is the chain connecting the WAN and HAN. As mentioned in the project definition this thesis only focuses on the Home Area Network of the smart grid. An overview of how the three networks works together can be seen below in Figure 7.

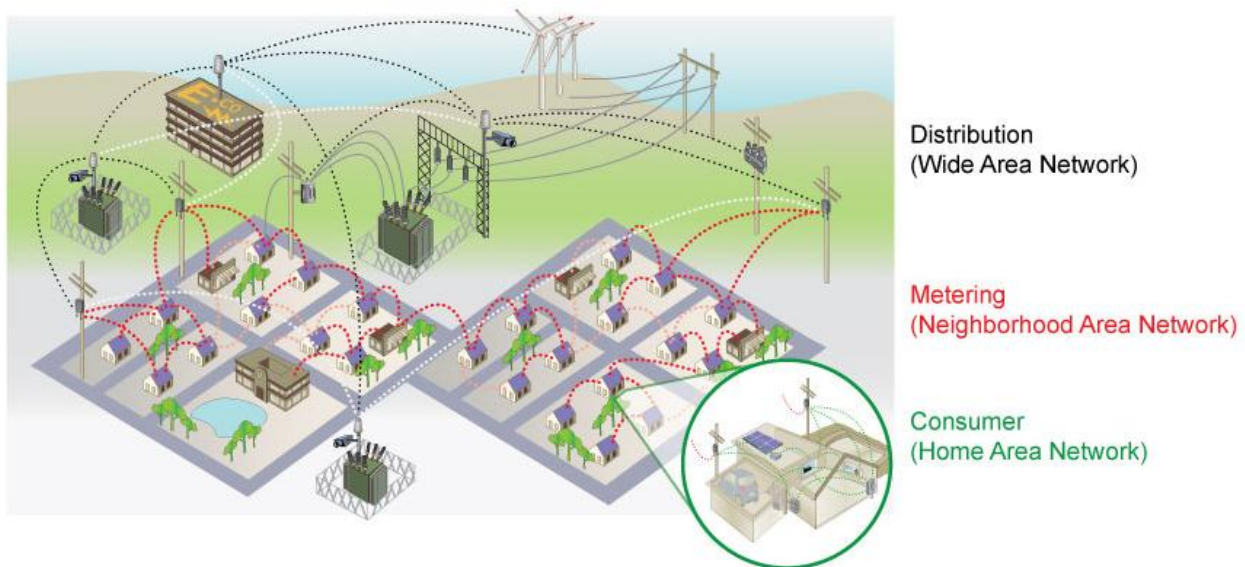


Figure 7: Overview of a Smart Grid power grid setup - Source: [37: Trilliant]

## 2.2.2. Smart Meters

In order to add intelligence to the power grid the power companies need information about the energy consumption both on distribution and transmission level, but also on consumer level. To be able to access the information on consumer level they need a way to communicate with the installed equipment in the homes. The first Smart Meters opened a one way automated meter reading (AMR) communication that allowed the distribution companies to read how much energy the consumer had used. While this enabled the power companies to have better overview of the power grid it did not reduce energy consumptions at consumer level.

In order to enable the consumer to use less energy or use it outside of peak hours the consumers need information about the current energy prices. This is provided by future smart meters that open a two way communication so that the consumer can see the current price for a kWh. Smart meters are, however, not necessary in order to receive price information. This can be seen with solutions like GreenWave Reality's smart home solution [4: GreenWave Reality] where they can receive the price information by installing gateways at each customer that communicates with services hosted at the energy company.

The communication between the smart meter and the distribution network is at the moment standardized with the set of standards within the IEC 62056 [38: International Electrotechnical Commission]. This set of

standards covers meter reading, tariff control, load control and standards for the layers of the data exchange.

The current energy prices can be retrieved from the smart meter and displayed by an in-home-display (IHD) to the user. The smart meter can also make the price data available to other appliances such that they can base their energy consumption on this (see 2.2.3 *Types of Home Area Networks* for details). Some of the existing smart meters, like Kamstrup's smart meters [39: Kamstrup], supports plugging external modules into the smart meter which enables them to communicate using the standard that the plugged-in module supports. Such types of smart meters are created to cope with the lack of a common communication standard and it also enables distribution companies to roll out smart meters now instead of waiting until the standards are decided upon.

In Denmark the distribution companies decide which smart meter manufacturer they will use in their network so because of the low number of distribution companies the amount of smart meter manufacturers is also low. Kamstrup [39: Kamstrup] and Echelon [40: Echelon] are currently the two most widely used smart meter manufacturers in Denmark as they offer solutions that can support multiple communication standards which opens up for communication in the home area network. In US the largest manufacturer of smart meters is currently Trilliant Inc. [41: Trilliant] which provides California with smart meters and in Texas it is Silver Spring Networks [35: Silver Spring Networks] that produces their own smart meters.

The purposes of a smart meter are simply to provide the HAN with information about energy prices as well as keeping track of the energy going in and out of the home. This is a requirement of all smart meters and as such the use of a specific smart meter does not change the consumption pattern of a home compared to another smart meter and therefore does not affect the result of a simulation of the energy consumption in a house. It can, however, provide real historic data that can be used in simulation scenarios if integrated with a simulation tool.

#### **2.2.2.1. *Legislative Restrictions on Privacy of Smart Meter Data***

A smart meter records all consumption of energy in the house. Based on the so called appliance load signature the type of appliance using the energy can be identified [42: IEEE Spectrum]. This means that the data from a smart meter is highly sensitive personal data as the holder of the data can get accurate details about what devices are in a house, when the person is home etc. According to the National Institute of Standards and Technology (NIST) [43] many users are hesitant to let utility companies get such information as they are afraid it might fall into the wrong hands. Parties that might be interested in this kind of data includes burglars who can see when people are home; insurance companies who can deduce potential health issues from the consumer's behavior in his home; or the police who can monitor suspicious patterns. All these aspects work against the common principal of freedom in your home. Therefore legal regulations should be put in place to ensure that data is handled in a satisfying manner. In Germany this has led to the Federal Office for Information Security describing a number of regulations that must be met by gateways that read data from smart meters as well as regulations about data stored in the smart meters themselves [44: Federal Office for Information Security]. The main points are to require smart meters to only send data to authorized third party gateways; to only allow gateways to receive data from authorized smart meters;

to allow authorized users to configure the gateway and to prevent gateways from storing meter data they no longer *need* to store.

Similarly in the Netherlands a consumer's energy consumption is considered private data and therefore such data cannot be published. In Denmark there are no such restrictions which may result in resistance to the introduction of smart homes. A similar situation exists in USA where NIST has pointed out the privacy issues with smart meter data and propose legislative changes [43: NIST].

If the application being developed in this thesis is to integrate real smart meter data the above mentioned restrictions must be considered. Therefore such functionality should either be postponed until the legislative issues have been resolved or left out for regions where such data is not legally available.

### 2.2.3. Types of Home Area Networks

There are currently two overall approaches for constructing a Home Area Network without looking at which technologies to use. Common for both of the approaches is that the smart meter is connected to the HAN along with all the appliances.

This part of the chapter will present the two approaches to a Home Area Network.

#### 2.2.3.1. *Decentralized*

In a *decentralized* HAN solution each appliance on the network will be responsible for their own scheduling. Each appliance is communicating to the smart meter directly using the chosen standard and they contain embedded controller software that based on the current energy prices received from the smart meter decides how the appliance needs to act.

This means that the smart meter is required to have a communication module that converts the data sent from the distribution network and transmits it in a readable format to the appliances in the HAN. Therefore the appliances and smart meter in a decentralized HAN will have to support the same communication standard. A common standard for all smart meters would be highly recommended.

With a decentralized HAN you will not be depending on one unit controlling your energy usage but each appliance controls itself. In a decentralized HAN you will be less flexible as each appliance cannot take other appliances status into consideration when deciding when to operate, but an advantage with a decentralized solution, however, is that the software on each appliance can be made by the appliance manufacturer and optimized for the specific appliance and thereby yield a better result.

#### 2.2.3.2. *Centralized*

A *centralized* HAN network is a network where each appliance does not contain any controlling logic internally, but instead exposes a software interface that enables external components to control the appliance. A single controller unit is connected to the network and based on the data from the smart meter (and potentially other sources) it controls the appliances in the HAN using the provided interface.

A great benefit of a centralized solution is that it creates one point where the security can be implemented and therefore each appliance does not have to provide their own security. A centralized provides a central point where all the data can be collected and stored if there is a period without internet access. Having a central controller will also reduce the risk of situations where all appliances start at exactly the same time

and overload the grid with high startup consumption. This can be solved by allowing the controllers, in cooperation with an even more central coordinator at the power supplier, to start the appliances when it is most suitable for the power grid.

A centralized HAN network can easily take information about power generation from DERs on the network into consideration, together with information about all the other connected appliances and coordinate the appliances to avoid simultaneous starts. This provides a better foundation for deciding when to start the appliances.

In a centralized solution both the smart meter, the appliances and the controller unit need to use a common standard for HAN communication as the controller will have to read the current price from the smart meter, read status on the appliances and control the appliances. As in the decentralized solution the smart meter again needs a module that converts the price data into the data format of the used communication standard.

### 2.2.4. HAN Communication Technologies

Regardless of which type of Home Area Network is used a common standard for communication between the connected units is required. There are currently two main categories of communication standards: Wired and Wireless solutions. Several technologies have been developed in these two categories that cover the needed functionalities that a HAN should provide.

When looking at a technology for a Home Area Network there will be a number of factors that are affecting the evaluation. This includes the pace of the technology innovation, upgradability, interoperability, cost and performance [45: Huq and Islam].

An analysis of the properties, advantages and disadvantages of the four state-of-the-art communication protocols used in HANs was conducted. These examined protocols are Ethernet, HomePlug, Wi-Fi and ZigBee. The analysis can be found in Appendix B. A summary of the results found in the analysis can be seen in Table 2.

	<b>Ethernet</b>	<b>HomePlug</b>	<b>Wi-Fi</b>	<b>ZigBee</b>
Data Rate	10-1000Mbps	4-8 Mbps	54-300 Mbps	320 Mbps in EU
Frequency	-	2-30 MHz	2.4GHz	868 MHz
Battery life	-	-	Weeks	Years
Security	On Device	56-bit DES	802.11i (WPA2)	128-bit AES
Range (In- & Outside)	100 m p. cable	300 m p. cable	100+ m	10 m to 75 m
Latency	Low	Low	Low	High

Table 2: Comparison of HAN communication technologies

A Home Area Network in itself does not provide any intelligence to the home. This intelligence comes from the controller units connected to it. Therefore the choice of communication technology does not influence how the appliances are scheduled or what information is available to the controller units. The analysis showed, however, that not all of the communication technologies can communicate directly with each other without the use of adapters. Therefore it is essential that all appliances use a protocol supported by the smart meter (in the case of a decentralized HAN) or that a central controller unit is inserted in the

network that supports the technologies (in a centralized HAN). Therefore it makes sense to take the communication technologies into account when modeling a smart home in a simulation tool.

### 2.2.5. Controllers for Home Automation

The controller units are the brains of the smart homes and they decide how each of the appliances need to act based on the information that the smart meter is providing as well as other data sources. This chapter will go through the reasons for using a controller unit, the concepts of a controller unit and look at which parameters that can be controlled on the appliances.

By making the user aware of the dynamic prices the consumer can save a significant amount of energy if he is using this information in his daily routines. A report from California Energy Commission [46] states that the average household can save 34.5% through technology used to inform them of the varying prices and units automatically adjusting their consumption to the price. In order to maximize the benefit from this information, however, you need something that can control energy consumption 24 hours a day. This is the so-called *controller unit*. A controller unit can, as mentioned in section 2.2 *Home Area Network*, be placed in either a centralized or decentralized HAN (in a decentralized HAN it is placed in each device). The centralized approach can be divided into two types of controller units: A physical unit installed in the home that controls each unit based on the customer's preferences or an external service communicating with the household over e.g. the Internet.

There is another aspect of controller systems as well: Who controls the devices? Michaels, H. & Donnelly, K. [47] introduces two overall architectures: Utility-controlled and consumer-controlled architectures. In the utility-controlled architecture the utility company controls when a home's devices are turned on and off. This is e.g. applicable to air conditioning, water heaters, pool pumps,  $\mu$ CHPs etc., but not so much to dishwashers, washing machines and the like. In the consumer-controlled architecture the devices are controlled based on consumer preferences. This is obviously the case in a centralized system where the consumer configures the controller unit, but can also be applied in a decentralized system where the user configures each unit to operate as desired. The utility-controlled approach has the advantage of clear energy savings as it is independent of the consumer's preferences and can be optimized for a national energy saving and the good of the power grid. Additionally the user does not need to buy any additional units to control his devices. According to Michaels, H. & Donnelly, K. [47], however, it sparks a fear of a Big Brother-like scenario in the consumers, which means that it is less likely to be adopted.

Consumer-controlled architectures have great potential if combined with time-of-use prices and a third-party controller unit or internal controller units in each device. It is possible that the same national energy savings can be made using a consumer-controlled architecture compared to utility-controlled architectures since people will typically aim for the lowest total price which in the scenario of properly set dynamic prices should result in the best usage of the energy from a national perspective. It does, however, mean that third-party controller units must be developed (or appliances with built-in controller logic) and the consumer must purchase these for the scenario to achieve the same savings as the utility-controlled architectures.

The rest of this chapter will examine existing controller units. It is worth noting that a lot of monitoring tools are available such as Google PowerMeter [3: Google] (discontinued), Microsoft Hohm [1: Microsoft] (discontinued) and EnergyHub [48: EnergyHub] which is a tool provided to energy companies, but these

systems only monitor the use or analyze the use of energy and possibly provide recommendations for changes to implement to save energy, but they do not perform any control of units and will therefore not be examined in this thesis. Similarly units such as Belkin Conserve Gateway [2: Belkin] provide functionality to display the dynamic prices from the smart meter and display energy use of utilities, but it does not perform any control. Additionally there are a wide range of units available that can automatically control your heating, lighting and security alarms in your house based on time of day, events such as opening doors or manually set preferences. Examples of these units are Commstar Home Automation Controller [49: SmartHomeUSA.com], HAI Omni Home Control Systems [50: SmartHomeUSA.com] and SmartLinc INSTEON Central Controller [51: SmartHome.com]. While they save a certain amount of energy, none of these take dynamic pricing into consideration (i.e. no interaction with the smart meter) and should therefore not be considered full-blown smart grid enabled controller units. All scheduling is based on when the user has manually chosen that the devices should start/stop or, in the case of lighting, occupation sensors. These tools will therefore not be examined here.

Smart grid enabled controller units that base their decisions on the dynamic prices from the smart meter are few, but the existing units will be presented below.

#### **2.2.5.1. GreenWave Reality**

The company *GreenWave Reality* has developed a home automation system that allows the user to setup automatic control of all electrical units in the home based on time-of-day, relation to other units or other preferences. Like the ones mentioned above the GreenWave Reality system does not currently use the dynamic prices from a smart meter, but this is under development through several projects such as eFlex [52: DONG Energy], iPower [53: DTU Elektro] and EcoGrid [54: EcoGrid].

The current GreenWave Reality solution consists of the following items: A gateway, an in-home display and a number of PowerNodes. The PowerNodes are electrical nodes that are placed between the electrical device and the plug allowing it to turn the electricity supply for a unit on and off. The gateway is the heart of the system and is connected to the smart meter to receive data about the energy usage, to the PowerNodes in the house and to a backend service over the Internet. All control of when to turn on and off devices is performed in the backend service, typically hosted by the utility company, and is sent to the gateway which then tells the PowerNodes to turn the power supply for the units on or off. Decisions on how to control the devices is based on preferences set by the user in a browser on a website connected to the backend system.

All communication is done by converting the device's communication from its original protocol (ZigBee, Z-wave, Wi-Fi etc.) to TCP/IP before sending it to other devices. When it reaches the device it is transformed into the communication protocol used by that device. This way all devices regardless of communication protocol can be used in the same HAN [GreenWave Reality, personal communication, 2012].

GreenWave Reality is currently working on several projects in which dynamic pricings are used. Unlike other controller units they have chosen to receive the dynamic prices from the utility company through the backend system rather than through the smart meter. The reasons for this are twofold. Firstly it allows them to introduce controller units in houses without smart meters. GreenWave Reality has developed a Pulse Reader that can be mounted on traditional electricity meters and send information about how much energy is being used and thereby the smart meter is not strictly needed in their implementation. The



second reason is that some countries have restrictions on what information you are allowed to extract from a smart meter which may prevent the sale of the unit in certain countries if they rely on data extracted from smart meters (see section 2.2.2.1) [GreenWave Reality, personal communication, 2012].

### 2.2.5.2. Control4

Control4 EMS 100 [22: Control4] is one of the few commercially available home automation systems that integrates with dynamic prices received from the utility through the smart meter and is therefore of interest for this thesis.

The Control4 EMS 100 controller system consists of three components: The EC-100 Controller, the WT-100 Thermostat and the Control4 Advantage Web Services. The relationship between these components and their place in the smart grid can be seen in Figure 8. The central component is the EC-100 Controller. It is connected to the smart meter using SEP 10 and to the local HAN using ZigBee. Based on the dynamic prices received from the smart meter as well as user preferences entered directly into the EC-100 it decides when to start and stop the appliances, lights, electric vehicle etc. in the house. Appliances are communicated with directly using the HAN and heating in the house is controlled by sending requests to the WT-100 Thermostat, also using the HAN. The EC-100 also allows the user to examine their energy use divided into overall categories such as lights, heating etc.

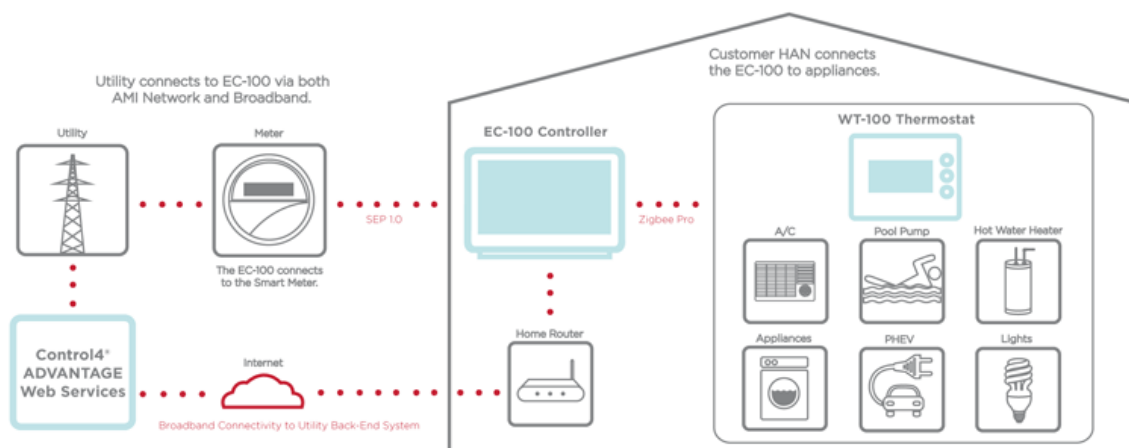


Figure 8: An overview of the Control4 EMS-100 controller system. Source: [22: Control4]

The EC-100 is also connected to the Internet through the home's regular router and this way it performs two-way communication with the Advantage web services which allows the utility companies to analyze the data about the home's energy consumption and use this for future marketing.

The Control4 EMS 100 system allows the user to control the charging of their electric vehicle based on user preferences and dynamic pricing [22: Control4]. While this is interesting and useful in many countries it is not necessary in the infrastructure introduced in Denmark by BetterPlace as described in 1.1.3.2.1 *Electric Vehicles (EVs)*.

Control4 EMS 100 is good example of a fully consumer-controlled architecture as it only allows the consumer, not the utility, to control when devices start and stop. The system is implemented in a centralized HAN which can be seen by the fact that the controlling logic is performed by the central controller and not by the individual appliances.

### 2.2.5.3. Tendril Connect

Tendril Connect [55: Tendril] is a home controller system developed by the American company Tendril and it aims at displaying, analyzing and controlling the energy use in a home. Figure 9 shows an overview of the Tendril Connect system. It consists overall of three important components: A cloud service, a web-based user interface and a number of in-home products. The important in-home products are Tendril Insight which is the in-home display and Tendril Transport which is the router creating the connection to the cloud service, and the Tendril Load Control Switch (LCS) which is the unit responsible for sending start and stop requests to the appliances in the home.

The consumer can configure his preferences through the web user interface, *Tendril Energize*, which is connected to the cloud services. The Tendril Transport product communicates with the smart meter to get the dynamic pricing updates and with the devices in the HAN. The information from these sources are sent to the cloud service in which the Orchestration and Coordination component determines when to run each appliance based on simulations performed in the Home Simulation Model. This information is sent to the LCS which then starts or stops the appliances as needed. Tendril Connect also offers the energy service providers the possibility of remotely turning off devices such as air conditioning and pool pumps in the peak hours in the homes. This is done through the cloud services and to the Tendril LCS [55: Tendril]. This shows that Tendril Connect implements a combination of the utility-controlled and consumer-controlled architecture as described by Michaels, H. & Donnelly, K. [47]. Tendril uses the centralized HAN approach by letting all control of the devices be performed by the cloud application and sent to the appliances. The appliances themselves do not implement and controlling logic.

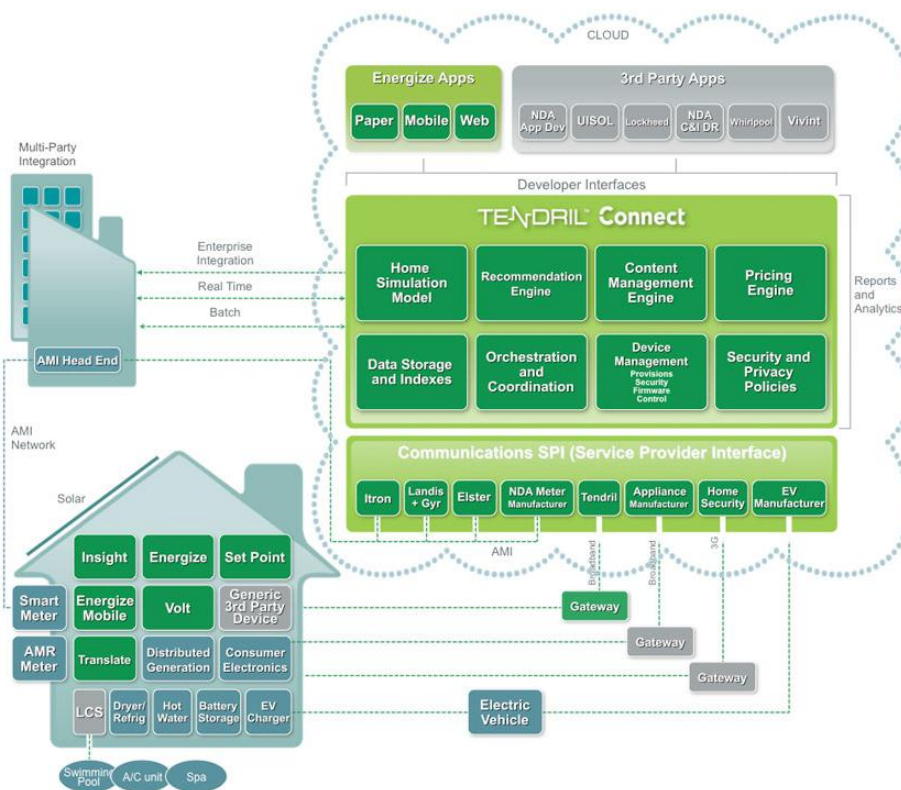


Figure 9: An overview of the Tendril Connect controller system. Source: [55: Tendril]



The Home Area Network that is set up when using Tendril Connect supports communication in ZigBee, Wi-Fi or HomePlug, but communication between Tendril components is performed using ZigBee.

#### 2.2.5.4. Comparison of the Controller Units

As seen in Table 3 all of the above mentioned controller units are using a centralized approach. This is most likely because of the lack of appliances that can control themselves, but also because it allows the manufacturer to send information about the consumption to their own servers for analysis. Such information can be used in attempts to sell more products to the given customer and is therefore of significant value. This is not possible if there is no single point through which all the appliances communicate.

	Controlled By	Home Area Network	Uses Varying Energy Prices	Communication Protocols
GreenWave Reality	Consumer / Utility	Centralized	No	All
Control4	Consumer	Centralized	Yes	ZigBee + WiFi
Tendril Connect	Consumer / Utility	Centralized	Yes	ZigBee

Table 3: Comparison of the examined controller units.

Additionally the solutions seem to open up for utility control of the homes, but the focus seems to be on letting the consumer define his preferences.

As mentioned earlier the controller units provide the logic that controls when to start and stop each appliance. Therefore the choice of controller unit could have a massive effect on how appliances are scheduled and therefore make a big difference in a simulation tool. This can by no means be ignored if a generic simulation tool is to be created.

#### 2.2.5.5. Closed Standards for Appliances

In order to achieve full effect of a smart home the central controller unit must be able to communicate with all the appliances in the home which therefore need to provide an interface that allows the controller unit to start and stop them as a minimum. Currently the appliance manufacturers that have developed such appliances are using their own closed standards for this and are not willing to share these with external companies as they seek to control the appliances with their own controller systems [GreenWave Reality, personal communication, 2012]. Therefore it is difficult for controller units to control appliances from different manufacturers in the same HAN which highly limits the usefulness of the controller units for the consumer. In order to resolve this, a common standard must be developed and made publicly available so controller units can control appliances from all manufacturers. It may require legislative action to achieve this if the industry association cannot agree on the use of such a standard.

## 2.3. Consumer Benefits

A challenge faced by the smart grid is that rolling out smart meters and upgrading the power grid will not be enough to implement a full scale smart grid. The smart meters can easily be rolled out as this is controlled by the utility companies, but in order to make the consumers adapt their energy consumption patterns they need either in-home displays or controller units. If consumers are to buy this they will need to be convinced that they will benefit from it. Some factors that influence this are whether they can see the

effect on their private setup at home as well as an estimate of the financial benefits of upgrading to a *smart home setup*.

As mentioned earlier none of the existing simulation tools provide possibility of accurately modeling the user's home. According to Barki & Hartwick [56] as well as McGilla & Klobas [57] users of a program feel more involved and positive towards it if it is of relevance to them. Involvement can e.g. be increased by allowing the user of the simulation tool to create their own personalized scenarios representing their own home and environment such as local weather data and energy prices. This could be done by letting the user of the simulation tool choose specific appliances that match their home and see what the energy bill will be for precisely their home. It also opens up for possibilities of testing out new appliances in their home without buying them as well as seeing the economic benefits of investing in distributed energy resources for the home. The personalized result will give them a realistic overview of the financial benefits of getting involved in the smart grid enrolment and provide a better incentive to buying a smart home system.

## 2.4. The Societal Impact of the Smart Home

This thesis will focus on the effects a smart grid will have on a private home, and more specifically the energy consumption and potential financial benefits related to this for the consumer. The smart grid, and thereby the smart home, provides different benefits for society and for the utility companies though. Using a smart grid will help reduce the load on the power grid in peak hours which will mean that society can save much money on not having to upgrade the existing power grid significantly. Additionally the utility companies (and society) can benefit from being able to move the homes' energy consumption to periods with high energy production. This way less energy is wasted and fewer booster power plants are required.

While smart homes that use local controller units to schedule their energy consumption can be very beneficial in moving the consumption to cheaper periods and thereby utilizing the energy better, they can also be counterproductive. One reason for this is that if all homes use the same controller they will move the consumption to the same point in time and start at the same time. Not only does this mean that the available energy may be exceeded at that point in time, but it also puts a very serious strain on the power grid since the energy consumption of components typically spikes for a short while when they start [Esben Larsen, personal communication, 2012]. Additionally, if all homes automatically schedule their consumption based on energy prices, the consumption will always move to the cheapest periods. This will even out the consumption, but also even out the prices which removes the financial benefits of moving the consumption which were the motivation for some consumers in the first place.

Therefore if local controller units are to be used in the real world it would be important to consider how the above issues can be addressed. One partial solution could be to let various areas of the grid receive different energy price information such that they do not all turn on at the same time. Alternatively consumers could be billed based on their flexibility and let external controller units (e.g. at the utility companies) control the consumption in the home as described in *2.2.5 Controllers for Home Automation* and *1.1.3.4 Flexibility*. The societal benefits will not be examined further in this thesis and an assumption has been made that a solution to the described issue with local controller units has been implemented.

## 2.5. Conclusion

A number of simulation tools have been developed in the area of energy consumption in a home. Some of them focus on heat generation while others focus on the energy consumption in a generic household. A comparison of these simulation tools has shown that none of them cover the following areas:

Communication technologies in home area networks; varying energy prices, the effect of using different controller units; personalizing the simulation scenarios to the user; and examining economic effects on a private household.

An analysis showed that the all smart meters are intended to behave similarly and the choice of smart meter will therefore not have an impact on the simulation result and is therefore not an important aspect of a simulation tool, although it can provide some benefits such as access to real data for energy consumption pattern matching. It turned out, however, that controller units and communication technologies used in the HAN cannot be ignored and therefore an improved simulation tool can be made that incorporate these two aspects. Additionally it was found that users feel more involved and positive towards programs that are relevant to them and as such the success of a simulation tool can be increased by personalizing the simulation scenarios. This can be achieved by opening up for modeling specific scenarios as well as focusing on the economic benefits a user can get in a smart home.

The rest of this thesis will cover the development of a simulation tool that incorporate the focus areas summed up in this conclusion.



## 3. Requirements

---

In the previous chapter the importance of smart meters, HAN communication protocols and controller units were analyzed along with an examination of existing simulation tools. Based on the conclusions drawn from these analysis this this chapter will go describe the desired functionality of this thesis' application. First the target audience is described and special focus areas are identified. Then an overview of the application domain is presented along with a description of the appliances in a home that are relevant to the simulation. All this leads to the formulation of a number of non-functional and functional requirements that are used as building blocks for the behavioral requirements, which will be formed as use cases. Finally the chapter presents a number of graphical mock-ups that describe the intentions of the graphical user interfaces for the program, which will be based on the information from the use cases.

### 3.1. Target Audience

Three groups of people have been identified that could be potential users of the application. These groups are *the consumer*, *energy advisers/salesmen* and *educators/educatees*. In this section each group will be examined and their preferences in the following four categories will be compared: *Focus*, *Interaction with Application*, *Desired Output* and *Generic vs. Real Data*. *Focus* refers to what the user wishes to achieve when using the application. *Interaction with Application* indicates what is most important to the user in the way he interacts with the application. *Desired Output* is the kind of output that supports the focus of the user the best. Finally, *Generic vs. Real Data* refers to whether the user prefers realistic data about weather, energy prices, and more importantly manufacturer-specific appliances or if he prefers generic versions of the appliances and customizable weather data and energy prices. The preferences are summed up in Table 4 on page 33.

#### 3.1.1. The Consumer

*The consumer* is the resident of a house or apartment that either wishes to implement intelligent control to his home or already has intelligent control. A consumer that already has intelligent control of his house can use the application to evaluate how new DERs or appliances can affect his electric bill. Similarly consumers that do not have a controller unit can use the application to see what they could gain from investing in intelligence to their home.

The focus of the consumer is to save money. This is the main concern for consumers when deciding whether they want to control their home intelligently. This has been documented a number of times e.g. by the Danish Competition Authority in 2009 [6: Konkurrencestyrelsen], the PowerCentsDC pilot program in Washington DC, USA in 2010 [58: eMeter Strategic Consulting] and Green Research in 2011 [59: GreenResearch]. Green Research found that 91% of the consumers in the survey valued saving money as the most important reason to use energy more intelligently. 8%, however, found that caring for the environment (the "clean feeling") was a motivating factor (see Figure 10). The survey also found that the longer the consumers used the smart meters the more they valued helping the environment.

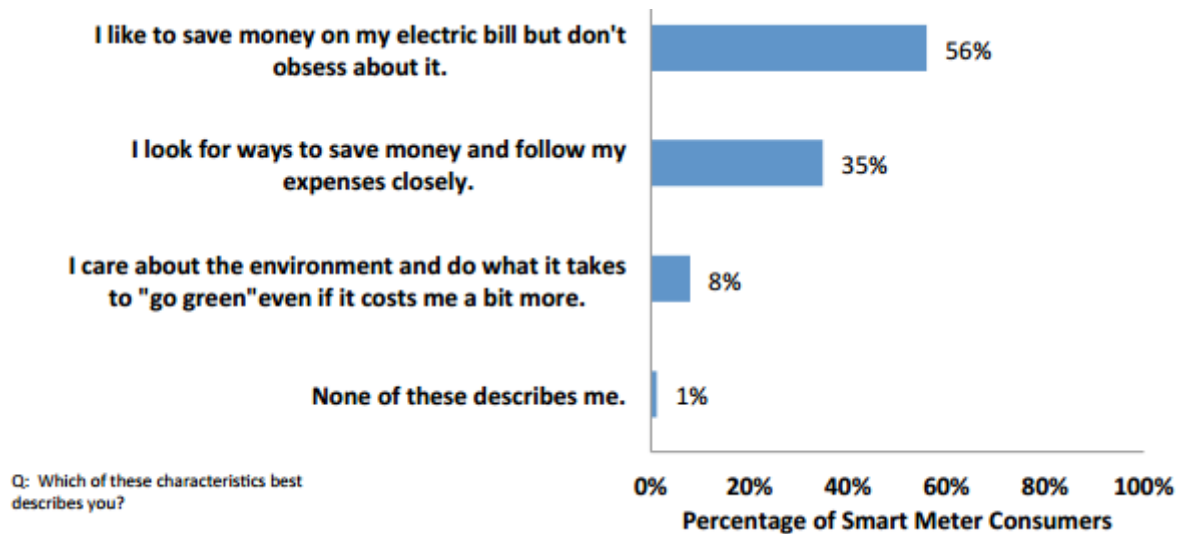


Figure 10: Motivations for saving energy [59: GreenResearch].

Since the consumer's main focus is to save money the consumer will desire an output of the simulation tool that can support this goal. The most important output will therefore be an indication of how much money is saved when comparing different scenarios. Information about how the appliances were scheduled in the scenario is less relevant.

When a private user is confronted with a new application it is very important that it is user friendly and easy to use. Otherwise it is likely that the user will not bother using the application and then the development has been wasted. How user friendliness will be achieved in this application can be seen in section 3.6.5 Usability.

For the consumer it is more interesting to be able to see simulations of their home using the actual appliances present in the home. If he is to trust the result and base decisions about purchase of new units on these results they have to be as close approximations of his home as possible and therefore the consumer is likely to prefer real data over generic data.

### 3.1.2. The Advisor/Salesman

This group consists of two subgroups that turn out to have almost the same preferences the application. One is the neutral energy advisor that helps the consumer changing his energy consumption. The other subgroup is the salesman that attempts to sell controller units, intelligent appliances or DERs to the consumer. The main focus of both subgroups, however, is to achieve positive results for the consumer's focus. For the advisor it is to provide the best possible guidance and for the salesman it is to convince the consumer to buy his product. Therefore the two subgroups have been combined into one group in this thesis.

The advisor/salesman will typically achieve this by comparing the simulated results of the scenarios. Not just the total price but also how the appliances were used in order to provide reasoning for changing appliances or adding new devices.

The advisor/salesman works with the application daily and while it is still nice to have an easy to use application as it decreases the learning curve it is more important that the application is rich in useful features such that they can provide detailed and customized results to their clients.

The advisor/salesman can use either generic or real data. Real data is useful for the same reasons as they were useful for the consumer, but generic data can be useful if the consumer cannot or does not provide detailed information about the actual appliances in the home. In such cases default generic data is very useful.

### 3.1.3. The Educator/Educatee

The application can also be used in education scenarios. This includes educating high school/university students, private house owners or public education in the mechanics of smart homes. As mentioned the main focus of education in smart homes is to examine the mechanics as well as the effects of the smart home. To do this it is essential to compare scenarios on a deeper level than just examining the money saved such as examining how appliances are scheduled, how it matches the dynamic energy prices, how much energy is generated by DERs etc.

Such education would probably be a course ranging from a few hours to a few days and therefore it is essential that the application is easy to use so the time is not wasted learning how to use the application.

For education purposes generic data is likely to be preferred as it simplifies the creating of scenarios in the sense that there is 1 generic washing machine rather than 25 different specific washing machines to choose from, and since the purpose isn't to achieve an accurate simulation of a specific scenario there is not much point in using specific appliances.

### 3.1.4. Comparison of Potential Target Audiences

It is obvious from Table 4 that the user groups have several things in common. Figure 11 shows Venn diagram of how the preferences of each user group match the preferences of the other groups in order to find similarities and thus important requirements for the application.

	<b>Focus</b>	<b>Desired Output</b>	<b>Interaction w. application</b>	<b>Generic vs. Real Data</b>
<b>Consumer</b>	- Saving money - ("Clean feeling")	- Price savings	- User friendly	- Real
<b>Advisor/Salesman</b>	- The consumer's interests - (Sale)	- Scenario comparisons	- Rich in functionality	- Both
<b>Educator/Educatee</b>	- Mechanics	- Scenario comparisons	- User friendly	- Generic

Table 4: The preferences for the application of each of the user groups.

Based on the similarities in Figure 11 and the descriptions of the use of the application above, it seems realistic to develop an application that can be used by all three user groups. Therefore the target audience will be the combination of the three user groups.

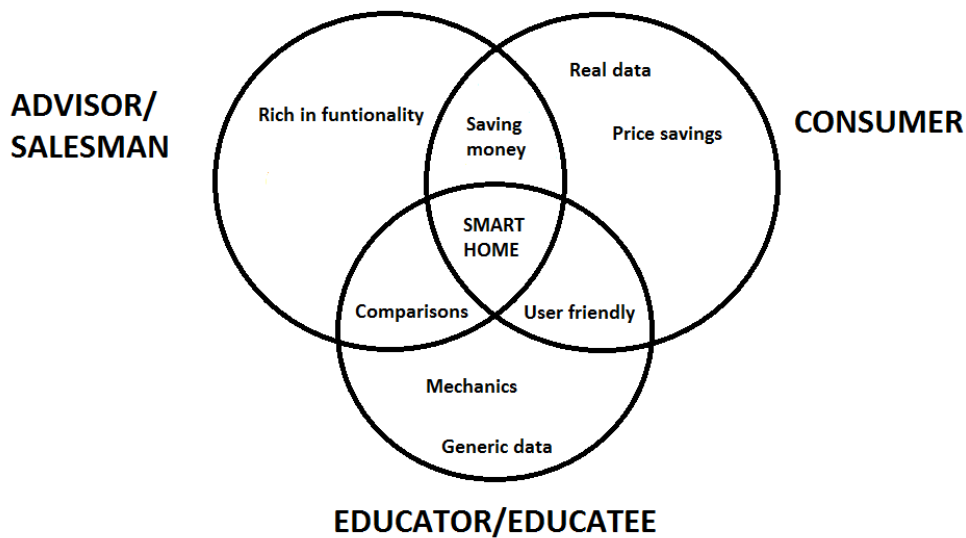


Figure 11: A Venn diagram showing the similarities in the preferences of the user groups.

The preferences shared by two user groups are the features that are important to most people and will therefore be the central functionality of the application. Therefore functionality to compare simulations and to see how much money can be saved will be essential and the application will prioritize usability very highly. Preferences only considered important by one user group may be implemented based on perceived importance to the user group and the preference’s capability of providing useful functionality for other user groups.

### 3.2. Main Purposes of the Program

As mentioned in the introduction to this thesis the program is a simulation tool that looks at the impact of integrating smart grid into a private household. The program needs to help the identified target audience to see the possibilities and benefits of integrating a home with the new intelligent power grid. Based on the analysis above of the target audiences the tool shall provide a user friendly interface that allows the user to easily construct a model of a home and then simulate how smart grid technology would affect the power consumption in their home. The program shall with these simulations be able to show the user the results in a customizable way that allows them to compare scenarios against each other. The results should also enable the user to examine the financial consequences of the smart grid and get an overview of possible improvements to the modeled setup.

The application should contain generic implementations of a number of components in the home and provide functionality to implement or import new specific components. Additionally, as the smart grid field still contains a range of areas in which the final structure of the grid and its standards has not yet been decided upon the application should be able to adapt to the changes that may occur in these areas.

The program is created as there is a need for educating the public in smart grid in private home and a need for acknowledging the effects of the new smart grid. Currently people know very little about the smart grid and hopefully this tool can help remedying this.



### 3.3. Domain Model

In order to get an overview of the smart home domain a model has been created that illustrates how the various concepts in a smart home/smart grid setup that were identified in the previous chapters are connected with the elements of an application as the one developed in this thesis. This model can be seen in Figure 12.

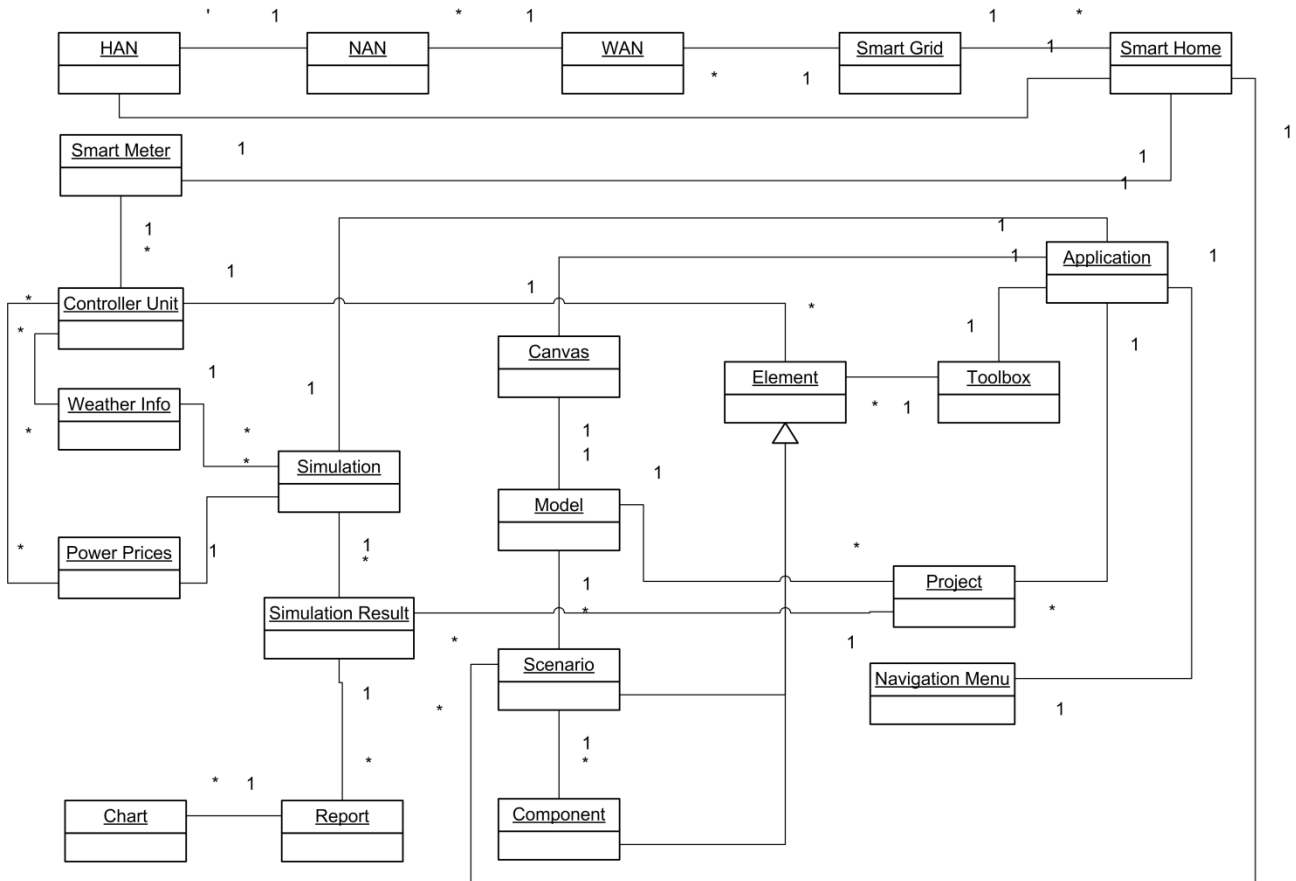


Figure 12: A domain model showing the relationships between the concepts in a smart home and smart grid as well as the application.

Based on the preferences of the target audiences identified in 3.1 *Target Audience* the domain model includes a number of concepts to help meet these preferences. The concepts of *canvas* and *toolbox* are used in many other modeling applications such as Visio and Enterprise Architect to provide a fast and easy modeling experience and are therefore included here as a potential way of creating the models in this application in a user friendly fashion by dragging elements from the toolbox onto the canvas. In order to achieve the goal of comparisons between simulations the concepts of *report* and *chart* have been introduced as reports consisting of charts are good at presenting data in an easily comparable way.

The concept *Element* refers to the elements that are part of a modeled home. Elements can either be scenarios, which are specific setups of a home including all the components in the home or it can be the components themselves. In the rest of this thesis the concept *scenario* will refer to a specific modeled setup of a home and the concept of *component* will refer to each element in that home. These components will be examined in detail in the next section.

### 3.4. Modeling Components

A home consists of a lot of components such as lights, refrigerators, freezers, computers, TVs, washing machines etc. Some of the components, such as washing machines, can be scheduled based on dynamic prices while others, such as TVs, computers and lights, cannot as they should work when the user requests it and not when the energy is cheap. While you can still implement energy saving initiatives for non-schedulable components such as automatically turning off the components when you're not home, this thesis focuses on how components can be scheduled based on dynamic prices during a day. Therefore components will be placed in two groups; schedulable and non-schedulable components. Schedulable components will, as the name indicates, be scheduled by the simulation based on the dynamic prices, whereas the non-schedulable components will simply be run during a user-defined period each day regardless of the price.

The schedulable components such as washing machines, dryers and dish washers can be scheduled based on when the user needs them to be finished and what the price is, whereas refrigerator, freezer, heat pump and water heater need to be kept at a certain level at all times. It is, however, possible to turn them off for a few hours during peaks to save some energy and then heat/cool a bit more in low periods [60: Capion]. This will later on be used to subgroup the schedulable components.

In addition to the already mentioned components there are two more groups: DERs and hybrids. DERs energy production is based on the weather at any given time and the production will be subtracted from the consumption of the other components. Overproduced energy will be fed back into the grid if possible or stored in a battery if one is available in the home. The battery can then be used to feed energy back to the home when the price is high. Electric vehicles work in much the same way since the EV's battery can be used for storage. This, however, only applies if the EV is connected to the home's power network (which is not the case in BetterPlace's infrastructure [20: BetterPlace]). The most important components are summed up in Table 5:

TYPE	Non-Schedulable			Schedulable			Other	
COMPONENT GROUP	Lights	Entertainment	Cooking	Appliances	Heating	Water	DERs	Hybrids
COMPONENT	Incandescent bulb	TV	Oven	Washing machine	Heat pump	Water heater	Solar panel	EV
	Energy saving bulb	Computer + Monitor	Cooking plates	Dryer	Electric Radiator		μCHP	Battery
	Halogen bulb	Gaming Console	Exhaust hood	Dish washer			Wind turbine	
		Laptop	Electric kettle	Refrigerator				
		Modem/Router		Freezer				

Table 5: The component groups and examples of what elements exist in the groups.

Each type of component has a specific range of properties that are relevant for the simulation and should be configurable by the user. These properties have been listed in Figure 13. All *periods* listed in the figure will include values for weekdays and weekends separately as these periods will often be highly different.

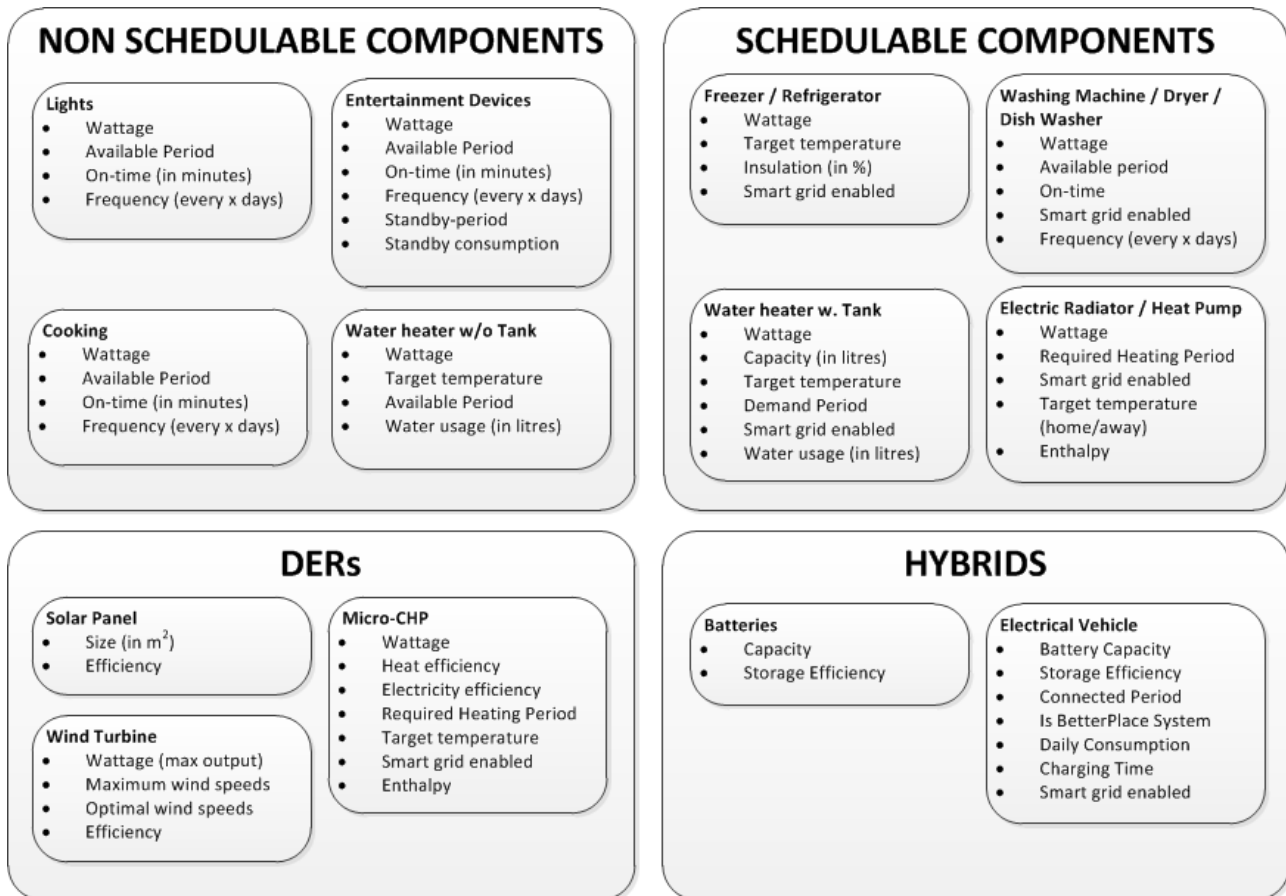


Figure 13: A diagram showing the configurable parameters for each type of component.

### 3.5. Realism in Simulations

The energy consumption in a home can be divided into different categories. Each category of components contributes with a certain percentage of the total consumption. Examples of these categories and percentages can be seen in Figure 14 showing data from IT-Energy ApS. They have conducted a data analysis and survey of the energy consumption in houses called ElModelBolig [61: Elmodelbolig]. Together with another project of theirs called *Selvtjek Bolig* [62: Elsparefonden] they inform the customer of how their energy is used and they have records of the consumption of each appliance in the households. Figure 14 shows a pie chart with the average consumption for a family of four in a normal sized house. It is taken from Go'Energy but is based on data from IT-Energy ApS.

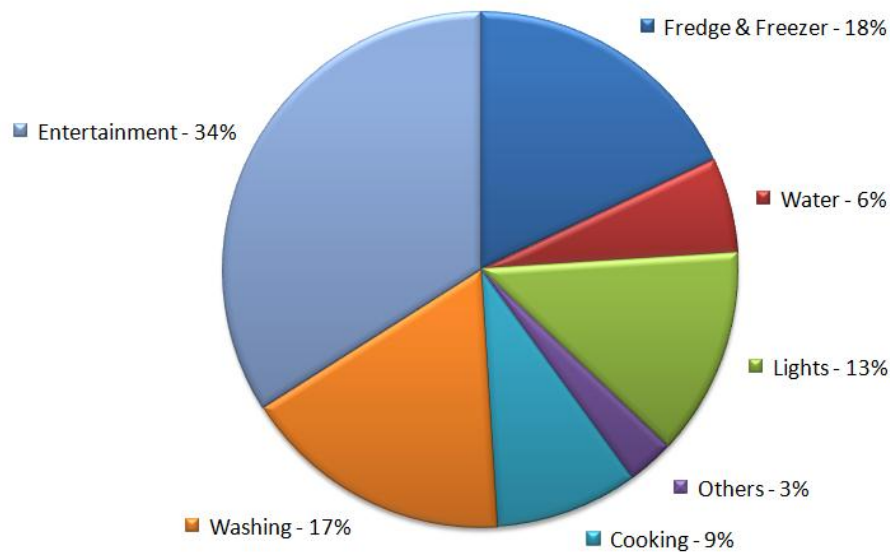


Figure 14: Energy consumption and distribution. Translation of figure from source: [63: Go Energi].

ElModelBolig also lists the wattage and standby wattage of a lot of elements in a home which can be used to provide realistic properties for the created elements [64: Elmodelbolig]

The data in Figure 14 can be used to ensure that the models created in the simulation tool are realistic. Based on the above surveyed results on how people energy consumption is divided into different categories, the program could perform a validation of the created models to ensure that the model created by the user is realistic.

## 3.6. Non-Functional Requirements

The program is going to focus on five key areas:

- Customizability
- Extensibility
- Reporting oriented result
- Performance
- Usability

These areas have been chosen based on the requirements from the target audience as well as the analysis performed in chapter 2. These areas define the non-functional requirements of the application that should define the overall philosophy of the design and implementation of the application. The following five sections will describe each of the five non-functional requirements.

### 3.6.1. Customizability

As preferred by two of the three target audiences the application must be able to create customizable models of homes such that they resemble the actual homes as close as possible. This includes enabling the user to customize any aspect of the application such as weather information, power prices, the components in the home etc. Therefore the user has to be able to create new and edit existing modeling

components to fit their needs. This can also be used to give the user ways of creating scenarios that can show the improvement when for example buying new appliances.

Allowing the user to setup their own reports of the simulations will give the user the freedom to see what interests them and help fulfill the preference of simulation comparisons as identified in *3.1 Target Audience*. Dedicated controls can give the user total control over what they see in the reports and compare the results. A salesman would e.g. not be interested in seeing a report describing how the energy consumption varies during the day, but more in showing the savings that the client could get by upgrading the appliances.

As some of the components in the modeling are depending on the weather information the program should contain information about previous weather conditions and it shall get a future weather forecast for consumption prediction. It shall be possible to create custom weather data sets so that the user can setup a specific scenario where the weather conditions are as desired. The same principle shall also apply to the energy price information that the system contains. This information shall be editable allowing people to simulate special events that cause a large energy price increase.

### **3.6.2. Extensibility**

There are currently two main reasons that the program need to be extensible. First of all there are still, as mentioned earlier, a lot of undecided standards within the smart grid. So therefore the program has to cope with the uncertain future and allow the developers to be able to easily implement standards as they get decided upon. Therefore the design must enable future developers to easily extend or modify the application. The second reason for having a requirement of extensibility is that the solution in this thesis is only a prototype, so there are some features that this thesis will not cover. So the solution needs to be easily extensible for further development on the prototype.

### **3.6.3. Report-Oriented Result**

A key requirement identified in *3.1 Target Audience* is to compare modeled scenarios. In order to enable this, the application must be able to create reports based on the simulated results that can be customized to the user's needs. Graphical reports containing charts of the simulated data provide the user with a powerful tool for comparison and can be used to show both a brief overview of the results as well as very detailed information about specific components.

### **3.6.4. Performance**

A main issue to be aware of in every program is performance. The program must not be too slow when performing the simulations and retrieving external data and it should be responsive at all times. Therefore performance must be considered when designing and implementing the application. Simulations are typically heavy computations and the more detail is added the higher the complexity gets which in turn increases the computation time. Therefore it is reasonable to assume that performance will be a significant factor in the success of this application.

### **3.6.5. Usability**

Section *3.1 Target Audience* found that both the *consumer* and the *educator* target audiences put significant value into usability. Consumers rarely want to spend a long time learning how to use a new tool

and educators do not have the time in their courses to cope with a steep learning curve. Therefore usability is a prerequisite for success in applications such as this. Relying on designs and features that are similar to what the user is used to from other applications can be a good way of lowering the learning curve. This section will present eight usability suggestions from the ISO standard 9241-171 and five suggestions from other sources. Section 5.8 *Usability Adherence* will describe how these areas have been used in the prototype implementation.

In 1998 the International Organization for Standardization (ISO) developed a standard regarding, among other things, how humans interact with systems. In 2008 this standard was revised under the name *ISO 9241: Ergonomics of Human-System Interaction*. Part 171 [65: ISO] of this standard is called *Guidance on Software Accessibility* and is concerned with general guidelines for usability of software systems. The standard covers many areas; some that are not applicable in all situations, but some of them are relevant to this thesis and will be presented below:

- **Meaningful element names**

According to section 8.1 of the standard it is important that all UI elements get meaningful, contextually unique displayed names except when it would be redundant. All words used in labels should be natural language words that are available in a dictionary (or alternatively a glossary provided with the software) such that the users know what the buttons do.

- **Location of elements should be platform coherent**

Lastly the location of labels should be consistent and use the platform standard if one is available [65: ISO]. This is supported by *Nielsen and Norman Group* who states that “*Location is only just slightly less important than appearance*” [66: Tognazzini].

- **Never only color to convey message**

The standard also stresses that colors (or font types or styles) should never be used as the sole way of expressing information to the user. This is partly due to people with disabilities that require AT to read the information and partly due to people with color blindness (which is 10% of all males [66: Tognazzini]). Therefore, when colors are used to convey information such as positive numbers being green and negative numbers being red additional clues should be given such as putting a minus in front of the negative number [65: ISO]. This is important in this application as it will often display information about consumptions which can be either positive or negative (in the case of energy production).

- **Optimize task steps**

Section 8.4 of the standard highlights a number of important aspects that this thesis should take into consideration. One is to optimize the number of steps a user has to perform when carrying out a task. A good balance is required between providing enough explanation and limiting the amount of time it takes the user to achieve the desired result. Generally frequent tasks should have fewer steps and more infrequent tasks should focus on better explanation [65: ISO].

- **Undo/redo**

Another important focus is undo mechanisms. It is important that users can undo unintended actions (which are highly likely for disabled users) and that they are not scared off from trying new features because they are afraid of ruining the state of the system [65: ISO]. Undo features are also highlighted as highly important by the *Nielsen and Norman Group* [66: Tognazzini; 67: Nielsen] and are therefore of high priority to this application.

- **Alternative input methods**

Additionally, it is a good approach to provide the user with alternatives to typing when the set of possible inputs is limited. This can e.g. be by using calendars to allow users to choose dates or by implementing dropdown menus or file dialogs. In this thesis these approaches will be used several times as choices between various input data as well as setting dates for simulations will be frequent actions.

- **Clear error messages**

The presentation of error messages is also touched upon in the standard. It is important that errors are displayed in a consistent way and the user should always immediately know if it is a notification, a warning or an error. The user should be able to navigate to the location of the error. An example of this would be to indicate the text field in which the input is invalid [65: ISO].

- **Multiple access to functionality**

Different users have different ways of performing certain tasks therefore multiple ways of performing the same task should be available. One of these should be using accelerator keys (also known as keyboard shortcuts) to allow more advanced users to perform certain tasks faster (section 9.3). These shortcuts should be consistent with the platform such that a *save* action on a Windows platform is performed using Ctrl-S, *undo* is Ctrl-Z etc.

- **Group controls based on functionality**

The last points from the standard that will be addressed in this thesis are recommendation 9.3.17 and 9.3.18 that suggests that navigation controls should be grouped based on their functionality and the logical order of the tasks to be performed. This approach will be used to arrange the navigation bar items in groups such that modeling specific controls are in one place, simulation controls in another etc. [65: ISO].

Besides the described suggestions from the ISO standard, there are also some other recommendations that the program could adhere to. The recommendations are made by Jakob Nielsen and Bruce "Tog" Tognazzini from the Nielsen Norman Group [68: Nielsen Norman Group]. They are both much respected usability engineers and have for more than 30 years both developed and designed software with usability in mind. Listed below are recommendations together with parts of the program for which they are relevant.

- **Drag-and-drop**

A drag-and-drop functionality could be very useful in the modeling part of the program as it can provide an easy and intuitive way of creating the models. By itself it is a very nice mechanism for a dynamic interface but the user needs to be aware of its existence. So to make the drag-and-drop mechanism more user-friendly you can improve your solution with several initiatives:

- You need to inform the user that they can drag the elements into a desired area. This can be done by small tooltips that appear on defined events within the program that tells the user what they can do. It can also be done by having a static area informing the user about the drag-and-drop functionality. An example of the last improvement can be seen in a blog on the User Interface Engineering Homepage [69: Spool] where they describe how this is done by Netflix.
- A simple improvement is just to let the mouse cursor change when the user has their mouse over the drag enabled elements within the program.

- **Keep users aware and informed** [66: Tognazzini]  
A common problem with applications is to keep the user informed of what is going on and aware of events happening within the program. If the user cannot see what is going on he might think the application has stopped working. Therefore a program needs to have a communication mechanism that allows the program to communicate to the user in a less confronting way than a popup message. A way of doing this is to use a status bar. It can easily be designed to always show what the program is doing and thereby fulfill this usability requirement.
- **Match between real world and system** [67: Nielsen]  
In order to make users understand what the application does it is important that words have the same meaning in the application as they do in the real world. If the user is already familiar with the domain the application is related to it will be confusing to see new words for existing concepts and if the user is unfamiliar with the real world domain it will be difficult to learn two new words for each new concept.
- **Avoid unnecessary information in dialogs such as rarely used fields etc.** [67: Nielsen]  
Showing forms to the user containing a lot of fields with default values and fields that are very advanced should be avoided. This can easily confuse the user and make them skip the assignment they are doing or just letting them go further with incorrect data.
  - A good initiative is to group the fields into a standard and advanced view. Then just let the user from the start see the standard view with possibility to switch to the advanced view with more settings if wanted.
- **Walkthrough usage guidance**  
Often users will not spend much time learning how to use a program and if it is not immediately obvious how to perform a task they are likely to not use the program. Therefore it is important that the application guides the user to perform the actions correctly without requiring him to read a manual first. This can e.g. be a step by step tutorial that appears when starting the application or it can be hints that are displayed while the user uses the application describing the next step.

### 3.7. Functional Requirements

The non-functional requirements described above define the basic philosophy of the application. In addition to this, a range of specific functionality can be defined which is generally referred to as *functional requirements*. This functionality describes the main features of the application and these requirements can be used to validate if the application fulfills the requirements when it is finished as well as be used as foundation for test cases (this will be done in chapter 6). The overall functional requirements are listed below:

1. **Create one or more models of a house (a scenario)**  
In order to fulfill the non-functional requirement of *customizability* and the target audiences' preference of personalized home modeling it is essential to allow the user to model their own house in the application. It is a prerequisite for any simulation that a model is created to base the simulation on.



2. **Simulate energy consumption of a scenario**

One of the main areas identified in the target audience analysis (see 3.1.4 *Comparison of Potential Target Audiences*) was the saving money. To enable the user to see how much money he can save it is necessary to perform one or more simulations of the modeled home that calculates the energy consumption in a given period.

3. **Create custom components to use in the modeling**

To enable the user to create accurate models of their home they must be able to use simulations of the actual components they have in their home. This requirement ties into the *customizability* requirement mentioned above.

4. **Import and edit energy prices**

As this thesis focuses on how homes will respond to dynamic energy prices it is essential that such energy price scan be imported into the application. Preferably these should also be editable based on the *customizability* requirement. This allows the user to simulate situations such as how the energy consumption would react to unnatural high energy prices.

5. **Import and edit weather forecasts**

As identified in chapter 1 the DERs play an important role in a smart grid and in order to simulate how much energy can be produced from solar panels and wind turbines it is necessary to import data such as wind speeds and sun intensities. To enable users to simulate hypothetical weather situations the data should be editable as defined by the *customizability* requirement

6. **Create graphical reports**

In order to leverage the non-functional requirement *report-oriented result* the application should be able to create graphical reports containing charts that display the simulated data in useful and customizable ways.

7. **Save/load files**

Save/load functionality does not provide extra features for the simulation, but highly important if the users are to use the program without having to recreate the models and simulations every time they use the program.

8. **Undo/redo mechanisms**

As described by the usability requirement in section 3.6.5 it is important to provide the user with ways of undoing their actions in order to ensure that they are not afraid of doing something wrong.

9. **Guide the user**

The *usability* requirement also specifies that users should be guided in their actions in the application such that they learn how to use the application without reading a manual first. This should be done by creating guidance mechanisms in the application.

These functional requirements create the backbone of the program together with the non-functional requirements and form the basis for the developed use cases of the application.

### 3.8. Behavioral Requirements (Use Cases)

A number of use cases have been developed that describe how the above mentioned functional requirements will be used in the application by the user. Table 6 shows all the identified use cases ranked by priority and displays which functional requirement the use case is related to as well as the importance of the use case.

A use case diagram can be seen in Appendix C, but has not been included in this chapter due to its simplicity, having only one actor in all use cases. The detailed use case descriptions describing each step in the use cases can be seen in Appendix D.

As seen in Table 6 there are a lot of use cases and it is unlikely that all these can be thoroughly implemented in the limited time frame available. Therefore each use case has been assigned an *importance level* that describes how critical the use case is to the main purpose of the application, i.e. to simulate the consumption of modeled homes. The use cases with the level *critical* are highly important and the application will be of no use without the implementation of these. Use cases marked as *important* are highly useful and should be implemented if at all possible as they provide the application with functionality to create more complex models and provide a much higher usability of the application in general. Use cases marked as *nice to have* provide useful functionality that helps bringing the application from a prototype to a final application, but these are not necessary in order to fulfill the overall goal. Lastly use cases marked as *cosmetic* are features that do not provide much value to the application in terms of overall goals, but provide some features that some users might find interesting in certain situations.

Priority	Use case name	Functional Requirement	Importance
1	Create Project	1	Critical
2	Create a new scenario	1	Critical
3	Add component to scenario	1	Critical
4	Delete component added to scenario	1	Critical
5	Simulate energy consumption in scenarios	2	Critical
6	Compare simulated scenarios	6	Important
7	Change settings of existing scenario	1	Important
8	Change settings of component added to scenario	1, 3	Important
9	Delete scenario	1	Important
10	Define period of simulations	2	Important
11	Create custom charts of energy consumption/production etc.	6	Important
12	See dynamic energy prices for a period	4	Important
13	See weather forecast for a period	5	Important
14	Create new component	3	Important
15	Edit existing component	3	Important
16	Delete existing component	1	Important
17	Delete chart from simulation window	6	Important
18	Save a project	7	Important
19	Load a project	7	Important
20	Undo action	8	Important
21	Redo action	8	Important

22	Print report of simulation	6	Nice to have
23	See estimated yearly usage in watts for a scenario	6	Nice to have
24	Browse components from simulation	6	Nice to have
25	Move a scenario	1	Nice to have
26	Duplicate scenario	1	Nice to have
27	Edit dynamic energy prices	4	Nice to have
28	Edit weather forecast for a period	5	Nice to have
29	See results of earlier simulations	2	Nice to have
30	Export Scenario from a project	7	Nice to have
31	Import Scenario into a project	7	Nice to have
32	Print model	6	Nice to have
33	Recreate model from simulation	1	Nice to have
34	Popup hints displayed	9	Nice to have
35	Hide popup hint	9	Nice to have
36	Select specific historic weather forecast	7	Cosmetic
37	Save chart as image	6	Cosmetic

Table 6: Use cases ranked by priority and displaying which functional requirement the use case is related to.

## 3.9. Graphical User Interface Mock-ups

In order to get an understanding of the user interface before the technical design and implementation is begun a number of GUI mock-ups have been created based on the use cases described earlier in this chapter. They show what the main parts of the application should look like and describe what functionality is available from the given window. These mock-ups will also help guide the design and implementation process by describing the intended graphical end results. The mock-ups are displayed and described below.

### 3.9.1. Main Window

The mock-up in Figure 15 shows the main window that is displayed when the user starts the application. It contains the canvas on which the user creates the model, a toolbox with all the components and a summary that gives some details about the created models. This is the starting point for all actions.

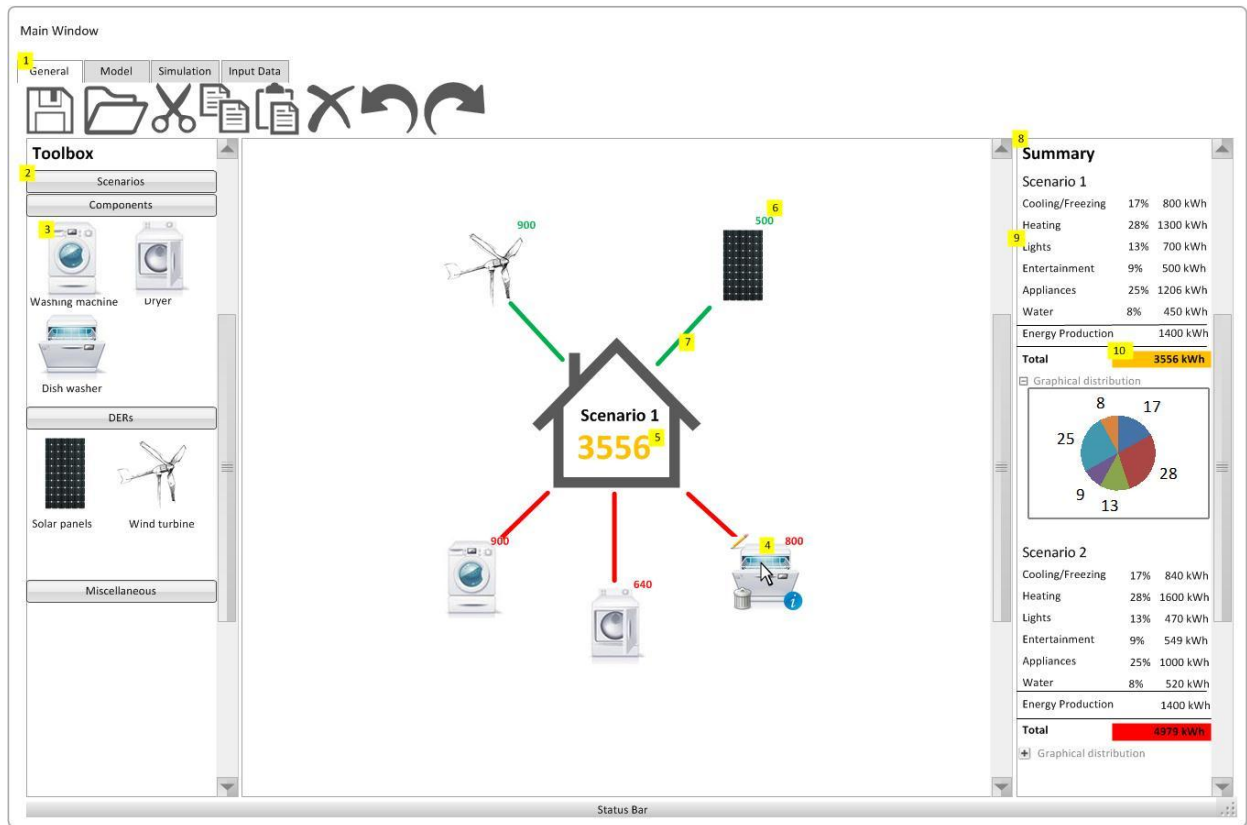


Figure 15: Mock-up of the main window in the application.

Note	Comment
1	The user can click on the tabs of the navigation menu (Ribbon) to change which icons are displayed in the menu. These icons are grouped into logical sections such that commands related to modeling are in the Model tab etc.
2	When the user clicks on a group in the toolbox the group is expanded to display the icons of the components in the group. When a user clicks on the group again it is collapsed.
3	The user can drag and drop icons from the toolbox onto the canvas to add them to a scenario.
4	When the user holds the mouse over a component or the house, small icons will be displayed allowing for quick access to Edit, Delete and Information functions.
5	This number displays the total approximated usage in kWh for the scenario. The color of the number is based on how high or low the consumption is, based on an average household of the same size. The same information can be seen as a percentage by hovering over the information icon of the scenario.
6	This number shows the yearly consumption or production of the unit. The number is green and shown with a negative sign if it is production and yellow, orange or red if it is consumption.
7	The line is automatically drawn from the house to the unit and its color is based on the type of component, i.e. the category as defined in 3.4 Modeling Components.
8	This panel shows an approximated distribution of the yearly energy consumption in the scenario in kWh and percentages for each scenario.
9	The user can hover over a category to get information about the breakdown of the category consumption.
10	The total yearly consumption of the house is displayed in the same color coding as the consumption value in the house on the canvas. This too, can be seen a percentage by hovering over the information icon of the scenario.

### 3.9.2. Navigation (Ribbons)

The primary navigation of the application will be displayed as a ribbon as known from Microsoft Office 2010 (and Word 2007 as well as Windows 8). Ribbons give a good overview of and easy access to the functionality and it is a design that most people should be familiar with by now and thereby reducing the learning curve. It is a design that favors the inexperienced users [70: Wang] and as there are no experienced users for this application the inexperienced users is the segment to target.

There will be four tabs in the ribbon; General, Model, Simulation and Input Data. Each tab will contain functionality for a specific part of the application. Grouping the functionality for separate tasks in separate tabs is called *task centered arrangement* and is central in the ribbon concept [71: Hedges] and supports the *usability* requirement described in 3.6.5 *Usability* regarding grouping functionality. Each tab will be displayed and explained below.

#### 3.9.2.1. General Tab



Figure 16: A mock-up of the General Ribbon tab.

Button	Comments
New Project	This button opens a dialog to create a new project.
Save Project	This button saves the currently open project
Load Project	This button opens a dialog to load a previously saved project
Export Scenario	This button opens a dialog to export the currently selected scenario on the canvas to a file
Import Scenario	This button opens a dialog to import a previously exported scenario into the currently open project
Print Model	This button opens a dialog to print the model on the canvas to a PDF file or a printer

### 3.9.2.2. Model Tab



Figure 17: A mock-up of the Model Ribbon tab.

Button	Comments
New Scenario	This button opens the dialog to create a new scenario
Edit Scenario	This button opens the settings window for the selected scenario
Delete Scenario	This button opens a confirmation box to delete the selected scenario
Duplicate Scenario	This button creates a copy of the selected scenario and places it on the canvas
New Component	This button opens the dialog to create a new component in the toolbox.
Edit Component	This button opens the settings window for the selected component. This works both for components in the toolbox and for components on the canvas. If a component in the toolbox is changed the related components on the canvas will also be changed
Delete Component	This button opens a confirmation box to delete the selected component. This works both for component in the toolbox and for components on the canvas.
Undo	This button undoes the last action
Redo	This button redoes the last undone action
Cut	This button removes the currently selected object from the canvas and puts it in the system clipboard
Copy	This button copies the currently selected object on the canvas and puts it in the system clipboard
Paste	This button pastes the object in the clipboard onto the canvas if possible

### 3.9.2.3. Simulation Tab

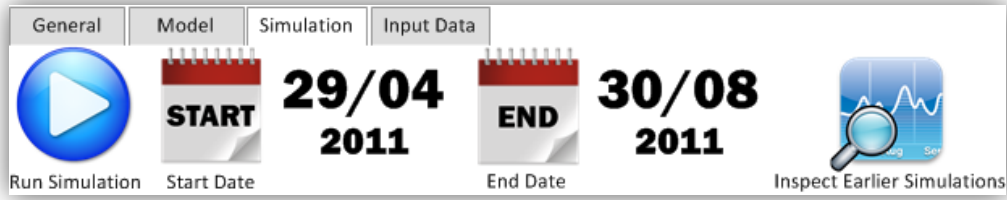


Figure 18: A mock-up of the Simulation Ribbon tab.

Button	Comments
Run Simulation	This button starts the simulation of the model on the canvas using the currently selected energy prices and weather forecasts from the input data ribbon and the start and end dates from the simulation ribbon. When the simulation finishes the simulation window is shown
Start Date	This button opens a calendar in which the user can select a start date for the simulation. The date shown to the right of the button is the currently selected start date.
End Date	This button opens a calendar in which the user can select an end date for the simulation. The date shown to the right of the button is the currently selected end date.
Inspect Earlier Simulations	This button opens a dialog in which the user can see previously run simulations for the open project.

### 3.9.2.4. Input Tab

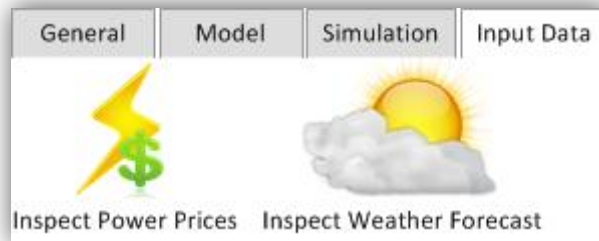


Figure 19: A mock-up of the Input Data Ribbon tab.

Button	Comments
Inspect Power Prices	This button opens a window in which the user can see the imported energy prices and edit these.
Inspect Weather Forecast	This button opens a window in which the user can see the imported weather forecast and edit these. The user can also choose specific historical weather data

### 3.9.3. Reporting Window

The reporting window contains all the information generated by the simulation. It can create charts that show the segments of the generated data required by the user. The window is divided into three main parts; the top part contains an overview of the performed simulation; the second part contains buttons that can be used to define custom charts and the last part shows the created charts.

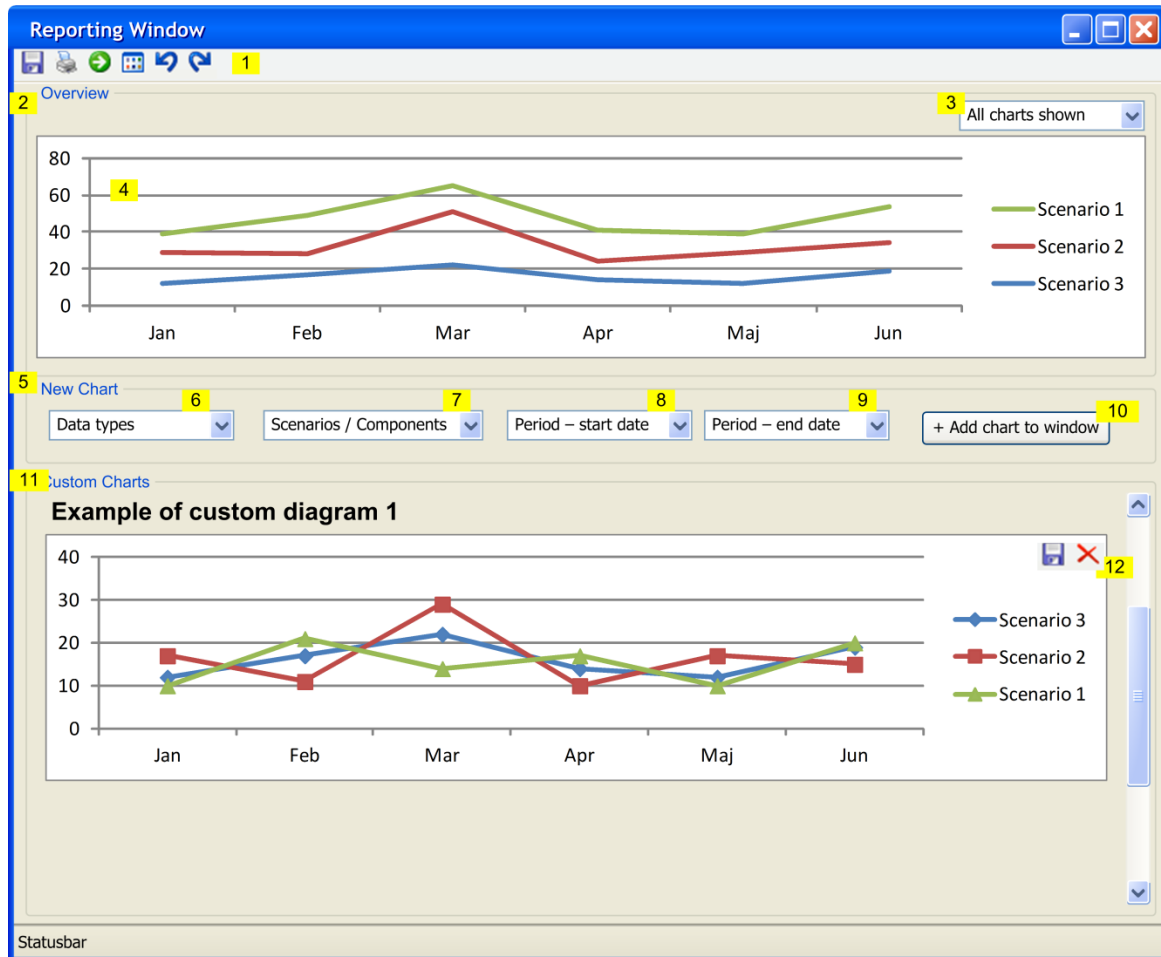


Figure 20: Mockup of the reporting window.

Note	Comments
1	Top menu containing functionality for saving, printing, exporting, inspecting components, undo/redo and recreating model from the simulation.
2	Overview section, where an overview of the generated information from the simulation is shown.
3	A drop down menu that gives possibilities for filtering out scenarios from the overview chart.
4	Chart area for the overview of the simulation
5	Control area for adding a new charts to the bottom half of the window
6	Selection menu for which data type, e.g. consumption, information about when it was on etc., that the new chart needs to display
7	Selection menu for scenarios or components that the new chart need to display



8	The start date for the period the new chart needs to display data for. Has to be within the period of the simulation and before the end date (9).
9	The end date for the period the new chart needs to display data for. Has to be within the period of the simulation and after the start date (8).
10	Button for adding new chart to the window. (6) - (9) needs to be filled out correctly
11	Custom chart area. Contains all the charts created with the chart adding tool (4).
12	A button for removing the chart from the window. A confirmation box will appear before deleting.

### 3.9.4. Settings Window

When the user wishes to edit a component or scenario a window should be displayed showing the properties that can be edited. A mock-up of a sample setting window can be seen in Figure 21. The settings windows will be dynamically populated with controls depending on the properties of the chosen component or scenario and therefore the window below shows examples of how various controls could look. The dynamic settings window can be used for changing settings on scenarios, components on the canvas and for creating elements like components for the toolbox.

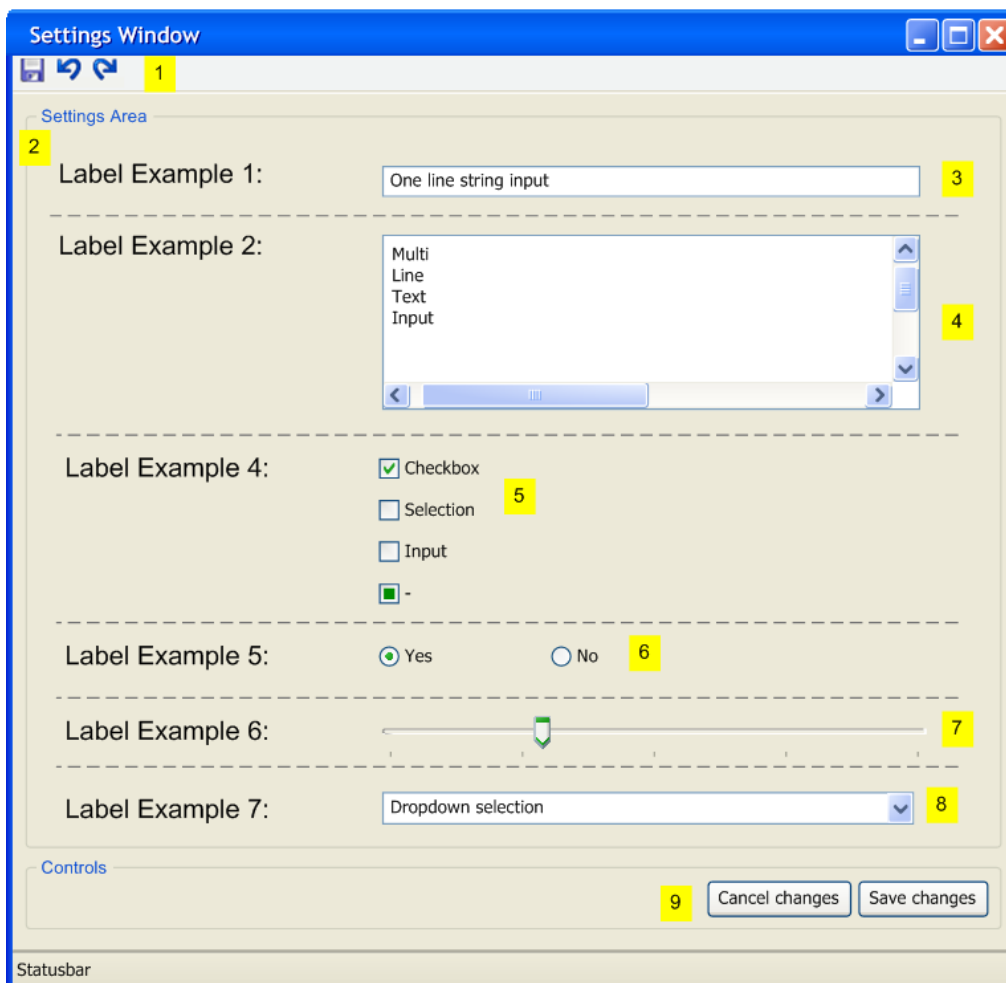


Figure 21: Mockup of the Settings window.

Note	Comment
1	Top menu containing possibilities for saving, undo/redo.
2	Settings area, containing all the input fields relevant to the element that is being edited/created.
3	Example of a one line text input field. Suitable for titles, names and the like.
4	Example of a multi-line text input field. Suitable for descriptions, code or formulas.
5	Example of checkboxes. Suitable for list selections for categories.
6	Example of radio buttons. Suitable for true/false values.
7	Example of a slider selector. Suitable for range selection.
8	Example of a dropdown selector. Suitable for properties that have a limited set of possible values as suggested by the usability ISO standard described in <i>3.6.5 Usability</i> [65: ISO].
9	Control area contains a cancel and save button.

### 3.10. Conclusion

The target audiences of the application were defined as three main groups; the consumer, advisor and educator/educatee. The three groups have different interests in the solution and have different preferences when using it. An analysis of these interests was made and three main focus areas (Saving money, user friendly and comparisons) were discovered and used as main pillars in this chapter and in the design phase of the program. Based on the analysis of the target audience the overall purposes of the application were identified and the components in a home that are relevant to the simulation were identified. This description led to the identification of a number of non-functional and functional requirements that define the foundation and direction of the application. The three main requirements for the solution are as follows:

- Create one or more models of a house (a scenario)
- Create custom components to use in the modeling
- Simulate energy consumption of modeled scenarios

Based on the overall direction and functionality of the application a range of use cases were developed that describe each of the functionalities and how they are performed by the user. These use cases led to the creation of mock-ups of some desired graphical user interfaces that provide the graphical direction for the design presented in the next chapter.

## 4. Design

---

This chapter describes the design of the application on the basis of the identified requirements in chapter 3. First an overview of the overall design will be presented in which the chosen architecture, design patterns, frameworks & toolkits will be described as well as the overall connections between the modules that make up the application. Then each module will be examined more closely and the detailed design of these will be described.

### 4.1. Overall Design

Before going into detailed design decisions in the program, it is important to establish some general design guidelines that should be the base of the solution. Such decisions are not trivial and can have significant implications for the end result. Therefore the detailed requirement analysis performed in the previous chapter is very important and provides some useful building blocks to base these decisions on.

#### 4.1.1. Architecture

This section describes the main architectural decisions made during the overall design of the application. These decisions are: Choice of desktop vs. client-server setup, modularization, event-driven architecture and programming language. The decisions will primarily be based on the non-functional requirements specified in 3.6.

##### 4.1.1.1. Desktop vs. Client-Server

The choice between a desktop application and a browser application highly depends on the target audience of the application. In this case the target audience is a combination of salesmen, end consumers and educators (see 3.1 *Target Audience*). These target audiences have different preferences in this regard. For a salesman it is likely that he will prefer a desktop application as it allows him to go offline while visiting customers as well as providing a more fluent experience when the internet connection is poor. Consumers, however, are likely to prefer a browser version as it makes it possible for them to use the program anywhere without installing anything on their computers. For education purposes both application types seem viable. This tool will be created as a desktop application, which means that it does not have any integration to an application server, but it can have connections to web services that provide occasional data. Desktop application has been chosen because it fulfills the requirements of half of the target audience while being a simpler architecture. This is an advantage when developing a prototype as there is a considerable amount of work to be put into setting up a client-server-service environment and it introduces additional unnecessary error sources that remove the focus from the actual objective of developing the prototype. Desktop applications also do not require a constant Internet connection, but just need access when retrieving data from services. This time can be minimized by introducing caching. Additionally, desktop applications can relatively easily be converted into a browser version as most of the backend code will be the same and if using the same overall programming platforms (such as WPF for desktop, Silverlight for browser) and the graphical controls will be very similar too.

#### **4.1.1.2. Modularization**

The second main architectural decision is that the solution is going to be strongly module based, increasing the flexibility and extensibility of the application which is described as a key non-functional requirement in 3.6.2 *Extensibility*. By separating the logical parts of the solution, it becomes clearer who is responsible for what functionality. By dividing the solution into sub-components you also provide easier access for further development as you can remove a module responsible for some particular part of the application and replace it with another module without breaking the functionality of the rest of the application. This, however, requires that the application as a whole defines precise interfaces for how modules should behave. Working in strictly separated modules goes very well in hand with the chosen iterative prototyping development methodology and the prototype concepts. Each iteration can result in a working prototype and the upcoming iterations extend the prototype with more complex functionalities in new modules.

#### **4.1.1.3. Event-Driven Architecture**

Making the program event-driven goes very well in hand with the module based design decision as it enables a looser coupling between the modules. It also allows future extensions to listen to existing module'' events and react on these without having a direct reference to the module. Events also make it easier to notify the user interface when changes occur and thereby update what is displayed while still maintaining a clean separation between the model and the graphical representation of the model.

#### **4.1.1.4. Programming Language**

The last main architectural decision is the choice of programming language. The program will be developed in the C# .NET framework using Windows Presentation Foundation (WPF) as the programming modeling framework. C#.NET has been chosen as it has a strong community as well as a large codebase and wide range of custom built libraries available created by third parties which makes the development process a lot faster. Also, C#.NET is widely used in the industry which makes it easier to develop the prototype into an actual end product later on. The choice of WPF comes from the design decision to create a desktop application in which case WPF is the go-to framework in .NET. WPF provides graphical controls that are similar to those found in Windows and other Microsoft products and are therefore recognizable to the user. Another reason to choose C#.NET and WPF is that the alternatives available to make rich desktop applications are limited. The only real alternatives are C++ and Java, neither of which have the same industrial and communal support when it comes to desktop applications and they require considerably more work when creating rich graphical applications.

### **4.1.2. Design Patterns**

Design patterns are standardized ways of performing common actions in the application. Design patterns can be very useful to ensure consistency in the design of the application and using existing design patterns ensures that the approach being used is thoroughly tested before and can be implemented reasonably. Additionally the use of design patterns can decrease the learning curve for future developers as the description of designs and implementations can be done by referring to commonly known design patterns. Design patterns can be divided into four categories: Structural, behavioral, creational and UI design patterns [72: Garofalo]. This section will describe the overall design patterns that will be used in this thesis and describe why they have been chosen.

#### 4.1.2.1. Model-View-ViewModel (MVVM)

The most commonly used UI design pattern in WPF is the Model-View-ViewModel (MVVM). It was introduced in 2005 by John Gossman [73: Gossman] and is a .NET variation of the traditional Model-View-Presenter (MVP) or Model-View-Controller (MVC) patterns. The *Model* describes the underlying data in the form of business entities based on databases, web services etc. The model is typically application independent and can be reused by the business for other applications. The *Views* describe the GUI components of the application. Each view describes a specific part of the application and can be e.g. a window, an area in a window or a graphical representation of an entity. In MVVM the views must be created using a declarative format which in .NET is XAML. The view should be free of any model specific details as well as state information, but should contain templates and styles as well as the actual GUI controls. The connection between the views and the model is the *View Model* which provides the views state. It should contain commands used by the view to interact with the model (i.e. all behavioral logic), value converters for model data that does not easily map to the GUI types and the general state of the views [73: Gossman]. There is often a 1:1 relation between the views and the view models, but this is not a requirement of the design pattern. The view should be connected to the view model using weak connections which in XAML are represented as *data bindings*. All communication between views and view models should be done using *Commands*. WPF already offers commanding through the *ICommand* interface and can be used to easily to invoke functionality in the view models in a loosely coupled way [74: Microsoft].

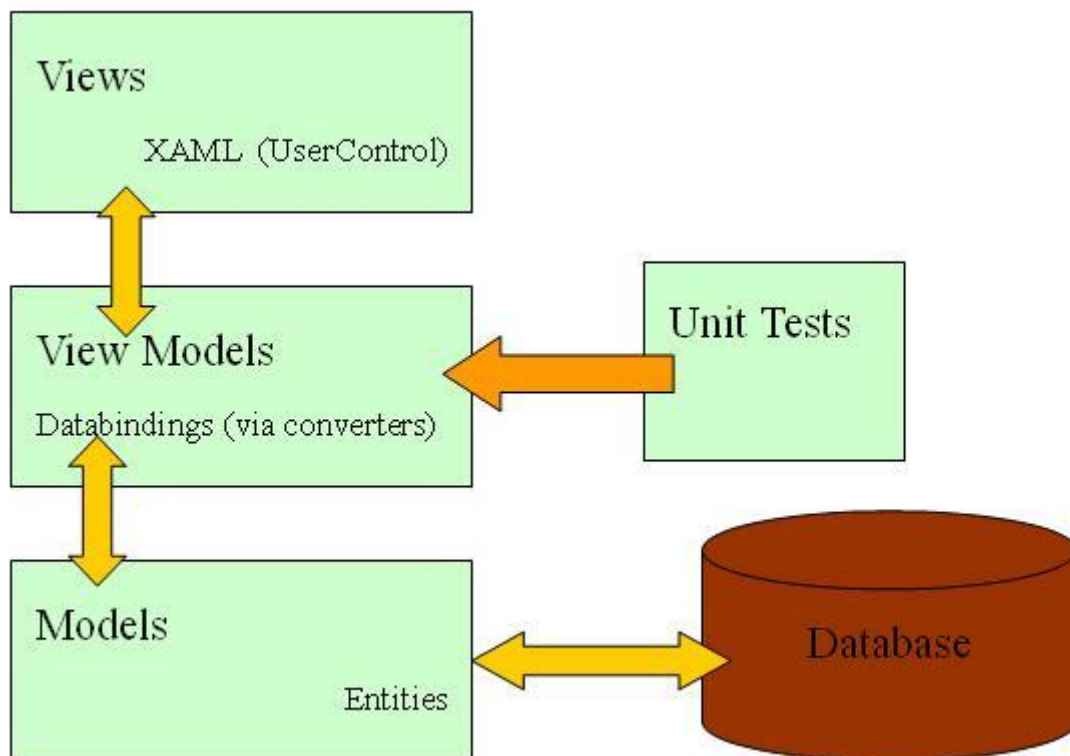


Figure 22: A graphical representation of the MVVM design pattern. Source: (Li, 2010)

The MVVM design pattern has been chosen because it provides a very good separation between the view, the logic and the model data and is highly adapted to the WPF framework. In fact WPF is to a large degree developed to make the implementation of the MVVM pattern as easy as possible [75: Smith]. Additionally it

provides a structure that is highly testable as all logic can be tested in the view model separate from the view and the model can be tested separately from the view model [76: McCarter].

#### 4.1.2.2. *Command Pattern*

The Command pattern is a behavioral design pattern in which actions are encapsulated in objects. This pattern is especially used when developing undo/redo functionality as it allows you to encapsulate each action such that it can be stored and used later [72: Garofalo]. As described in use cases 20 and 21 (see 3.8 *Behavioral Requirements (Use Cases)*) undo/redo functionality is required in this application and therefore the command pattern will be used.

#### 4.1.2.3. *Inversion of Control (IoC)*

Inversion of Control (IoC) is a creational design pattern in which the creation of objects and their relations is controlled by an adapter class at runtime [72: Garofalo]. At design time classes communicate through contracts (interfaces) and at runtime the actual implementations of the interfaces will be automatically resolved through the adapter that maps the interfaces to implementations (typically a container), and injected into the classes that use them using Dependency Injection (DI).

IoC will be used because it provides loose coupling between modules (or classes in general) such that modules are robust towards changes in dependency modules as long as the contract (interfaces) is fulfilled. This is in itself desirable, but it also reduces the complexity of the code as creation of objects and their dependencies is handled in a single place (the container), like in the Factory design pattern. IoC also lets each module focus on its task and be independent of other modules which makes it a lot easier to create reusable modules and to replace them without affecting the rest of the application. This is very useful in this thesis as the end result is a prototype that will require further development.

### 4.1.3. Frameworks and Toolkits

In the previous sections a number of design decisions have been made. This includes:

- Modularization
- Event driven design
- Model-View-ViewModel
- Inversion of Control

The implementation of all the decisions above can be assisted by using Microsoft's Prism library [77: Microsoft]. It is a library created to guide the development of large composite applications developed in WPF or Silverlight by implementing functionality to easily use best practice design patterns and achieve loose coupling [78: Microsoft]. The most fundamental concepts of Prism are: The bootstrapper, the *IModule* interface, *regions* and the Inversion-of-Control-container (IoC-container). The building blocks of Prism are *Modules* (implementations of the *IModule* interface). They describe a logical part of an application and should be completely separated from other modules. Each module contains all the logic and graphical information relevant for the module's assignment and they are stored in a module catalog in which their dependencies on other modules are defined. Modules are not restricted to being graphical units, but can also be services providing data to be used by other modules. If they, however, have a graphical component they can be plugged into *regions* in the overall GUI defined by the designer. This will display the view of the module in the specified area of the window. The composition of the overall GUI can

also be changed completely by moving the regions or assigning modules to other regions thereby giving the GUI a completely new look without even touching the existing modules. This separation between the overall GUI and the module specific GUI components ensures that replacement of modules is very easy [79: Microsoft]. A few examples of graphically similar concepts could be SharePoint's WebParts and the modules in iGoogle.

The *container* in Prism is a concrete implementation of an adapter class described in the Inversion of Control design pattern. The programmer can describe relations and dependencies between interfaces and concrete classes and let the container handle the creation of the objects at runtime. A class can request an object that implements some specific interface and the container will then at runtime return such an object, but the requesting class will only be aware of the information described in the interface [80: Microsoft]. This creates a very weak coupling between the classes as the concrete implementations used by the application can be changed in a single place by registering another class with the instance and then the entire application will use that class instead without any further changes.

Finally the *Bootstrapper* is a class that sets up the application by instantiating all the modules and their dependencies, adding them to the module catalog and configuring the container [80: Microsoft].

As mentioned earlier the modules should be completely separated from each other so any communication between the modules must either be done using Dependency Injection through the centralized container, and thereby only knowing interfaces of other modules, or by using the *EventAggregator* in Prism. This is a class that allows modules to subscribe to and publish events that can be used by other modules in a loosely coupled manner [81: Microsoft].

When comparing the functionality provided by Prism with the overall design decisions of this thesis it can be seen that Prism provides guidelines for each of the major decisions. Modularization is achieved by using modules in Prism as these are logical groupings of functionality and graphical representation that are separated from other logical parts. Event driven design is achieved by using the *EventAggregator* in Prism to perform communication between modules and Inversion of Control is provided by the *Container* class in Prism. Prism is built to provide guidance for MVVM as well and has therefore implemented a range of extended functionality to make the implementation of MVVM as easy as possible such as the new *DelegateCommand* that extends the functionality of the regular *RelayCommand* in WPF [82: Microsoft].

In this application each logical module will be represented by a Prism module. Each of the modules exposes a manager interface in the globally available IoC-container such that other modules can communicate with the module in a loosely coupled way. When messages are to be exchanged between components where it does not make sense to expose a classical .NET event in the interface, Prism's *EventAggregator* will be used to publish the event.

Prism provides two implementations of the container. One implementation is based on the *Unity* framework's approach and the other one uses a *MEF* approach. They both essentially provide the same features, but MEF is focused on making it easier to load third-party modules placed in separate DLLs and exposing limited access to the existing application whereas Unity focuses on improved control internally in the application. MEF does, however, not work great with MVVM due to its focus on using singleton instances which may cause UI failures so this application will use Unity [83: Kava]. It is possible to use a combination of the two where each container type controls the areas they excel at, but for simplicity this application will only use a Unity container.

#### 4.1.4. Modules

Based on the functional requirements and use cases defined in chapter 3 a number of logical groups of functionality were identified. Each group was designed as a separate module and interfaces were defined that exposes the necessary communication between the modules. An overview of the defined modules can be seen in the component diagram below.

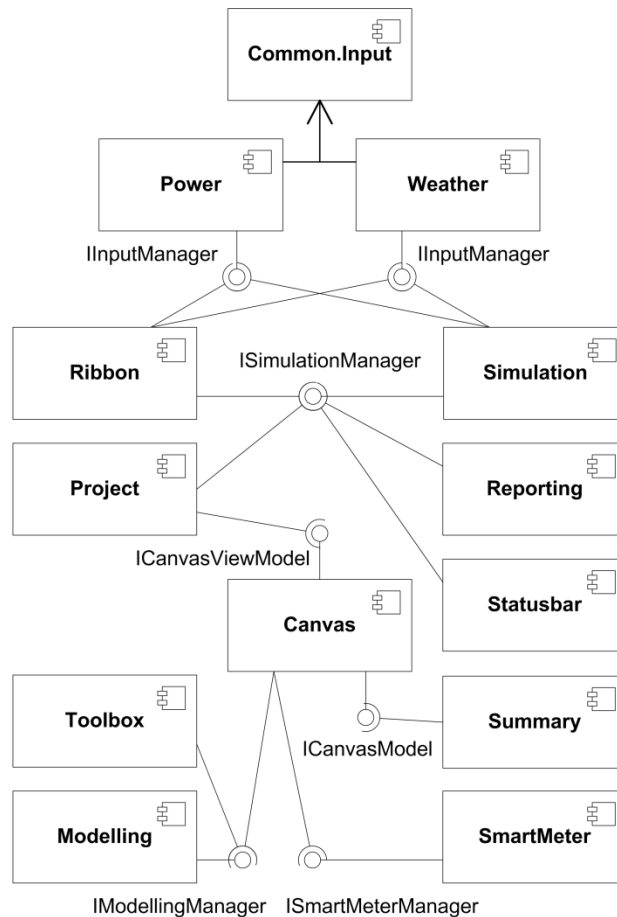


Figure 23: A component diagram showing the interactions between the modules of the application

In the following sections each of the modules in the diagram will be described.

##### Canvas

The main task of this module is to keep track of the elements that the user has put on the canvas. It shall handle the drag and drop actions of elements from the toolbox as well as notifying other modules when elements are added to the canvas. The canvas shall also allow the user to edit the settings of each of the elements on the canvas. This module supports the functional requirement 1: *Create one or more models of a house (a scenario)* and the non-functional requirements of *Customizability* (through editing components and modeling homes) and *Usability* (through drag-and-drop).

##### Modelling

This module contains the definitions of all the modeling elements in the application and their graphical representations. These elements are scenarios as well as electrical devices, DERs and hybrids (the last three collectively called *components* throughout the thesis). When other modules, such as the canvas module,



require instances of modeling classes they are created by this module. The modeling module will support the functional requirements *1: Create one or more models of a house (a scenario)* and *3: Create custom components to use in the modeling* as well as the non-functional requirement of *Customizability* (by enabling creation of new components) and *Extensibility* (by enabling easy implementation of new types, see 4.2.2 *Modelling*).

### **Toolbox**

The task of this module is to present the elements that can be dragged onto the canvas. It shall get the elements through the *Modelling* component and then sort them logically based on their categories. This module supports the functional requirements *1: Create one or more models of a house (a scenario)* and *3: Create custom components to use in the modeling* as well as the non-functional requirements of *Customizability* (through creation of new components) and *Usability* (through drag-and-drop).

### **Ribbon**

The *Ribbon* module is responsible for the overall navigation of the application and contains the menu of the program. By placing the navigation in a separate module it is easy to replace the navigation system with an entirely different menu. The navigation shall bind to globally available commands that other modules implement. This module supports the nonfunctional requirement *Usability* as described in 3.9.2 *Navigation (Ribbons)* as well as enabling a global access to the functional requirement *8: Undo/redo mechanisms*.

### **Common.Input**

As defined by functional requirements *4: Import and edit energy prices* and *5: Import and edit weather data* the application uses two types of input data. The way input data should be handled is largely the same for the two types so this module contains a shared structure for use by input data managers. This also enables future extensions with new data types. Both the *Weather* and *Power* modules inherit from this module. Besides the two functional requirements already mentioned this modules supports the non-functional requirement of *Extensibility* (by enabling new input data types and sources to be added easily).

### **Power**

The *Power* module is responsible for handling everything concerning the power data, such as caching and retrieving the data as well as providing a GUI for editing/viewing the data. This module shall on demand connect to a web service or data scrape a source on the internet for up-to-date power information. This information should then be added to the local data cache to provide faster data access in the future. The interface of this module will be used by the simulation module to request power data when starting simulations. This module supports the functional requirement *4: Import and edit energy prices* and the nonfunctional requirement of *Customizability* (by allowing users to modify the imported power data).

### **Weather**

The *Weather* module is responsible for handling the weather data in the program that is important for the DERs. It performs the same tasks as the Power module and its interface is also used by the simulation module, so it needs to store both the standard data and user customized data sets. This module supports the functional requirement *5: Import and edit weather forecasts* as well as *Customizability* (by enabling the users to modify the imported weather data).

### **Simulation**

The *Simulation* module will contain all logic for performing simulations based on a model provided by the *Canvas* module, energy prices provided by the *Power* module and weather data provided by the *Weather* module. The result of the simulation is stored and passed to the Reporting module. This module supports the functional requirement 2: *Simulate energy consumption of a scenario*.

### **Reporting**

The *Reporting* module provides the functionality of showing the simulation result in charts in a reporting window and providing the user with the means of creating new charts, saving reports, printing reports to PDF or printers etc. This module supports the non-functional requirement of *Report-oriented results* as well as the functional requirement 6: *Create graphical reports*.

### **Summary**

The *Summary* module visualizes an overview of the current content on the canvas. Therefore it has a strong cooperation with the *Canvas* module, but all communication is still done using interfaces and Inversion-of-Control to ensure loose coupling. This module supports the non-functional requirement *Report-oriented results* and the functional requirement 6: *Create graphical reports* by providing graphical representations of the estimated results of the existing scenario during modeling without performing a simulation.

### **Project**

The *Project* module shall be responsible for controlling the concept of *projects* in the program which is the term of a state of the application that can be saved. This includes information about which elements that are currently on the canvas as well as information about performed simulations. This module will fulfill the functional requirement 7: *Save/load files* by enabling the user to save and load these projects. This module has to work together with the canvas module and the simulation module to retrieve the information to store in the project when saving.

### **SmartMeter**

The *SmartMeter* module handles the use of classes that enables integration to real smart meter data. These integrations could e.g. be done through HTTP communication or through a cabled connection, but are not provided in the application.

### **StatusBar**

This module is a simple graphical information channel to the user displayed at the bottom of the main window where the program can inform the user about the state of the program. The module's main task in this program will be to inform the user about the progress of a simulation and therefore the module must communicate with the *Simulation* module.

#### 4.1.4.1. Requirements-Module Mapping

As shown above the modules have been designed with close attention to the defined functional and non-functional requirements (see 3.6 *Non-Functional Requirements* and 3.7 *Functional Requirements*). Table 7 sums up the relationship between the design of the modules described above and the most important requirements described in 3.6 and 3.7.

Requirement	Modules
<b>Nonfunctional</b>	
Customizability	Canvas, Modelling, Toolbox. Power, Weather
Extensibility	Modelling, Common.Input
Reporting oriented result	Reporting, Summary
Usability	Canvas, Toolbox, Ribbon
<b>Functional</b>	
Create one or more models of a house (a scenario)	Canvas, Modelling, Toolbox
Create custom components to use in the modeling	Modelling, Toolbox
Simulate energy consumption of a scenario	Simulation
Create graphical reports	Reporting, Summary
Undo/redo mechanisms	Ribbon
Import and edit energy prices	Common.Input, Power
Import and edit weather forecasts	Common.Input, Weather
Save/load files	Project

Table 7: The relationship between the requirements and the designed components.

#### 4.1.5. Interfaces and Abstract Classes

A general design decisions worth mentioning that is used frequently throughout the application is the application's heavy use of polymorphism which will be done using inheritance and a wide range of interfaces. This will help fulfilling the identified requirement of *extensibility* as the application does not need to know the exact class implementations being used, but can simply base the computations on super classes or interfaces. This way the application can be extended with new implementations without breaking the existing functionality. Polymorphism is also a central concept in Inversion of Control where the classes using the container do not know about the implementations of the classes they resolve from the container and as such it fits very well into the chosen framework.

When using deep hierarchies of interfaces, the class implementing the most specific interfaces may have to implement a lot of functionality. If multiple classes implement these interfaces and if they in many aspects work similarly it will result in a considerable amount of duplicated code. To counter this, a range of abstract classes have been created that contain the implementations of functionality that should be the same for all implementations of a specific interface. All such functionality will of course be overridable by the classes that extend them, but they provide a default behavior that can be used. This also makes extending the application considerably easier as existing implementations can be inherited and only the functionality that differs needs to be implemented for the new classes to work with the application. So instead of implementing the interfaces directly the new classes should in many cases extend the existing abstract classes and only override the methods that are required (these will be marked as *abstract*, thus forcing the new class to override them).

## 4.2. Detailed Design

The previous section described the high-level design of the application. This section will describe how these overall design decisions and modules can be designed to fulfill the defined requirements. The application contains four main areas of functionality:

- Modeling
- Input data handling
- Simulation
- Data visualization

All features in the application either relate to one of these essential areas or provide general functionality required to make the work and provide essential features such as navigation, saving/loading etc.

In this section each of these categories will be examined and the modules connected to each of these areas will be described. The modules' important interfaces, classes and responsibilities will be described as well as diagrams of the most essential flows in the application.

### 4.2.1. General Functionality

The application contains a range of functionality that does not directly relate to the four main areas of the application, but is required to make the rest of the application work correctly. The modules in this category are: Ribbon, Project, Shell and a number of Common projects.

When creating a module-based solution loose coupling is very important. To make this possible and still enable the modules to communicate, the solution needs some global shared resources that every module of the system can access. The global resources will be made as separate projects within the solution based on which functionality group the resources are connected to. These projects will be named *Common.\** and be placed separately from the modules. By separating the global resources based on functionality you ensure that the modules that use them only have access to the resources that are required to perform the tasks they are responsible for. The common projects will need to contain things that can be reused in multiple places in the program such as:

1. All the interfaces that defines the communication between the modules
2. General class implementation for resource access, for example to images
3. Globally shared commands
4. Enumerations
5. Global events that need to be sent between modules
6. Generic behaviors
7. Converters for XAML
8. Custom controls for XAML

As described in 4.1.2.2 *Command Pattern* the solution is implementing the command pattern and some of the commands must be accessible by multiple modules, e.g. the undo/redo functionality. Therefore a number of globally available *composite commands* have been defined in the shared resources. Each module can then create concrete implementations of what they want to execute when the command is invoked

and register these commands with the global composite commands. All registered commands will then be invoked when the composite command is invoked. This way the command can be available globally while the implementation resides in the module to which the functionality belongs. This also enables the application to use the same commands from multiple places in the application and thereby enabling the user to access the functionality from all places in which it makes sense.

The four *Common-projects* are listed below with a description of them along with a list of which modules use them.

- **Common**  
This project contains all the interfaces, classes and resources required by all modules. This project is used by the modules: *Canvas, Modelling, Power, Project, Reporting, Ribbon, Simulation, SmartMeter, Statusbar, Summary, Toolbox and Weather*
- **Common.Input**  
This project contains all the interfaces and classes required for integrating real life data such as power and weather data in the application. This project is used by the modules: *Power and Weather*
- **Common.Modelling**  
This project contains all the interfaces and classes required by modules that contribute to the modelling functionalities. This project is used by the modules: *Canvas, Modelling, Project, Reporting, Ribbon and Toolbox*
- **Common.Simulation**  
This project contains all the interfaces and classes required by modules that contribute to the simulation functionalities. This project is used by the modules: *Power, Project, Ribbon, Simulation, Statusbar and Weather*

#### 4.2.1.1. *The Shell*

The chosen module structure (Prism) requires the solution to have a base/shell project that is the start-up project of the solution. This project will be responsible for initializing all the modules, initializing settings and creating the IoC-container that can be used for resolving types or instances (see 4.1.2.3 *Inversion of Control (IoC)*). The shell module will contain a catalogue of all the active modules in the program and will initialize them in a correct order based on the defined dependencies between the modules. All these procedures are defined in the Prism library [84: Microsoft] and are therefore well-defined for the implementation.

#### 4.2.1.2. *Navigation*

As mentioned in the navigation mock-up section, 3.9.2 *Navigation (Ribbons)*, the ribbon navigation provides the user with a task divided navigation, where all the related functionality is grouped together. This gives the user easier access to their goal. The Microsoft Ribbon Library [85: Microsoft] is chosen for the task, as it provides a complete set of WPF controls that contains all the functionalities required of a ribbon. The navigation can easily access the earlier mentioned global commands and bind to these through data binding. The ribbon implementation shall be placed in a separate module and the view displayed in a prism region by the region-manager, so that it can be replaced with a new implementation or new navigation concept if later required.

### 4.2.1.3. Projects

For making the program usable from session to session a *project* concept needs to be implemented so the user can save and load projects. A project module will be responsible for handling this. The project manager keeps track of the state of the program and if the user requests a load/new command, it informs the user about unsaved changes if any. It also initializes the program based on the data coming from a loaded project, by notifying the other modules of the changed data through events. The last task is to keep track of the latest opened projects, so that they can be opened quickly through the ribbon menu. An interface has been created so that other modules can invoke the project functionalities, through the IoC-container. The *IProjectManager* interface can be seen below.

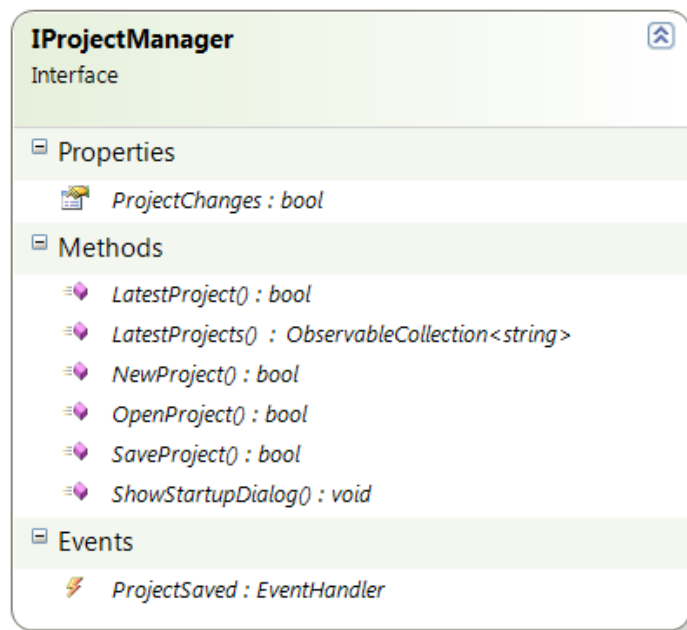


Figure 24: Class diagram of the *IProjectManager* interface.

## 4.2.2. Modelling

The modelling section of the application is one of the primary areas of functionality and one of the areas that differentiates the application from other simulation tools. A number of modules are related to this area of functionality such as Modelling, Canvas, Toolbox and SmartMeter.

As described in the requirements chapter (3.2 *Main Purposes of the Program*) the modelling allows the user to create a home (called a scenario) and add a number of components such as DERs, appliances etc. to the home. Each component can be configured separately. The available components will be displayed in a toolbox and dragging them onto a canvas adds them to the home. It is also possible to add new components to the toolbox.

There are a lot of different component types that all work differently, but they have some similarities with the other components and the scenarios in some situations. Therefore a hierarchy of classes and interfaces has been designed that through polymorphism allows a high amount of reusability of the code while still providing sufficient customizability and specializability of each component. This hierarchy will be explained in the coming sections.

#### 4.2.2.1. Models

As mentioned in 4.1.2.1 *Model-View-ViewModel (MVVM)* the application uses MVVM as a UI design pattern. Therefore separate class hierarchies have been developed for the models and the view models. The class hierarchy for the model classes can be seen in Figure 25. At the model level the scenarios (homes) and components (elements in the home) do not have anything in common and are therefore two separate hierarchies.

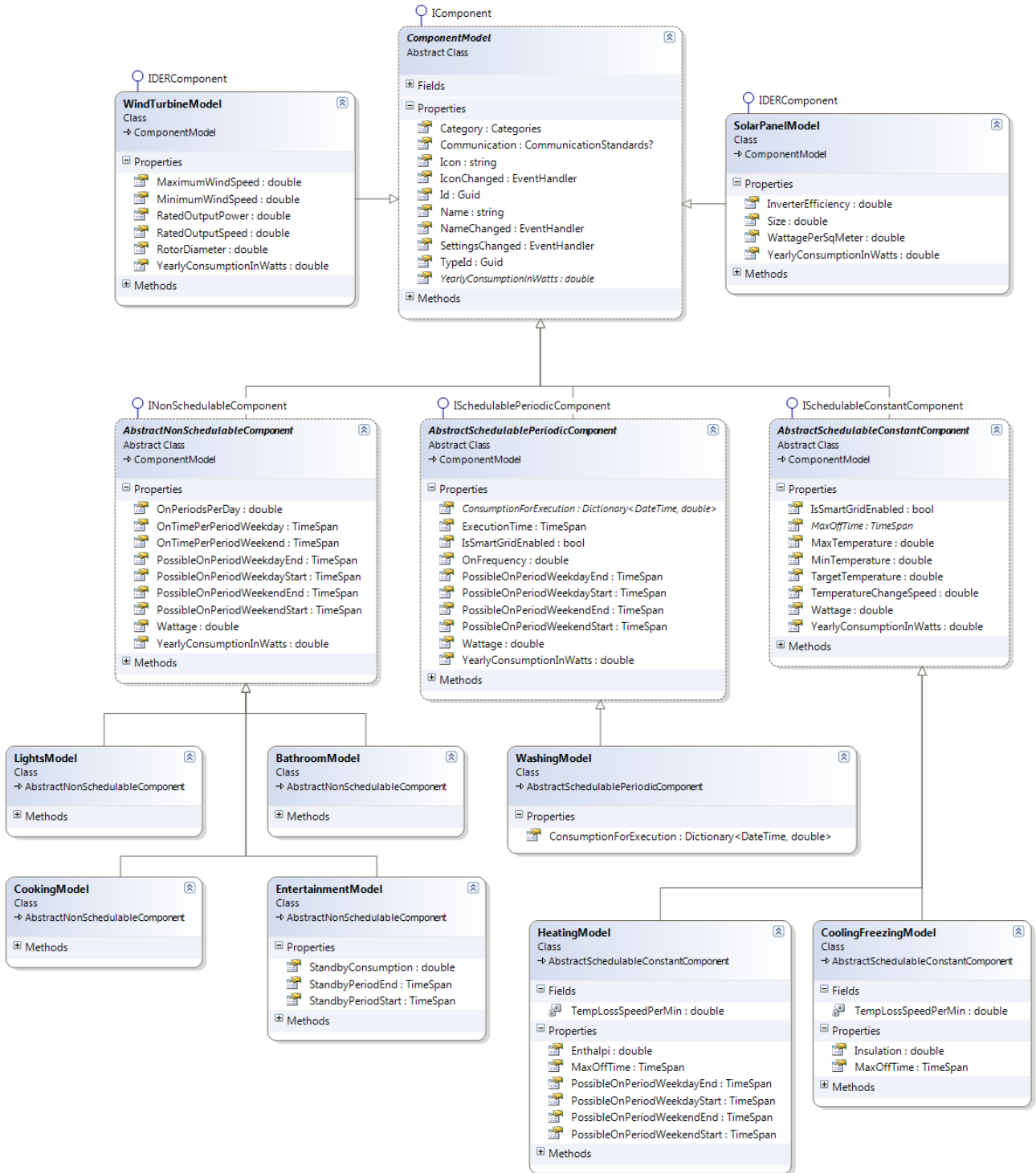


Figure 25: Class diagram of the model classes describing components.

The most general class in the model hierarchy is the *ComponentModel* which is an abstract implementation of the *IComponent* interface that describes the basic functionalities that all components should have. The three abstract classes that derive from *ComponentModel* describe three groups of components that within each group behave similarly overall.

*AbstractNonSchedulableComponent* describe components for which it does not make sense to schedule their energy consumption as they must run when the consumer needs them. This includes lights, computers, TVs, cooking equipment etc. This group of components is described in *3.4 Modeling Components*. These components are then further divided into concrete implementations that describe how each subgroup (cooking, lights, entertainment etc.) actually consumes energy.

*AbstractSchedulableConstantComponent* describes components that can be scheduled and that are consuming energy in a more or less constant fashion. This includes components such as radiators, refrigerators, freezers etc. This group of components is described in *3.4 Modeling Components*. The concrete implementations of the interfaces differ between heating- and cooling components.

The last abstract class inheriting from *ComponentModel* is *AbstractSchedulablePeriodicComponent* and it describes elements that can be scheduled based on the energy prices and only run periodically. This group of components is described in *3.4 Modeling Components*. Currently this only includes one concrete model which is the *WashingModel* that describes the functionality of washing machines, dish washers and dryers.

Finally two concrete models inherit directly from *ComponentModel*. These are the two DER models, *SolarPanelModel* and *WindTurbineModel*, as they do not share any functionality worth creating an abstract class for.

All scenario types supported by the application share the same functionality and are therefore instances of the same model class. This means that the class hierarchy for the scenario is very simple; it is simply an interface called *IScenario* that is implemented by a class called *ScenarioModel*. The scenario contains a collection of *IComponents* which are the components that are currently connected to the scenario on the canvas, see *Figure 26*.

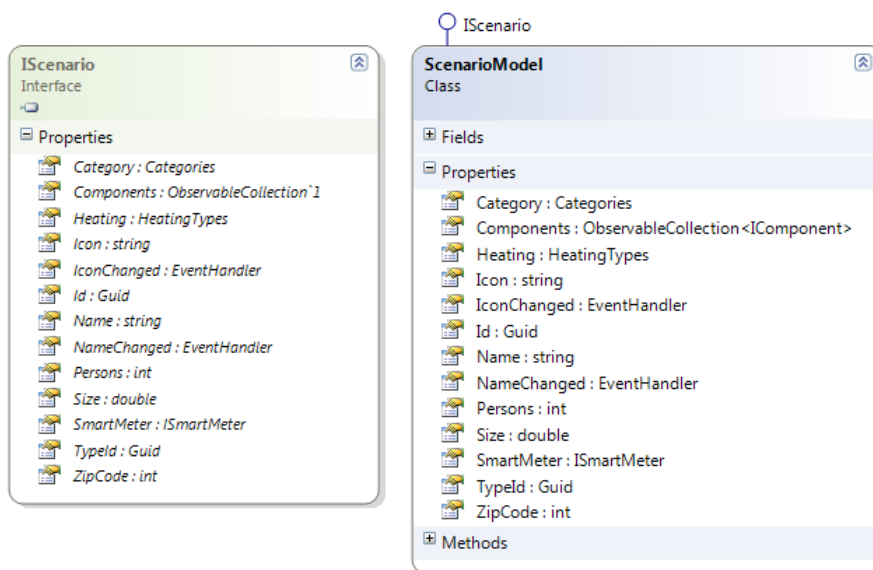


Figure 26: The *IScenario* interface and *ScenarioModel* class.



#### 4.2.2.2. Modelling View Models

The view models of the modeling models that were described in the previous section (4.2.2.1) serve the purpose of exposing the functionality required to display the models properly in a view. Therefore they have a somewhat different inheritance hierarchy which is based on graphical similarities rather than model properties. The hierarchy can be described at two levels: The interfaces and the classes (abstract as well as concrete). The interface hierarchy can be seen in Figure 27.

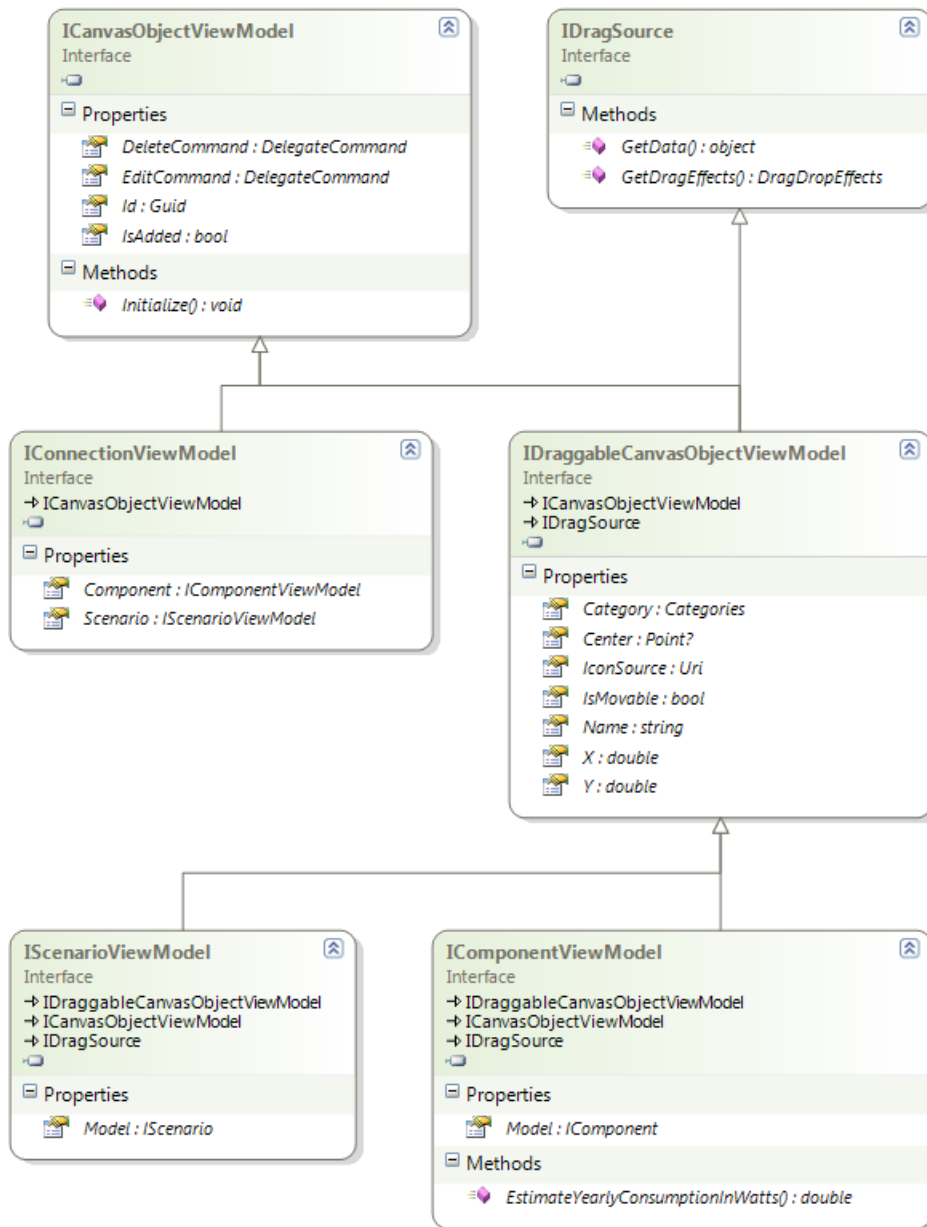


Figure 27: The interfaces used by the view models and the relationships between these.

There are two base interfaces: *IDragSource* and *ICanvasObjectViewModel*. *IDragSource* describes functionality required to allow an object to be dragged on the screen which is required for all elements that need to be dragged onto the canvas. It is, however, not restricted to canvas elements and can be used for other draggable elements later on if such are to be added. The *ICanvasObjectViewModel* interface

describes the basic functionality of an object that should be able to be placed on the canvas. This includes commands to enable deletion and editing, an Id and a flag describing whether the object is currently added to the canvas or not.

When the scenarios and components are added to the canvas they will graphically be connected with connections. These connections do not explicitly exist on the model-level, but are simply relations between the scenario and component models held by the scenario. They need, however, to be explicitly defined at the view model-level as they need to be graphically displayed. This view model is described by the *IConnectionViewModel* interface. As the connections are not explicitly added to the canvas by the user and they shouldn't be movable this view model should not inherit from *IDragSource*. It should, however, be present on the canvas which requires the view model to implement *ICanvasObjectViewModel*.

Both the scenarios and components are draggable elements and therefore a shared interface, *IDraggableCanvasObjectViewModel*, has been defined which combines the *ICanvasObjectViewModel* and *IDragSource* interfaces and adds a few extra properties which makes it easier to handle scenarios and components on the canvas in the same way.

The above-mentioned interfaces are implemented in a number of abstract and concrete classes. The hierarchy can be seen in Figure 28.

The most general view model class is the *BaseViewModel*. It is the base class for all view models in the application and simply provides an implementation of the *INotifyPropertyChanged* interface which allows the view model to notify the connected view of changes using weak bindings. *CanvasObjectViewModel*, *DraggableCanvasObjectViewModel*, *AbstractConnectionViewModel*, *AbstractComponentViewModel* and *AbstractScenarioViewModel* are abstract implementations of the corresponding interfaces described above. As mentioned earlier this makes extending the application with new types considerably easier as the most basic functionality is already implemented and only the component specific functions need to be added.

*ConnectionViewModel* and *ScenarioViewModel* are the concrete implementations for the connections and the scenarios. The component part of the hierarchy requires an extra class, since there are multiple types of components that require different properties, but the component should still be able to function similarly in many aspects. For example a wind turbine requires an attribute stating the maximum wind speeds it can sustain, whereas this information makes no sense for a TV. Therefore an abstract class called *BaseComponentViewModel* is introduced. This class holds the basic properties common to all components, but not held by scenarios, such as their estimated yearly consumption and a communication protocol. Finally, for each of the component model classes a view model is created that inherits from *BaseComponentViewModel* and exposes the properties of the model along with the properties inherited from the abstract classes that deal with graphical representation.

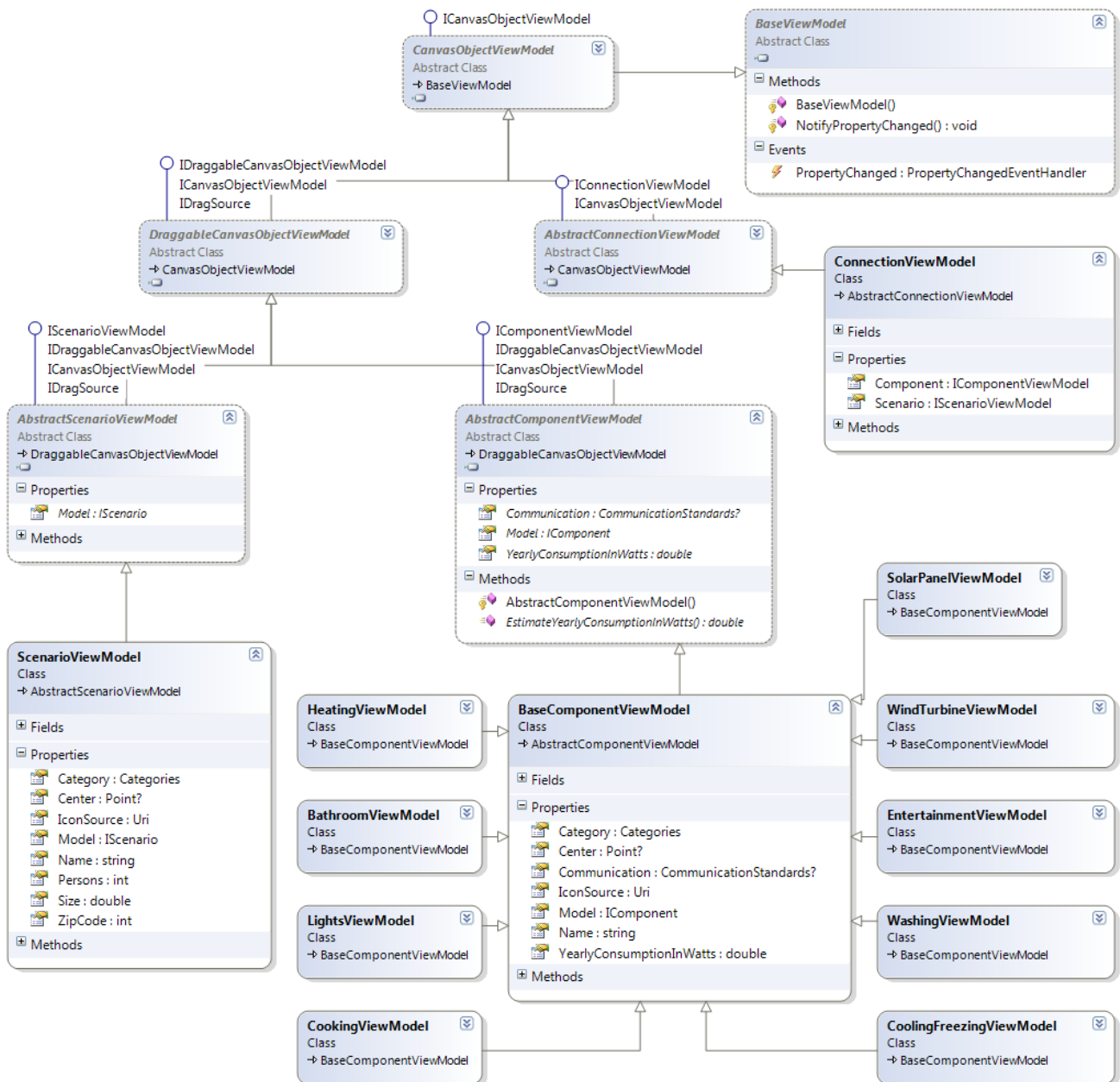


Figure 28: Class diagram of the abstract and concrete view model classes for the modeling module.

#### 4.2.2.3. *IModellingManager*

All the models and view model described above are controlled by a module called *ModellingModule*. The knowledge of these models is exclusive to this module and all usages of the components outside the *ModellingModule* must be based on the provided interfaces (*IComponent*, *IScenario*, *IDERComponent*, *ISchedulableConstantComponent*, *ISchedulablePeriodicComponent* and *INonSchedulableComponent*). The use of the three last interfaces allows other modules to recognize the properties and type of the component (as described in 3.4 *Modeling Components*) while still not knowing the concrete class. This is highly useful for the simulation module in order to simulate a component correctly during simulations.

The modelling module controls the classes and provides instances of the components through the *IModellingManager* interface to other modules. It exposes functionality such as retrieving object instances, creating new components in the toolbox, updating and deleting these and creating a view model for a connection between a scenario and a component. This interface is exposed through the container like all other interfaces that need to be accessible to other modules.

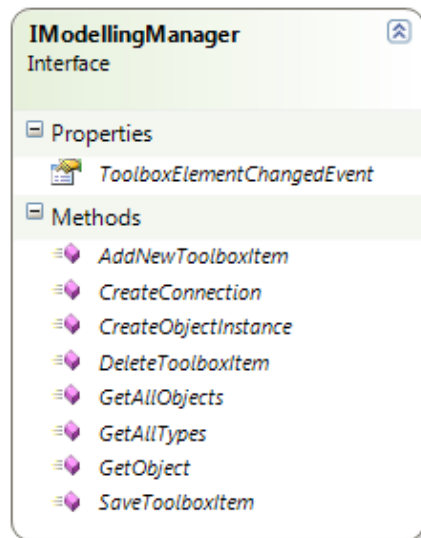


Figure 29: The *IModellingManager* interface used by other modules to retrieve modelling elements.

The *IModellingManager* does not know anything about where the data for the components and scenarios are coming from. Nor does the *ModellingModule* class, which is the implementation of the interface. It simply retrieves data from the class registered in the container fulfilling the *IDataProvider* interface. This way the modelling module is robust to changes in how the data about the components is stored and new ways of storing the data can easily be created by adding a class that implements the *IDataProvider* interface and register it in the container instead of the existing class. The application's default data provider is an *XMLParser* that uses XML to store the information about the components and scenarios.

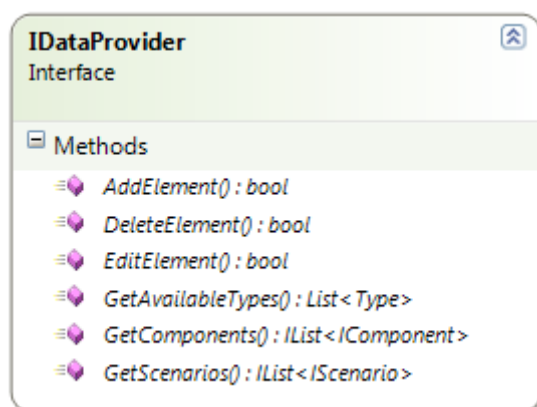


Figure 30: The *IDataProvider* interface used to retrieve data about available components and scenarios from the underlying data store.

#### 4.2.2.4. Toolbox

The toolbox is the left-most region of the main window as seen in the mock-up in the requirement section (see 3.9.1 Main Window). It contains all the elements that the users are allowed to drag onto the canvas. The elements in the toolbox can be seen as templates for the actual elements, as they contain default values which are specific to the given element. These values can then be changed for the individual element once they are added to the canvas. Doing this does not change the value of the elements in the toolbox, but only the specific element on the canvas. The toolbox divides the elements into logical groups such that the users can easily find the elements that they want to add to the model. It is possible for the user to add new and edit existing elements in the toolbox, so that they can create exactly the elements that they require.

The elements in the toolbox consists of the same view model as used on the canvas (retrieved from the modeling module), but they are presented with a different view. When the element is added to the canvas a copy of the view model is created by the modeling module and a view specific to that of the canvas is attached to the view model by the canvas. By copying the view models from the toolbox all elements added to the canvas will receive the default values of the elements in the toolbox such that users do not have to specify all values of the element.

When the toolbox initializes the elements in the toolbox it gets them through the *IModellingManager* which gets them through the *IDataProvider* interface (see section 4.2.2.3 *IModellingManager*). Figure 31 shows a sequence diagram of the flow used when the toolbox is retrieving all of the objects from the modeling manager.

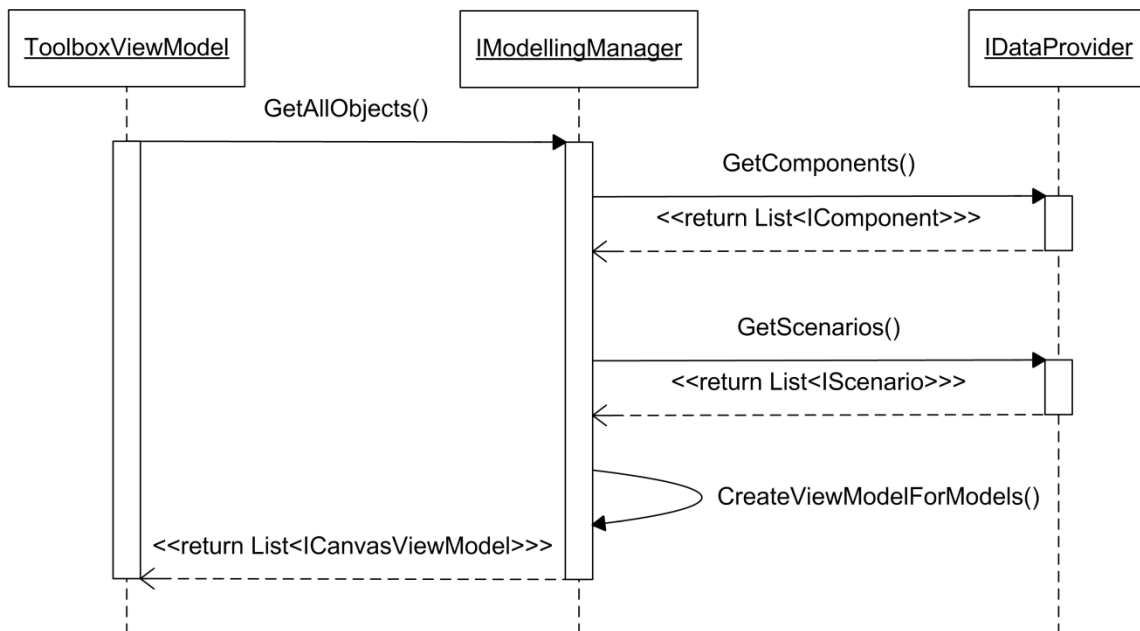


Figure 31: Sequence diagram of showing the flow when the toolbox is getting all objects from the data source.

#### 4.2.2.5. Canvas

The canvas is the graphical container that holds all the scenarios being modeled in the current project. Like all other graphical elements in the MVVM design this element consists of a model, a view model and a view. A separate module, *CanvasModule*, has been created to handle the functionality related to the canvas.

All the scenarios and components are added to the canvas by dragging them from the toolbox onto the canvas. In order to be able to drop the elements on the canvas the canvas must first allow this and specify what should happen when an element is dropped. The created interface *IDropTarget* specifies what a class must implement in order to allow elements to be dropped onto it and therefore this interface is implemented by the *CanvasViewModel*. The *CanvasViewModel* also implements the *ICanvasViewModel* interface. This interface has been created in order to allow other modules to interact with the active canvas. This is currently used to let the Project Manager (see 4.2.1.3 *Projects*) replace the content on the canvas when loading a project. The interface does, however not expose any information about what is on the canvas. This information must be retrieved using the *ICanvasModel*, which is the model representation of the canvas. It contains a collection of *IScenarios* that are the models of the scenarios currently on the canvas and these scenarios then each contain references to the components connected to them. These relations implicitly define the connections displayed between scenarios and components. This way all the data about the elements on the canvas can be stored in a single list of *IScenarios*. The class diagram of the canvas model and view model can be seen in Figure 32.

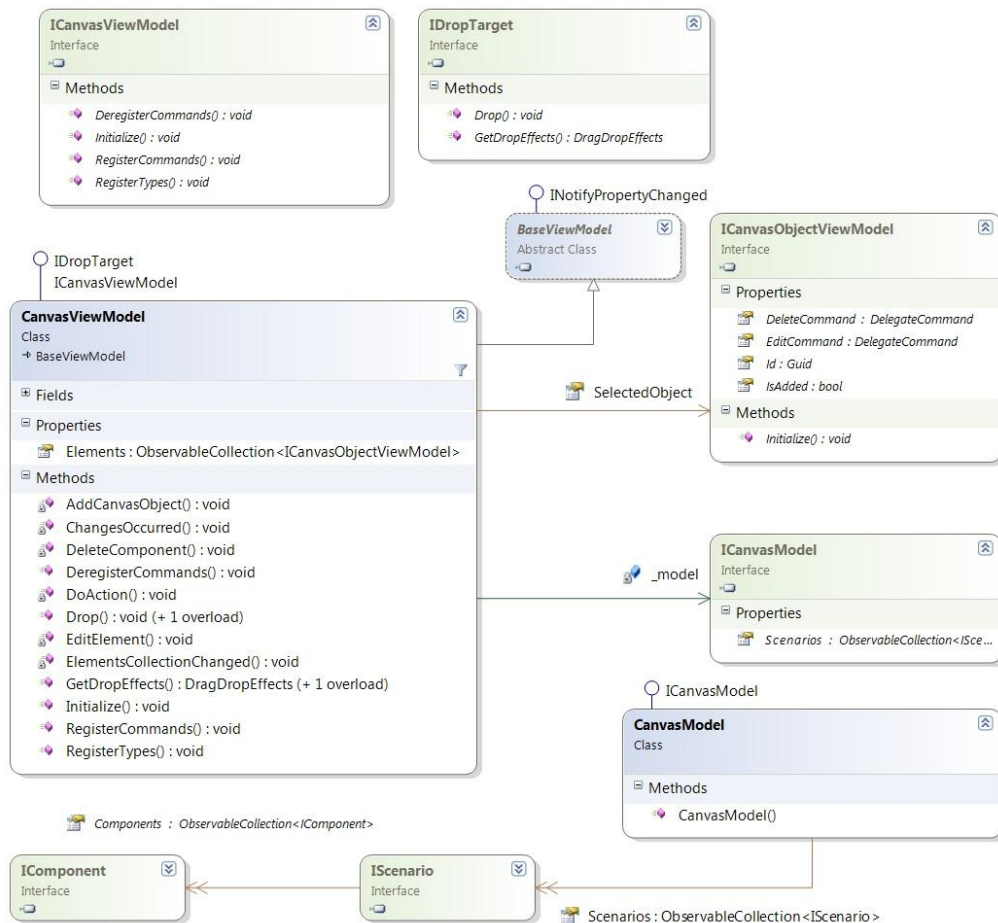


Figure 32: Class diagram of the models and view models used in the canvas module.

The Canvas Module is responsible for all graphical interpretations of the modeling experience on the canvas. This is not only the view and view models of the canvas but also the views of the elements on the canvas. Separate views describe how each of the canvas objects (connections, scenarios and components) should be graphically displayed. By placing the views of the elements in the canvas module and the models and view models in the modeling module, the definition of the components and their types is completely separated from their graphical representation. It also ensures that knowledge of the concrete types of components remains in the modeling module as the canvas module only receives the elements as *ICanvasObjectViewModels* from the Toolbox (see 4.2.2.4 Toolbox).

This makes it very easy to make a completely different modeling experience, by simply replacing the canvas module with a module that specifies alternative views of all the elements and the canvas as well as a canvas model and canvas view model implementation that define how to store the modeled data. As long as the *ICanvasViewModel* and *ICanvasModel* interfaces are implemented it will work with the rest of the application. The process of adding an element to the canvas is illustrated in Figure 33.

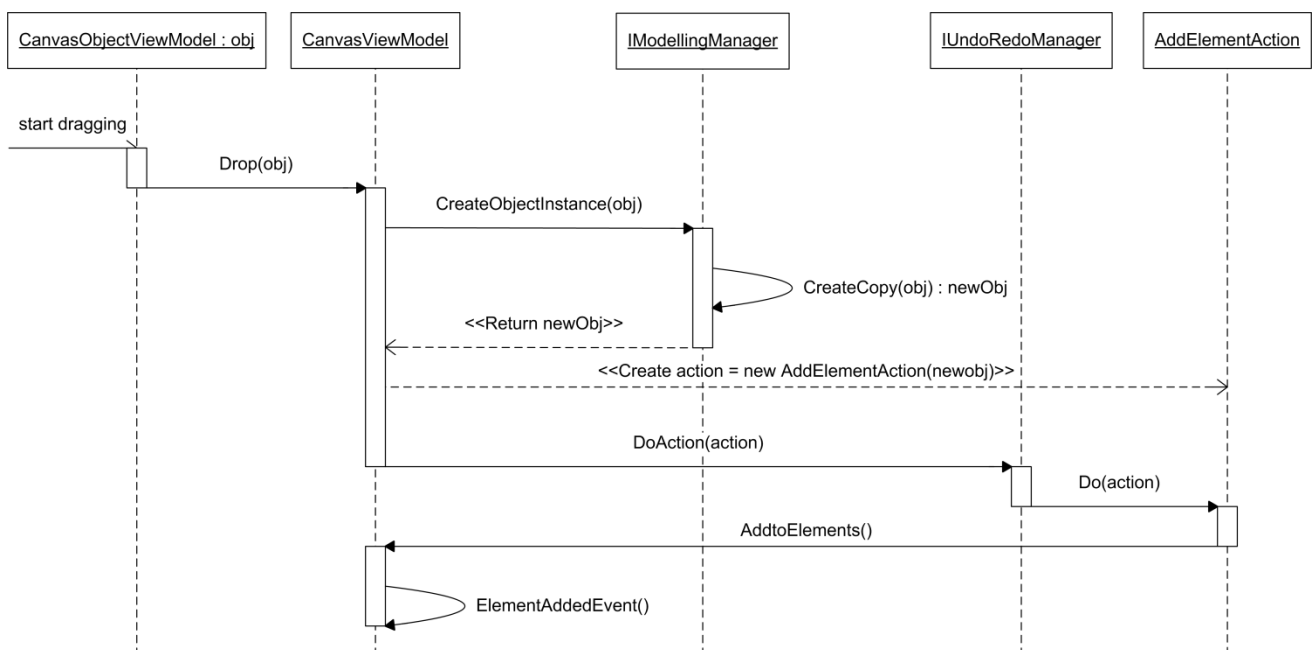


Figure 33: A sequence diagram showing how the operation of adding an element to the canvas is performed between the modules.

When a component or scenario is dropped onto the canvas the *CanvasViewModel* asks the *IModellingManager* for a new canvas object that is a copy of the dropped element. It must be a copy as the actual dropped element belongs in the toolbox and should not be added to the canvas. Based on the newly created object a new *AddElementAction* is created which is executed in the *IUndoRedoManager*. For more information about this see 5.2.3 Undo/Redo.

#### 4.2.2.6. Smart Meter Integration

The application also contains the basic design and preparation for integration with data from a real smart meter. The idea is that data about the home's actual consumption and energy generation as well as energy prices can be imported into the program and used for comparisons with the simulated results. As identified in section 2.2.2.1 *Legislative Restrictions on Privacy of Smart Meter Data* such data is highly sensitive and many countries still have not defined rules regarding management of such data. Therefore the integration of smart meter data should be carefully considered and possibly postponed until the legal issues have been resolved. Therefore the functionality has not been implemented, but preparations have been made to include it if the legal situation is resolved. Currently the application exposes an *ISmartMeter* interface that describes what a smart meter should be able to provide. Implementations of this interface will then have to be made that connect to the actual smart meter and formats the received data into the format specified by the interface.

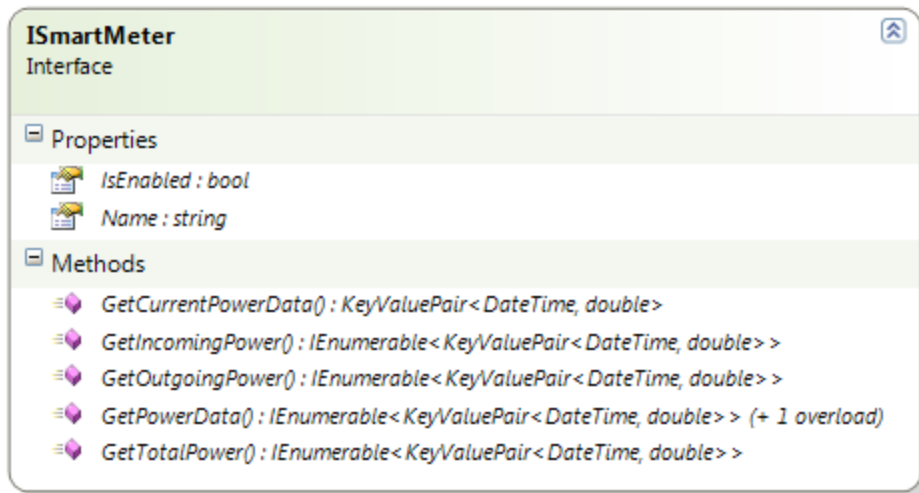


Figure 34: The *ISmartMeter* interface used to integrate with real smart meters.

A smart meter manager has also been designed that exposes all the smart meter implementations available to the application. These are loaded at runtime from DLLs that contain implementations of *ISmartMeter*. These DLLs can be created by third-parties and added to the application after its release by placing them in the designated folder.

#### 4.2.3. Data Handling

For providing a more useful result of the simulation, the application needs to integrate real life data such as the weather information and the energy prices. The idea is that the user can get some real updated data from a data source that they can use in their simulation with their setup and date preferences.

There are two types of data providers: Power and weather. The weather providers need to return information about the sun intensity for calculation with solar cells and wind information for calculations for small wind turbines in a given time interval. The power providers must provide data about the current energy price and if possible the amount of CO<sub>2</sub> that has been emitted generating the energy at the source.

The similarities between the weather and power providers are significant and therefore a shared class structure can easily be created. By having common base classes and interfaces for the two types of input



data it is also considerably easier to extend with new types of input data other than weather and power later on and thereby leveraging the non-functional requirement of *extensibility*. The shared interfaces for the input classes will be placed in the *Common.Simulation* project and the shared abstract implementations of these interfaces are placed in *Common.Input* which is only used by the power and weather modules. This allows the simulation classes to use and resolve instances of the data input classes without knowing the concrete implementations.

There are four main classes in this module that have four specific assignments. *IInputManager* is the module's interface which is responsible for providing access to available data sources of the given input type and is available for all modules through the container. The *IInputSource* is a representation of the source of the data. The *IInputModel* is a wrapping of the actual data gained from the source, which can be either user-edited data (saved data models) or data directly from the source. The *IInputEntry* specifies the actual data representation as a specific point in time and contains information about which point in time the entry is connected to. Data retrieved from the source will typically consist of a list of *IInputEntries*. The *IInputEntry* does not contain any information about the data in the given point. It is up to each input data type's specific module (power and weather) to implement the interface and let that implementation describe which data can be contained in the entry other than the time stamp. Similarly, type specific implementations of *InputSource* and *InputModel* have been created for each type of input data to accommodate for the subtle differences between power and weather data handling.

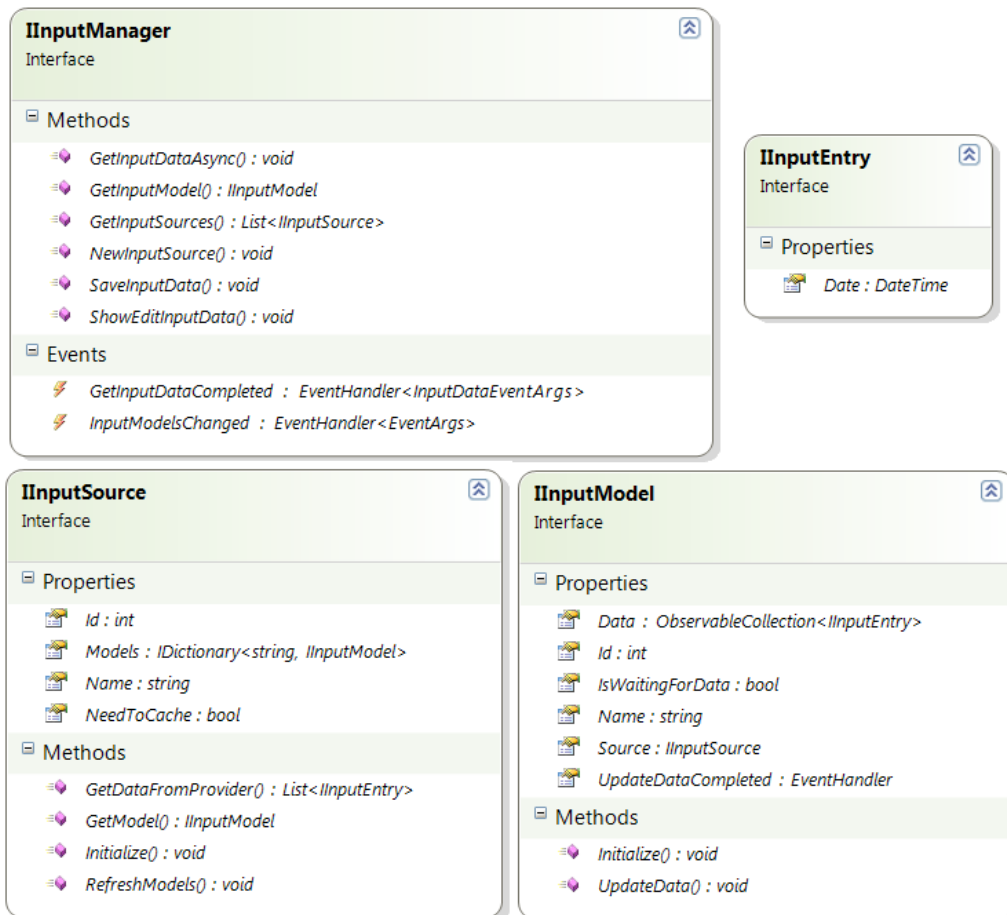


Figure 35: Interfaces for the four shared and implemented classes in *Common.Input*.

It is important that the application can be extended in the future and this is certainly true for input data as the data providers vary depending on geographical location. Danish consumers might use NordPool for power data, whereas this source does not contain any information about prices in the USA and is therefore useless there. Therefore the simple *IInputProvider* interface has been created and any class that implements this can be used as a data provider for the application (See Figure 36 below). This makes it very easy to add new data providers.

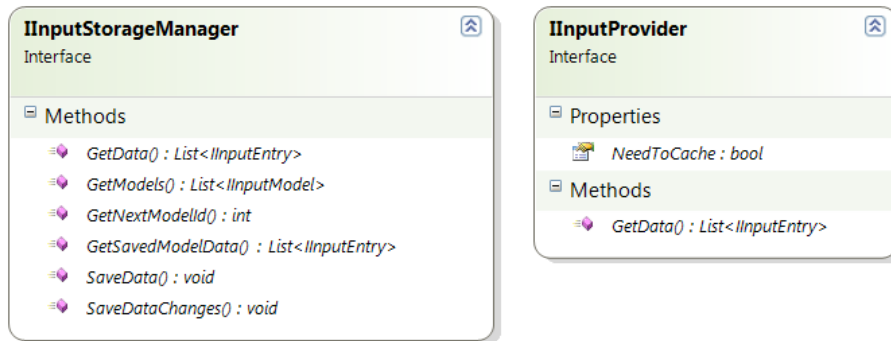


Figure 36: The *IInputStorageManger* and *IInputProvider* interfaces.

As mentioned in the usability section in the requirements chapter (see 3.6.5 Usability) the application should perform well and not introduce unnecessary waiting time for the user. Therefore *IInputManager* uses a caching database, where data that has previously been retrieved can be stored for better performance next time. An *IInputStorageManager* (See Figure 36 above) interface has been created that controls the caching system. If another caching system is preferred it can easily be created by implementing the interface and replacing the registration in the container with the new implementation. A sequence diagram can be seen in Figure 37 that describes the overall flow of how data is retrieved. The system starts off by requesting data from an *IInputModel*, which then communicates to the *IInputManager*. The manager then checks if any of the data is already in the cache and requests the data which is not cached from the source, if any. When it is done fetching the data from the cache and/or the provider, it notifies the *IInputModel* that it has updated the data which in turn notifies the caller.

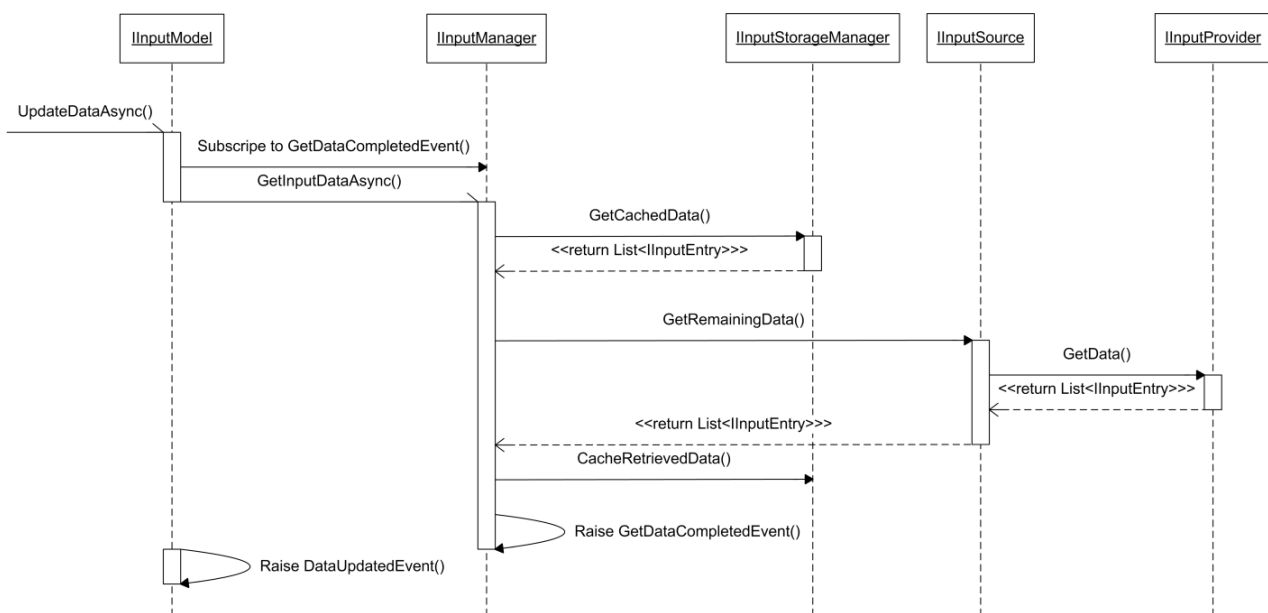


Figure 37: Sequence diagram for a data request from an *IInputModel*.

## 4.2.4. Simulation

As this is a simulation tool an essential part of the application is to perform simulations of the modeled scenarios. In order to separate this from the rest of the logic a module called *SimulationModule* has been created. Simulation is a non-graphical process, the graphical representation of the simulation is provided by the *ReportingModule*, see 4.2.5.1 *Reporting*. Therefore the simulation module is only responsible for performing the computations required to create the simulation. In section 2.2.5 *Controllers for Home Automation* it was identified that the controller units can work in several different ways and that it is only logical to assume that this affects the simulation results. To reflect this it must be possible to choose different simulation strategies when performing the simulation, representing the various controller units available in the market. To avoid having to implement every controller unit available in the world into the program, the application is capable of dynamically loading implementations of controller units that are compiled as DLL-assemblies outside the application and added to a folder in the program's file structure. Such implementations can be made after the application's release using an API (a few of the *Common* namespaces) and included in the application at any point in time.

A controller unit is represented by an interface called *IControllerUnit*, see Figure 38. The interface is very simple and requires only a name for the controller unit and a *Run* method. The simplicity of the interface means that the controller unit's implementation is in complete control of how to perform the scheduling. This allows for a very wide range of simulation strategies to be implemented which should make it possible to implement most real life controller units. The application does not restrict the implementations to using a centralized controlling strategy (as described in 2.2.3 *Types of Home Area Networks*); decentralized systems can just as easily be created by implementing a multi-agent system in the *Run* method that starts up each component as an agent.

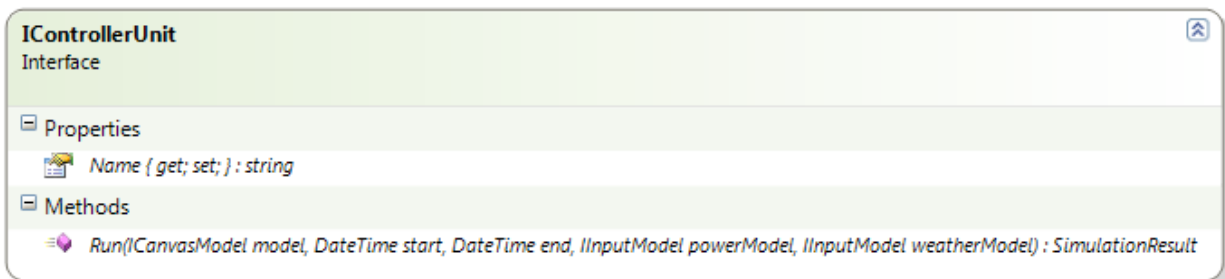


Figure 38: The *IControllerUnit* interface describing what controller units, external or internal, must provide.

Simulation parameters such as choice of controller unit, power- and weather model as well as start- and end dates are chosen in the navigation menu, but are forwarded to and controlled by an implementation of the *ISimulationManager* interface. The forwarding happens through events.

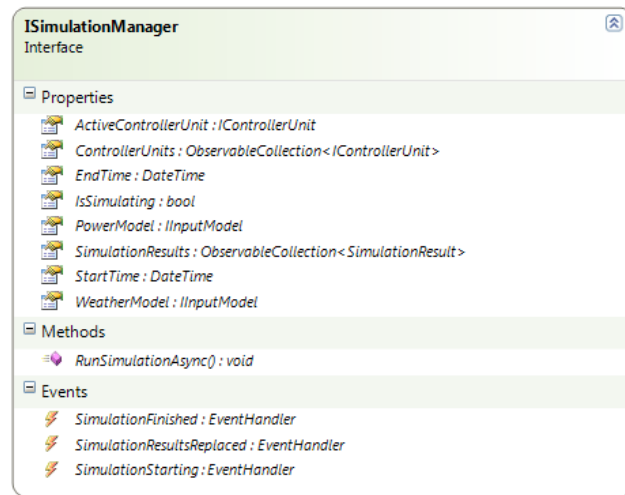


Figure 39: The *ISimulationManager* interface exposing access to simulation relevant properties to be used by other modules interested in the simulations being run.

When a user initiates a simulation action the *ISimulationManager*'s *RunSimulationAsync* method is called. The flow of this method can be seen in Figure 40.

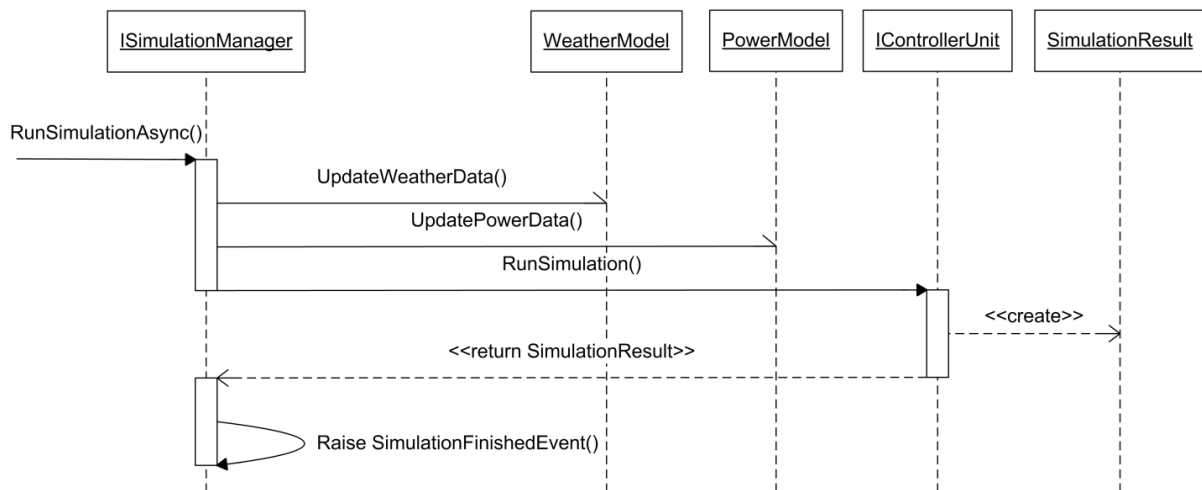


Figure 40: A sequence diagram showing an overview of the calls involved when running a simulation.

When the simulation is started it first updates the power and weather models to be used in order to ensure that they contain the most recent data. Then it invokes the *Run* method in the selected controller unit in an asynchronous fashion in order to avoid blocking the GUI thread. This controller unit then creates a *SimulationResult* which it returns in a callback method to the *ISimulationManager* which then raises the *SimulationFinished* event. This event, along with the *SimulationStarting* event can be used by other modules to react on the simulation, such as showing a progress bar when the simulation is in progress.

The *SimulationResult* returned by the controller unit to the simulation module contains the power and weather models used in the simulation as well as the simulated results from all the scenarios as *ScenarioResults* which contain the results from each connected component as *ComponentResults*. These component results contain the actual data that has been simulated as well as a list of *RunningInfos* that describe when the component is turned on and off. The relationships between the classes can be seen in Figure 41.

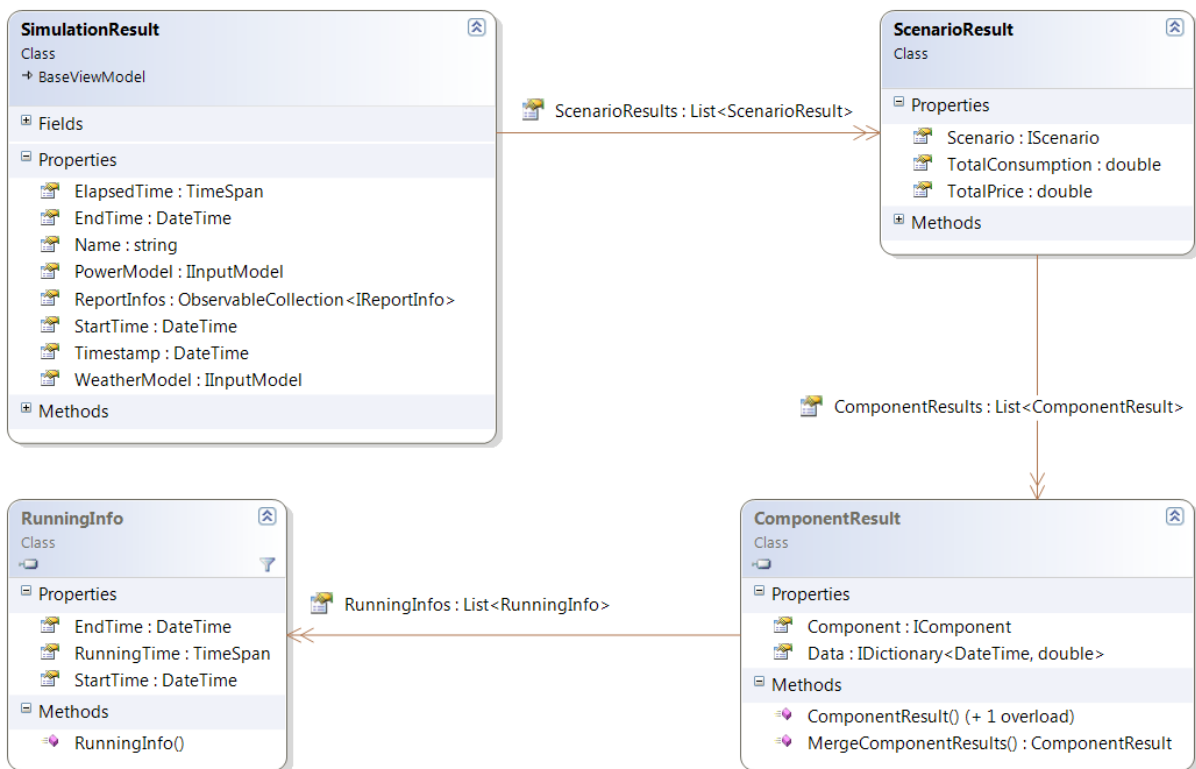


Figure 41: The relationship between the *SimulationResult*, *ScenarioResult*, *ComponentResult* and *RunningInfo* classes.

## 4.2.5. Data Visualization

One of the functional requirements specified in 3.7 *Functional Requirements* is to be able to create graphical reports. This is used in two areas of the application. One is the Summary which shows an estimation of the consumption of elements on the canvas and the other is the Report which enables the user to analyze the data of the simulations. These two areas will be examined in the following sections.

### 4.2.5.1. Reporting

In order to analyze the data produced by the simulation, a module is required that can display the data in meaningful ways. The user should be able to analyze the data from specific components in the home or of the home as a whole and compare the energy consumption with other scenarios in the same simulation. Additionally the user should be able to see a percentage breakdown of the energy consumption of the different categories of components (cooking, entertainment etc.). This way the user can get a good understanding of how his energy consumption is distributed as well as identify the group of components that consume the most energy.

In order to provide this functionality a reporting module has been developed that allows the user to see an overview of each scenario's energy consumption and to create custom charts displaying the data produced by the simulation as well as the power and weather data used by the simulation. All of the data is stored in the simulation result produced by the simulation and this simulation result is the only communication the reporting module has with other modules.

The reporting module consists of a main window that displays three separate user controls: The first control shows the overview of the scenarios, the second allows the user to create custom charts and the

third shows the custom charts. All of these controls share the same view model which makes the communication between them considerably easier and less information has to be passed between view models. The module also contains a second window which allows the user to edit the existing custom charts. This window uses its own view model.

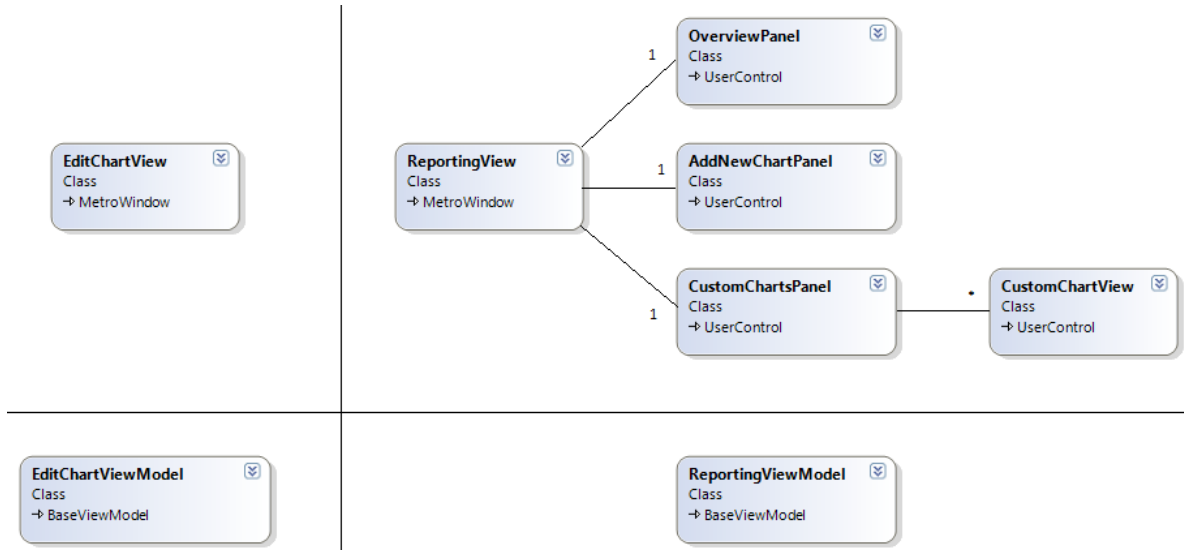


Figure 42: The views and view models of the reporting module.

#### 4.2.5.1.1. Reports

In this application a collection of charts (custom as well as the initial overview) is considered a *report*. It needs to be possible to save the reports, but since the charts can contain a significant amount of data it is impractical to save the data with the charts. Therefore a number of structures have been designed to represent the data being displayed by each chart without storing the actual data. These classes are *ReportInfo*, *ChartInfo* and *SeriesInfo* and can be seen in Figure 43.

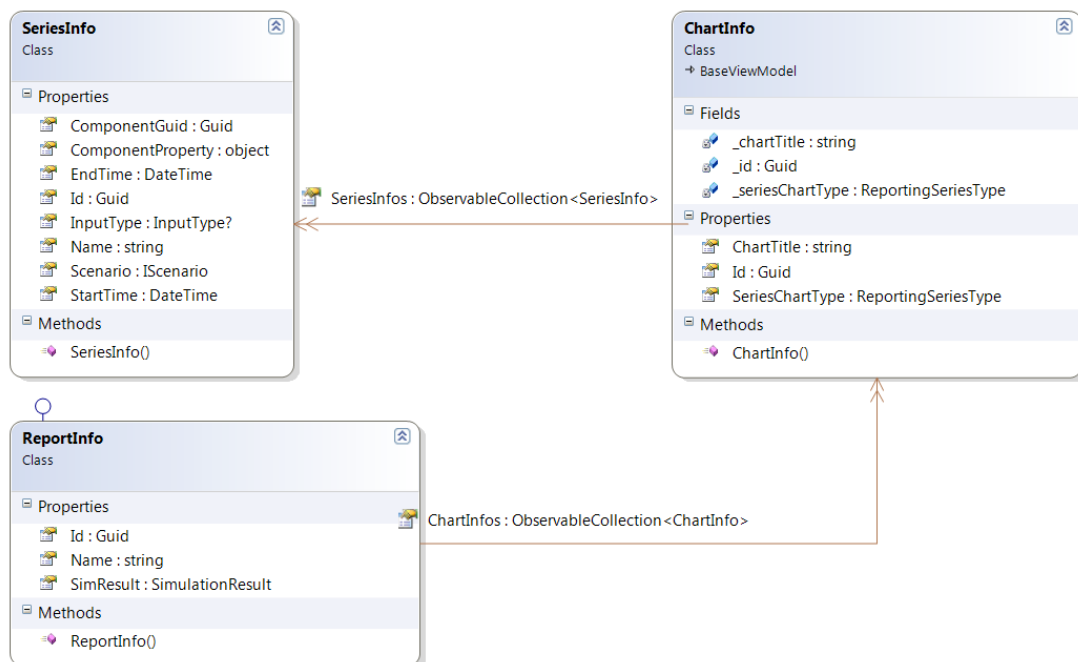


Figure 43: The classes used to represent reports and their custom charts.

*SeriesInfo* represents the data series in a chart and contains information about which component is being displayed, the start and end time of the data etc. The *ChartInfo* contains information about a chart such as a title, a chart type and a collection of *SeriesInfos*. Lastly, *ReportInfo* contains information about the report as a whole, such as a collection of *ChartInfos* and a reference to the simulation result that contains the actual data.

#### 4.2.5.1.2. Charting

The charting framework used to produce the custom charts is the *DataVisualization* namespace from the WPF Toolkit library [86: Microsoft] which is a Microsoft-created downloadable library that adds new tools to WPF. This framework has been chosen because it is free, it's created by Microsoft which gives it certain credibility and it is easily incorporated into a WPF solution. The framework does, however, have two drawbacks. One is that it does not support data binding the series in the charts to collections such that the charts can have a dynamic number of series. This can, however, be fixed by creating a wrapper around the charts that provides that functionality. The second drawback is performance. The charts do not perform well when the data series become very large. There is not much to do about this as it is a consequence of the rendering implementation in the framework as well as the rendering system of WPF [87: Eberhardt]. All the alternative frameworks that are free also have this drawback, however, and as long as this is a prototype there is no point in buying a better framework to improve performance. Figure 44 shows a comparison of the performance of different charting frameworks in .NET and from that chart it can be seen that the Toolkit charting framework used in this application performs very poorly compared to commercial products such as Visiblox Ultimate and SciChart.

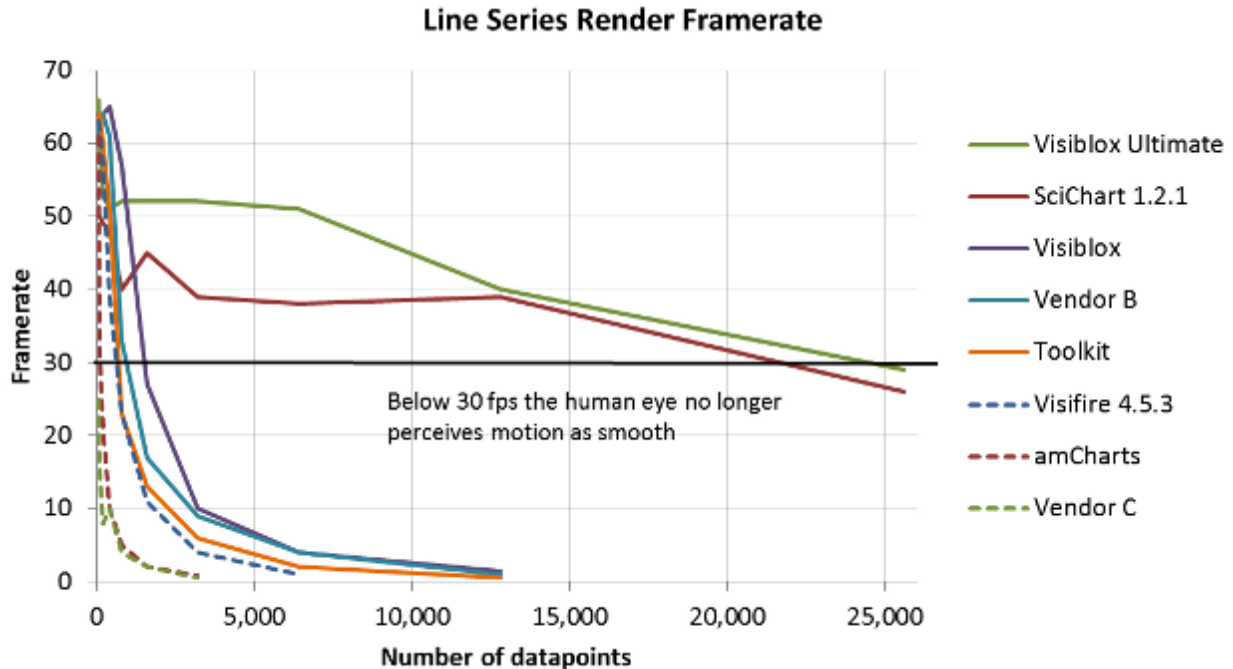


Figure 44: A comparison of some of the available WPF charting frameworks. Source: [87: Eberhardt].

#### 4.2.5.2. Summary

As described in the requirements section (see 3.9.1 *Main Window*) a summary should be displayed next to the canvas showing a brief overview of each of the scenarios in order to support the functional requirement of *creating graphical reports*. This overview is an estimate of how much each category of components will consume in a year. This summary is created as a separate module, *SummaryModule*. This component is an obvious candidate for a module as the summary functionality is completely separated from the rest of the modules. It just needs a notification of what is in the canvas model and when this content changes.

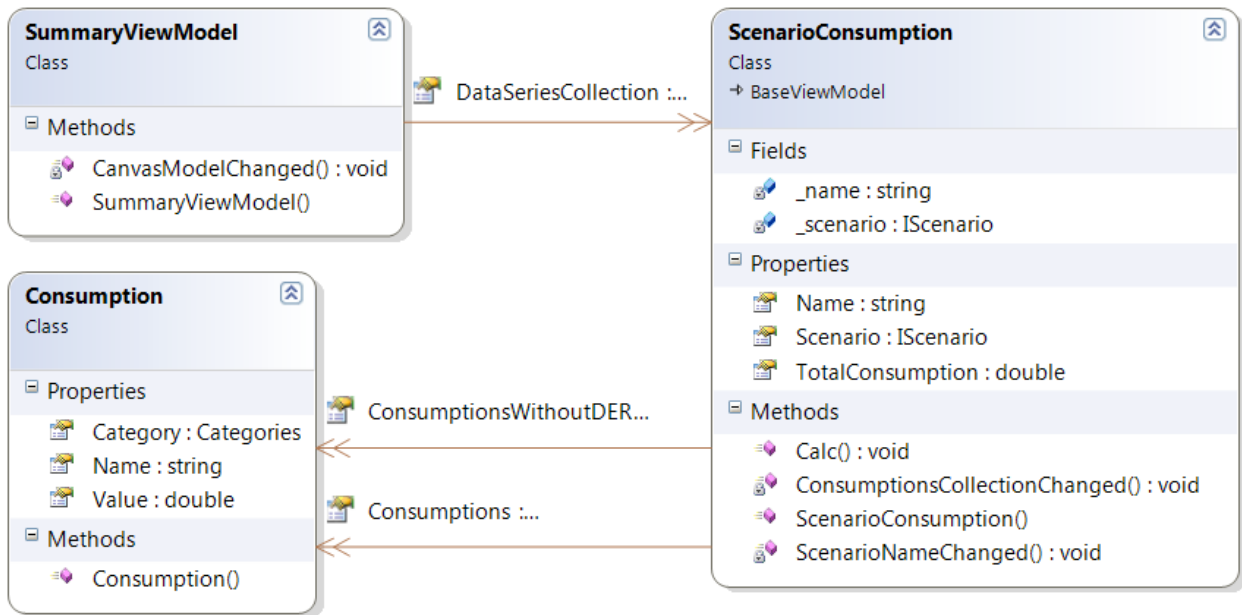


Figure 45: The model and view model classes in the summary module.

The module contains four classes: *SummaryViewModel*, *ScenarioConsumption*, *Consumption* and *SummaryView*. *SummaryViewModel* is the view model of the summary view; *Consumption* is a specification of a group of components estimated consumption in a year and *ScenarioConsumption* specifies the consumption of the entire scenario by containing several *Consumption* instances.

### 4.3. Conclusion

Based on the requirements for the application specified in chapter 3 an application design was made and presented in this chapter. First the general design choices were described in which it was decided that the application should be a desktop application created using C# and Windows Presentation Foundation and it should use the Microsoft Prism library as a foundation. The Prism library was chosen based on integration of several important design patterns, such as the Model-View-ViewModel and command patterns, and it thereby provides a solid base foundation for the application. The Inversion-of-Control pattern is also covered by the Prism library and is chosen in order to fulfil the non-functional requirement of *extensibility* and to provide loose coupling in the application in general.

Based on the functional and non-functional requirements identified in sections 3.6 and 3.7 a number of logical groups of functionality were identified. These groups were then converted to Prism modules (and



separate Visual Studio projects). How these modules met the requirements set in 3.6 and 3.7 was summed up in a table to ensure that the design would meet the requirements for the application (see Table 7). The detailed design of the identified modules was then described and the important interfaces and data structures were introduced. Some important communication internally and between modules has been defined by describing their flows using sequences diagrams. The flows both identified and confirmed the interfaces that each module should expose to the rest of the application.

One of the design challenges was to find a suitable framework that could display the results of a simulation as charts. Several charting frameworks were examined and the Microsoft Charting Toolkit was chosen. Though it does not provide high performance, it is free and therefore fits a prototype application. A comparison of charting frameworks showed that some commercial charting frameworks provide considerably better performance and should be considered if the application is developed beyond the point of a prototype.



# 5. Implementation and Presentation

This chapter will present the end result of the application, one section at a time. Each section of the application will be presented in order to illustrate the functionality available and some interesting details about how the design of the section was implemented will be described.

First an overview of the application will be presented and some general implementation aspects will be described. Then each section of the application will be described, starting with the modeling experience, followed by the summary and input data windows. After this the implementation of the simulations will be described as well as the reporting section. Finally the project functionalities such as save and load are presented.

## 5.1. General

When the user opens the program they are prompted with a startup dialog which is described and shown in 5.7 *Projects*. Depending on whether the user chooses *new-*, *load-* or *latest project* the main window is shown either as empty or filled with content. The main window is shown in Figure 46 and the four main regions of the program are marked: 1) Ribbon/Navigation. 2) Toolbox. 3) Canvas. 4) Summary.

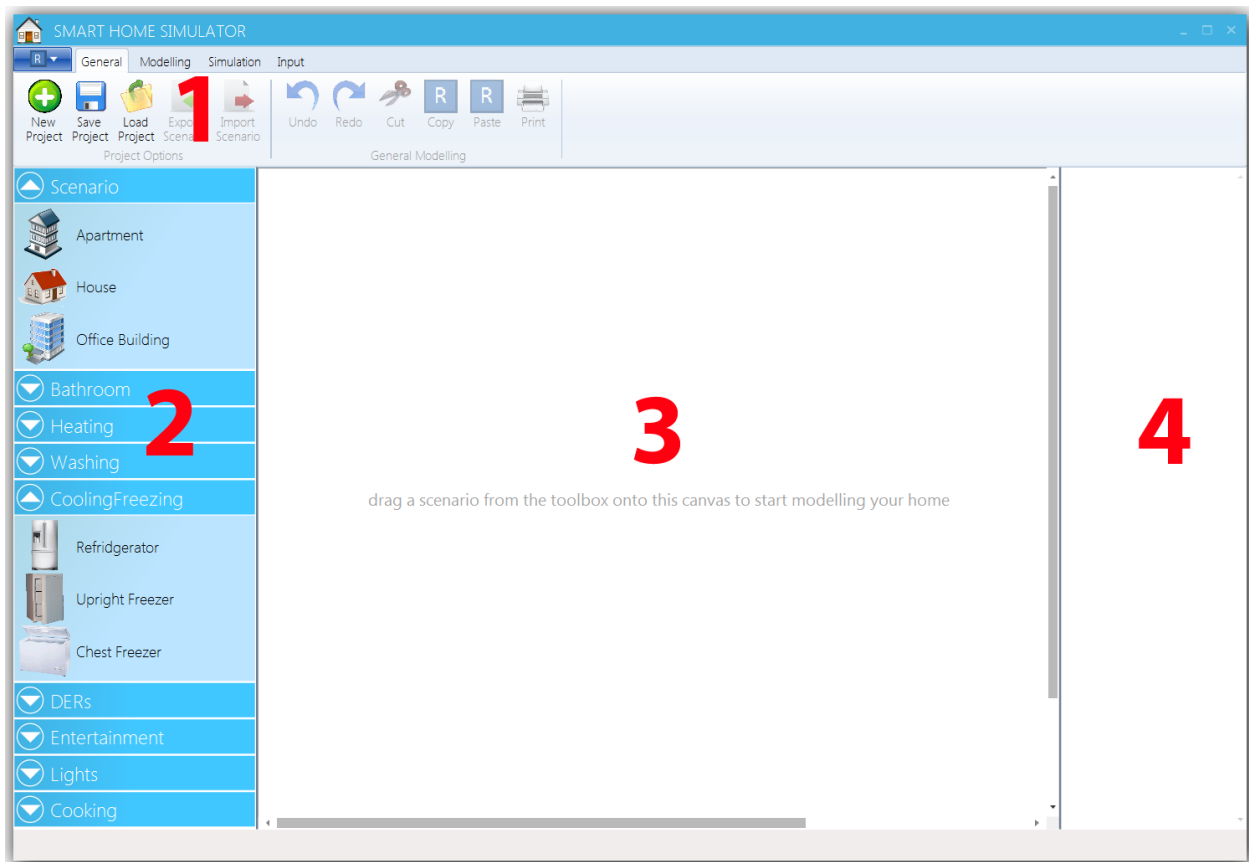


Figure 46: The main window of the application with each of the primary regions marked.

When the application is opened the user can start adding content to the Canvas (3) by dragging elements from the Toolbox (2) onto the Canvas or he can use the ribbon navigation (1) to navigate to other options. Initially the text on the Canvas informs the user that he can drag scenarios onto the canvas to start modeling. If the user chooses to add content to the Canvas (3) the text on the canvas will change to inform the user of the next step, i.e. adding components. Once the user adds components to the scenario the Summary (4) will update. The summary region will contain a sum up of the content currently on the canvas, with information about their estimated yearly consumption and their part of the total consumption of the scenario.

### 5.1.1. General Implementation

This section contains general implementation details about some of the keystones of the application such as the general framework, resource management and implementations of design patterns and workflows used throughout the application.

#### 5.1.1.1. Prism

As described in 4.1.3 *Frameworks and Toolkits* this application uses the Prism library as part of the overall architecture. In order to give the modules access to the IoC-container used to resolve all objects it has been placed in a static class in the *Common* library called *StaticResources* along with a number of other globally available resources and settings. An alternative way of doing it is to specify the container as a parameter to the constructor of all the objects that require access to it. The container will then automatically pass itself to the object's constructor when it creates the given object. This however seems messier as it requires a container parameter in almost all classes which is why the other approach has been chosen.

The *IModule* interface from Prism describes the definition of a module. Each module contains a class that implements the *IModule* interface and these classes are instantiated by the bootstrapper when the application starts up. Each of these module classes also implements a manager interface such as *IModellingManager*, *ISimulationManager* etc. that enables other modules to communicate with the module. When the module classes are created they initialize the rest of their respective module before registering themselves in the container using the specified interface, thereby becoming available to the other modules while remaining loosely coupled.

Lastly, the Prism library comes with a couple of pre-registered classes in the container. This includes the *EventAggregator* which provides mechanisms for inter-modular communication by publishing events without using a predefined interface. The events are implemented as extensions of the *CompositePresentationEvent<T>* class where T is the type of the payload to be transferred with the event. The subscribing class subscribes to the event through the *EventAggregator's Subscribe* method and the publishing class publishes the event through the *EventAggregator's Publish* method. Thereby the two classes have no knowledge of the other class' existence, not even an interface.

#### 5.1.1.2. The Style of the Application

The visual representation of the application is crucial when keeping the user interested in using the application. Therefore this thesis aims to provide a reasonable graphical representation that fits into the overall theme (*Metro*) that Microsoft has set for the next years and as seen in all their new products,

including Windows 8. A third party open-source library has been used which provides styles for the common UI elements in order to make them match the mentioned theme. The library is called MahApps.Metro [88: Aim Software] and is an open source project that is regularly updated to reflect the changes in the design. Some of the styles are used as they are, and some of them are modified to provide a graphical representation for the UI element such that its functionality in the context of this thesis is clearer. The UI elements that are most obviously changed in appearance by the library are the windows that receive a blue header, rounded corners and a shadow; the font which is changed to *Segoe UI Light* as used in Metro; and Combo Boxes that have a more modern look.

### 5.1.1.3. The Icon Provider

The *IconProvider* class is created to provide easier management of the images in the application. By having central access to the images in the application, the program can provide easier access to the images in different places of the program and it can dynamically convert a string key into a *FileSource* by using the name of the file without path and extension. The *IconProvider* is used by the views in the application by using the *IconConverter* class from the *Common* project. The *IconConverter* is a class extending the *IValueConverter* WPF class that enable XAML to convert strings to other values. The *IconConverter* converts the string parameter (file name) defined in the calling XAML view into an image URI by using the *IconProvider* as seen in Code Snippet 1.

```
public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
{
    if (!(parameter is string))
        throw new ArgumentException("The input parameter must be a string", "value");

    return StaticResources.Container.Resolve<IIconProvider>().GetIconUri((string)parameter);
}
```

Code Snippet 1: The Convert method of the *IconConverter* class used by XAML to retrieve an icon URI from a string key.

In Code Snippet 2 an example of a binding using the *IconConverter* can be seen. The example shows how to get an image with the filename "load". A side note to the example is that it only binds to the property called *Pseudo* because a XAML binding requires a property to bind to, before it is syntactically correct. The *Pseudo* property does not have any effect in this case and any existing property of the view model could have been used instead, but might be confusing for the reader of the code.

```
ImageSource="{Binding Pseudo, ConverterParameter=load, Converter={StaticResource IconConverter}}"
```

Code Snippet 2: Example of how the views use the *IconConverter* in XAML.

The *IconProvider* loads all the images from a specified folder into a collection when it is initialized. This allows the user to easily add new images to the program, by just placing them in the correct folder and restarting the application.

### 5.1.1.4. Displaying Views as Windows

In Model-View-ViewModel it is important to separate the views and the view models as much as possible. The declarative language of XAML helps considerably in this regard, and using Prism the *IModule* implementation classes can register the views in specified regions thus ensuring that the view models do not need to know about the views. Sometimes, however, views have to be created from events in the view model such as creating a popup window when a button is clicked or when some action finishes. The only

place this can be handled is in the view model so this class has to initiate the creation of the view. It is undesirable to have the view model know about the view though, especially if the view is displayed from a number of places in the application which would make code maintenance more complex. To solve this issue this thesis has developed a *view creation pattern* that reduces the coupling between the views and view models in such situations. The pattern is shown in Figure 47.

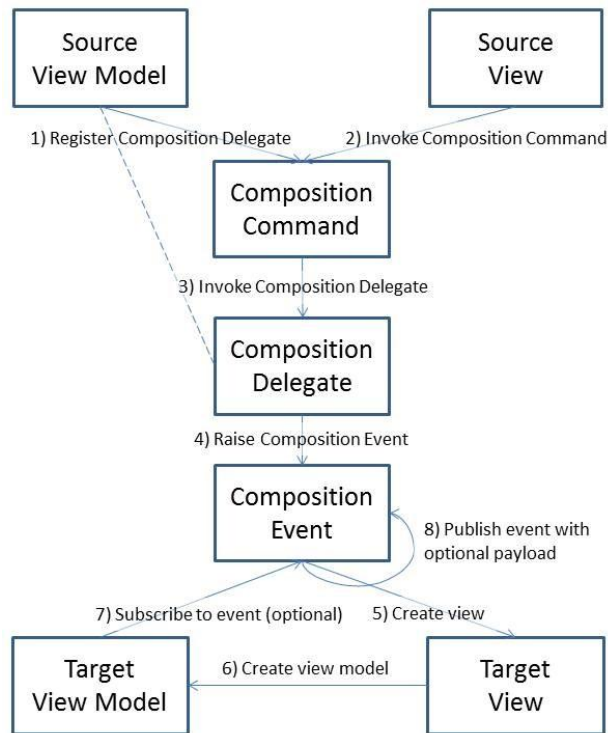


Figure 47: Abstract illustration of the developed view-creation pattern.

In the pattern described in Figure 47 the source view model registers a delegate in a globally available command (1) such that when the source view can invoke the delegate through the command (2). The delegate will raise a globally available event (4) which when raised will create the desired view (5) which in turn creates the view model (6). The target can subscribe to the event if parameters need to be passed to it (7+8).

In the application's implementation of the pattern the source view model registers a *DelegateCommand* (the *composition delegate*) held by the view model with a global *CompositeCommand* (the *composition command*) when initialized (see step 1 in Figure 47). Prism provides a class called *CompositeCommand* which acts like a container for commands into which you can register other commands that will all be invoked once the *CompositeCommand* is invoked. This class works very well with this pattern and has therefore been used in this implementation, but the pattern has been designed such that it does not depend on such a class to be the *composition command*. The *composition command* can just as well be a simple *DelegateCommand*, a *RelayCommand* or any other structure to which you can externally add code to be executed when it is invoked. Similarly the code to be executed (the *composition delegate*) does not have to be wrapped by a *DelegateCommand* as in this implementation, but can be any piece of code such as an *Action* or a *delegate* held by the view model. The reason why the *CompositeCommand/DelegateCommand* combination has been chosen is that it matches the rest of the

application that registers *DelegateCommands* with *CompositeCommands* in many situations and it also provides the view model with a command to access the functionality of the composition delegate without further code that can be used elsewhere. Code Snippet 3 to Code Snippet 5 shows how the composition command and composition delegates are declared and registered in the implementation:

Declaring the composition command globally:

```
public static class GlobalCommands
{
    ...
    public static CompositeCommand PreviousSimulationsCommand = new CompositeCommand();
    ...
}
```

Code Snippet 3: The composition command declared in the GlobalCommands class available to all modules.

Declaring the composition delegate in the source view model (here ReportingViewModel):

```
private DelegateCommand<SimulationResult> ShowReportingWindowCommand { get; set; }
...
ShowReportingWindowCommand = new DelegateCommand<SimulationResult>(ShowReportingWindow);
...
private void ShowReportingWindow(SimulationResult simResult)
{
    //this line shows step 4 of the view creation design pattern: Raising the event with the given payload
    StaticResources.Container.Resolve<IEventAggregator>().GetEvent<ShowReportingViewEvent>().
        Publish(simResult);
}
```

Code Snippet 4: The composition delegate declared in the source view model, here the reporting view model

Register composition delegate in source view model (step 1):

```
GlobalCommands.PreviousSimulationsCommand.RegisterCommand(ShowReportingWindowCommand);
```

Code Snippet 5: The composition delegate registered with the composition command in the source view model.

When the new view should be displayed the *composition command* must be invoked. This can either be done from code in the view model or through data binding in the view and both approaches are used in this application. Figure 47 shows the situation where a button in the view invokes the command.

Invoke composition command in source view (step 2):

```
<ribbon:RibbonButton x:Name="PreviousSimulations"
Command="{x:Static Commands:GlobalCommands.PreviousSimulationsCommand}"
CommandParameter="{Binding SelectedSimulation}" ... />
```

Code Snippet 6: Example of how the reporting view binds a button click to the composition command.

When the command is invoked it executes the code in the *composition delegate*. This code raises the *composition event* which should preferably be a globally available event. Such events are available in Prism by inheriting from the *CompositePresentationEvent<T>* class and can be subscribed to and published through the EventAggregator as seen in Code Snippet 4. If such events are not available an alternative implementation using regular .NET events can be done by implementing a *composition controller* class known by source and target that exposes the event as a regular .NET event as well as functionality to publish the event externally (effectively creating your own scoped EventAggregator).

Before the event is publicly raised it creates an instance of the target view (step 5).

```
public class ShowReportingViewEvent : CompositePresentationEvent<SimulationResult>
{
    public override void Publish(SimulationResult payload)
    {
        var view = new ReportingView();
        view.Show();

        base.Publish(payload);
    }
}
```

Code Snippet 7: The composition event creates the target view before being published.

This view then ensures that a target view model is created using standard declarative coupling (step 6).

```
<Controls:MetroWindow x:Class="ReportingModule.Views.ReportingView" ... >
...
    <Window.DataContext>
        <ViewModels:ReportingViewModel />
    </Window.DataContext>
...
</Controls:MetroWindow>
```

Code Snippet 8: XAML displaying how the target view creates the target view model when it is initialized.

If the target view model should react on the payload of the event it must subscribe to the event when it is created (step 7).

```
public class ReportingViewModel : BaseViewModel
{
    ...
    StaticResources.Container.Resolve<IEventAggregator>().GetEvent<ShowReportingViewEvent>().Subscribe(
        SetSimulationResult);
    ...
}
```

Code Snippet 9: Example of how the target view model subscribes to the composition event (the ShowReportingViewEvent) in order to receive the payload; in this case the simulation result to be displayed by the reporting view.

When the creation of the view and the view model has finished the event is publicly raised and the view model is, if it subscribed to the event, notified of the payload.

### 5.1.1.5. Closing Windows

Windows can be closed very easily in WPF by calling the *Close* method on the window. This is typically done in the code-behind of the view and added as a click event handler to the required button. If, however, you do not want to close the window before having evaluated some logic e.g. whether any changes have been made and if so asking the user if he wants to save before closing, there will be complications. The view should not (and cannot) contain such logic as the properties to check are almost certainly only available in the view model which means that the code-behind class cannot read these values without explicitly casting the data context of the window to the class of the view model which should be considered bad practice as it creates a strict connection between the concrete view and view model implementations. The view model, however, should not know about the view so it cannot call the *Close* operation on the window in order to close it once the logic is fulfilled. In order to solve this issue the flow illustrated in Figure 48 is used in this thesis.



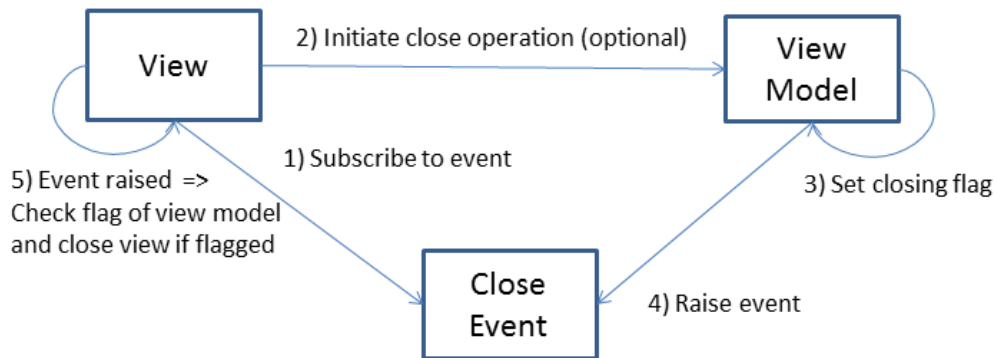


Figure 48: An abstract illustration of the flow being used to close views throughout the application.

When a view should be closed the view model sets a flag saying that the window connected to this view model should be closed and then raises an event (3). The view is subscribed to the event (1) and closes when the event is raised (5). The initiation of this operation will typically be from the view through a close command in the view model (2), but can also be initiated by the view model. The event is a global event that has no connection to the view model or the view which is required for proper loose coupling. Therefore all views that are closable are connected to the same event which is why it is necessary to have the view model set a flag so that the view knows that it should close. Reading the flag can be done using data binding to ensure loose coupling.

In order to ensure maximum reusability of this pattern, it has been implemented as three classes providing the functionality for the three elements of the pattern that can all be easily incorporated into existing views. The *CloseViewEvent* is the global event to which all closable views subscribe. It is a *CompositePresentationEvent* so it can be published by the EventAggregator of Prism.

View models of closable views need a flag they can set when a close operation is pending. Additionally it is useful with a command that can initiate the close operation. To make this as dynamic and extensible as possible an abstract *ClosableWindowViewModel* has been created and all view models of closable views then simply extend this class. It provides a Boolean property called *PendingClose* and a *DelegateCommand* called *CloseCommand* that, when called, sets the *PendingClose* flag to true and raises the *CloseViewEvent*. The delegate invoked by the command is called *Close* and is marked as *virtual*. This means that if the developer overrides the method in a subclass the subclass' implementation will be used rather than the abstract class' implementation regardless of how the method is called. The logic to be performed before the window is closed is added to the overridden implementation of the *Close* method followed by a call to *base.Close()* which will then set the flag and raise the event as seen in Code Snippet 10.

```

protected override void Close()
{
    if (MessageBox.Show("Are you sure you want to close the view?", "Close", MessageBoxButton.YesNo) ==
        MessageBoxResult.No)
    {
        return;
    }
    base.Close();
}
  
```

Code Snippet 10: An example a view model overriding the *Close()* method of the *ClosableWindowViewModel* to provide a logic check before closing the view. In this case the view will not close if the user chooses "No" in the displayed message box.

In order to provide the views with a generic way of subscribing to the *CloseViewEvent* without duplicating code, a *WPF behavior* has been created called *CloseViewBehavior*. Behaviors are pieces of code that can be attached to a class in XAML and thereby injecting reusable functionality into the view. The behavior simply subscribes to the global *CloseViewEvent* when it is attached to a view. When the event is raised the subscribed event handler checks the *PendingClose* property of the data context (i.e. the view model) and if it is set it closes the view. The use of the behavior in XAML can be seen in Code Snippet 11 and the functionality of the behavior can be seen in Code Snippet 12.

```
<Interaction:Interaction.Behaviors>
  <Behaviors:CloseViewBehavior />
</Interaction:Interaction.Behaviors>
```

Code Snippet 11: An example of how to attach the *CloseViewBehavior* to a view using XAML.

```
public class CloseViewBehavior : Behavior<Window>
{
    protected override void OnAttached()
    {
        base.OnAttached();

        if (!(AssociatedObject.DataContext is ClosableWindowViewModel))
            throw new ArgumentException("The Associated Object's DataContext must be a
                ClosableWindowViewModel");

        StaticResources.Container.Resolve<IEventAggregator>().GetEvent<CloseViewEvent>().Subscribe(o =>
        {
            var closableWindowViewModel = AssociatedObject.DataContext as ClosableWindowViewModel;
            var pendingClose = closableWindowViewModel != null && closableWindowViewModel.PendingClose;
            if (pendingClose)
                AssociatedObject.Close();
        });
    }
}
```

Code Snippet 12: The *CloseViewBehavior* implementation.

#### 5.1.1.6. Data Representation

The application works with a lot of data that is distributed over time. This includes energy prices, weather data, the energy consumption over time for a component etc. If for example the energy prices come in a different time granularity than the data provided by the simulation it is not easily comparable. Therefore in order to make this data comparable and easier to integrate a common format is required for such data. In this thesis all time distributed data will be represented as collections of points in time with a fixed time between each point. Each point in time must provide a *DateTime* that represents the time and a double which represents the value of the point. The structure of the collection does not matter. Two possible representations are to use a dictionary with *DateTimes* as keys (such as energy consumptions for components) or using a list of objects that contain *DateTime* and double properties (such as power and weather data).

The time between each point is stored in the *InputDataIntervalInMins* property in the *StaticResources* class in the *Common* project such that all modules can access it and use it when doing calculations or creating lists containing data over time. In the current implementation this value has been set to 15 minutes. The 15 minutes interval has been chosen as a compromise between the data available from power and weather which is one hour intervals and the detail desired in a simulation which is ideally around 1-5 minutes, as

well as performance which goes down the more data points you use. Currently the input data is distributed over the 15 minutes interval such that all quarters in a given hour have the same value. If more detailed input data is found at a later time they can be added to the application to use the 15 minutes intervals optimally.

## 5.2. Modelling

Functional requirement 1 in section 3.7 specifies that the user should be able to model specific scenarios and the design for such functionality was described in 4.2.2 *Modelling*. This section presents how the user can model scenarios and describes the implementation of the related functionality.

### 5.2.1. Toolbox and Components

On the left side of the program's main window is a toolbox containing all the elements that can be dragged onto the canvas in the center of the window. The toolbox is divided into logical categories where each category contains the elements related to it. Clicking the expander arrow of a category unfolds it and clicking it again collapses the category. Figure 49 shows a screenshot of how the toolbox looks with the scenario and washing categories folded out.

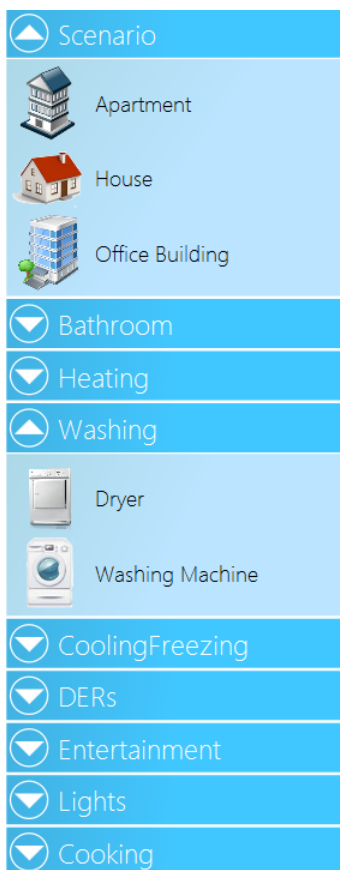


Figure 49: The toolbox with the Scenario and Washing categories folded out.

It is possible for the user to create new, delete or edit existing toolbox elements. This allows them to create exactly the elements that they use within their home and reuse them multiple times and across projects. When the user wants to edit or delete elements they can do this from two places; from the modeling tab in the Ribbon and from a context menu that appear when right-clicking on an element. Creating new elements can only be done from the modeling tab in the Ribbon. As specified in ISO standard 9241-171 in 3.6.5 *Usability* people have different preferences as to how they want to access functionality and this preference might change over time while using the application. Therefore the same functionality is exposed in multiple places to allow the user to access it in the way he wants (as mentioned in 4.2 *Detailed Design*) and thereby increasing the usability of the application, supporting the non-functional requirement specified in 3.6.5 *Usability*.

When choosing to delete or edit an element through the Ribbon the user will be prompted with a simple window where he can choose the element to edit or delete. When the user chooses to delete the element from either the window or from the context menu, the element is deleted and will be removed from the toolbox. When choosing to edit an element the user is prompted with a new window which contains all the properties of the selected element which can be edited. Before saving the changes the input values are verified in order to ensure that they are of the correct format. This could be to check for the correct separators in doubles, that numbers in general do not contain letter and so on. In Figure 50 a screenshot of the edit window can be seen, containing fields and values related to a Light Bulb.

Wattage	40
Possible On Period Weekday Start	07:30:00
Possible On Period Weekday End	22:30:00
Possible On Period Weekend Start	09:00:00
Possible On Period Weekend End	23:30:00
Actual On Time Weekday	05:00:00
Actual On Time Weekend	05:00:00
On Frequency	1
Name	Light Bulb
Icon	Light Bulb
Category	Lights
Communication	

Figure 50: The Edit Toolbox Element window that allows the user to edit properties of an element in the toolbox.

The window for creating a new element is very similar to the edit window. The main difference is that when creating a new element you need to choose which type of element you want to create before the related fields will occur. An example of the creation window can be seen Figure 51, where a *SolarPanel* type has been selected in the combo box in the top as the element the user wants to create. When new elements are added to the toolbox it is automatically refreshed to include the new element.

SolarPanelModel	
Size	
Efficiency	
Name	
Icon	
Category	
Communication	

Figure 51: The view used to create new toolbox elements.

### 5.2.1.1. Implementation of the Toolbox and Components

#### 5.2.1.1.1. Retrieving the Elements

The toolbox retrieves all the available elements through the modeling manager which it accesses through the *Prism* container as described in 4.2.2.4 *Toolbox*.

The current implementation of the *IDataProvider*, used by the modeling manager to retrieve modeling elements, stores all the elements in a folder as XML files. The data provider retrieves all the elements by going through each of the XML files in the folder and checking their validity against the set XSD schemas. An XSD schema has been developed for each valid type. The validation of the XML files is done to ensure that the modeling element contains valid data for all properties before it is added to the application. If it fails validation it will be skipped. The data provider uses the *System.Xml* package to load and create the XML documents. When the data provider converts the XML document to a specific Model class (see 4.2.2.1 *Models*), it uses the Prism container to resolve the type of model. The modeling manager registers all the model types at initialization, so they are available through the container.

When the data provider tries to set all the properties on the model to the values from the XML document, it uses *Reflection* to do so. If it finds a property in the model through reflection with the same name as an XML tag in the XML file it sets the value of the property to that of the XML tag. Before setting the values, however, the values are converted from the XML value types to the corresponding *C#* data type using the types own convert method or the general *Convert* class.

#### 5.2.1.1.2. Reflection of Elements

Most model types have different properties so the windows used to create or edit elements in the toolbox must be dynamic in order to accommodate this. In order to do this the application uses *Reflection* to discover the properties available in the model of the selected element. Each property is assigned a UI *Control* such as *TextBox*, *ComboBox*, *CheckBox* or similar based on the property's type. The created control is then placed in a collection along with the property name which is displayed in the GUI.

Upon saving the changes the user has made to an element, the program creates a *Dictionary* containing the each property name and the values of the related UI Controls. The window then resolves the *IModellingManager* and calls the *SaveToolboxItem* method with the toolbox element's *Typeid*<sup>1</sup> or the *AddNewToolboxItem* method with the *Type* of the element along with the *Dictionary* containing the values of the properties. The modelling manager forwards the request to the data provider (currently the XML parser) which uses the *XMLWriter* and *XmlDocument* classes from the *System.XML* namespace to create or edit the XML files.

## 5.2.2. Canvas

The canvas is a crucial part of the modeling in the program. It is responsible for containing and presenting the result of the users modeling actions. Graphically the canvas is simply a white background, but the elements which are shown upon it are represented by an icon and buttons used for control and information about the element. The icon in the middle is based on the *Icon* property in the Model of the element, and is displayed using the *IconConverter* (see 5.1.1.3 *The Icon Provider*). In Figure 52 an example of a Component

---

<sup>1</sup> All components have a *Typeid* which is a unique Id representing which model type the component is an instance of.

(Washing machine) can be seen. The estimated yearly consumption of the component is shown in the top right corner. The color indicates how high the consumption is; the darker red, the higher consumption. The information icon and the two buttons are only showed when hovering the mouse over the component. The information icon in the bottom right corner shows the name of the icon upon mouse-over. The button in the bottom left corner deletes the component from the canvas and the button on the top left opens up the edit component view.



Figure 52: An example of how a component looks on the canvas.

The components on a canvas are automatically placed in a circle around the scenario. As described in 3.9 *Graphical User Interface Mock-ups*, and seen in Figure 53 the components on the canvas are connected to a scenario by a line that goes from the center of the scenario to the center of the component and the color of the line shows the category of the component.

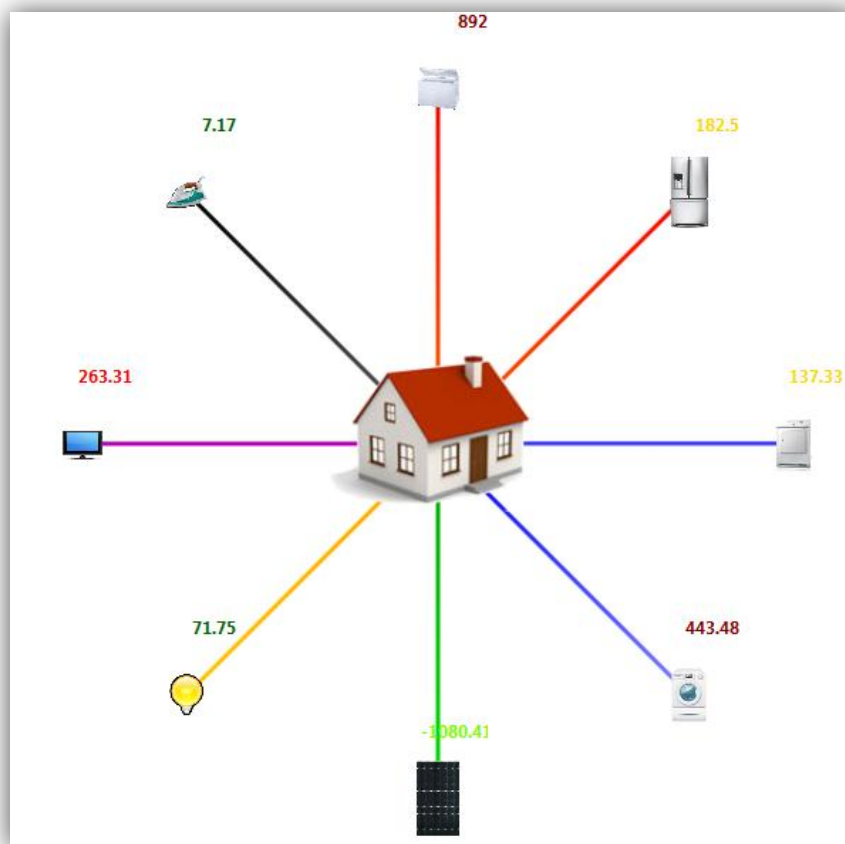


Figure 53: An example of a scenario with eight components as displayed on the canvas.

The component and scenario need to be graphically on top of the connecting line. The canvas class provides a *SetZIndex* method which should be able to place the element in a specific Z-level, making them appear on top of other elements, but when dynamically adding elements onto the canvas, this Z-Index is ignored and the line is then placed on top. This is because the canvas' content is bound to a list within the canvas model and the elements are added to the list as they get added to canvas. This means that the last added item receives the highest Z-level, making it appear on top. As the connection is added along with the component the connections will always be added after the scenario and therefore be displayed on top.

To solve this the program uses a custom sorting method on the list in the view model, which puts the connection elements in the first positions in the list (Lowest Z-level) followed by the components and scenarios (Higher Z-level). This means that each time the content on the canvas changes the element list needs to be refreshed and sorted in order to display the elements correctly. Sorting a list every time an element is added to the canvas is a potential performance issue and therefore the performance of this approach has been tested thoroughly in *6.1.1 Sorting Performance*.

To provide the user with a better experience with the canvas the program allows the user to resize the canvas as needed. Whenever an element is placed at the edge of the canvas, the canvas is extended to provide extra room around the element. To provide this functionality the canvas view uses a new class, *CanvasAutoResize*, that extends from the original Canvas class, but where the *MeasureOverride* method is overridden. The code in this method examines all the content on the canvas and finds the highest Top and Left values. These properties define the positions of the elements on the canvas. If these values indicate that a component is too close to the edge of the canvas the size of the canvas is increased by an appropriate amount.

#### **5.2.2.1. Drag and drop**

As described in *5.2.1 Toolbox and Components* elements are added to the canvas by dragging them from the toolbox onto the canvas. Windows Presentation Foundation contains some drag-drop functionality, but in order to use this properly in a MVVM setup some extra code must be developed. Inspired by an implementation from Marcin Kaluza [89: Kaluza] drag-drop functionality has been created. It consists of a *DragSourceBehavior*, a *DropTargetBehavior*, a *DragDropDataObject*, an *IDragSource* interface and an *IDropTarget* interface. *DragSourceBehavior* is a class extending the WPF *Behavior* class and it provides a UI Element with the ability to be dragged. The data context (i.e. the view model) of the UIElement should be a class implementing the *IDragSource* interface which describes how the element behaves when being dragged as well as providing a method of retrieving the data object that is being dragged as an implementation of the *DragDropDataObject* class.

The *DropTargetBehavior* also extends *Behavior* and it provides the functionality required for a UIElement to react on elements being dragged over or dropped onto it. View models of such UIElements are required to implement *IDropTarget* which describes what should happen when the item is dropped on the UIElement. The classes can be seen in Figure 54.

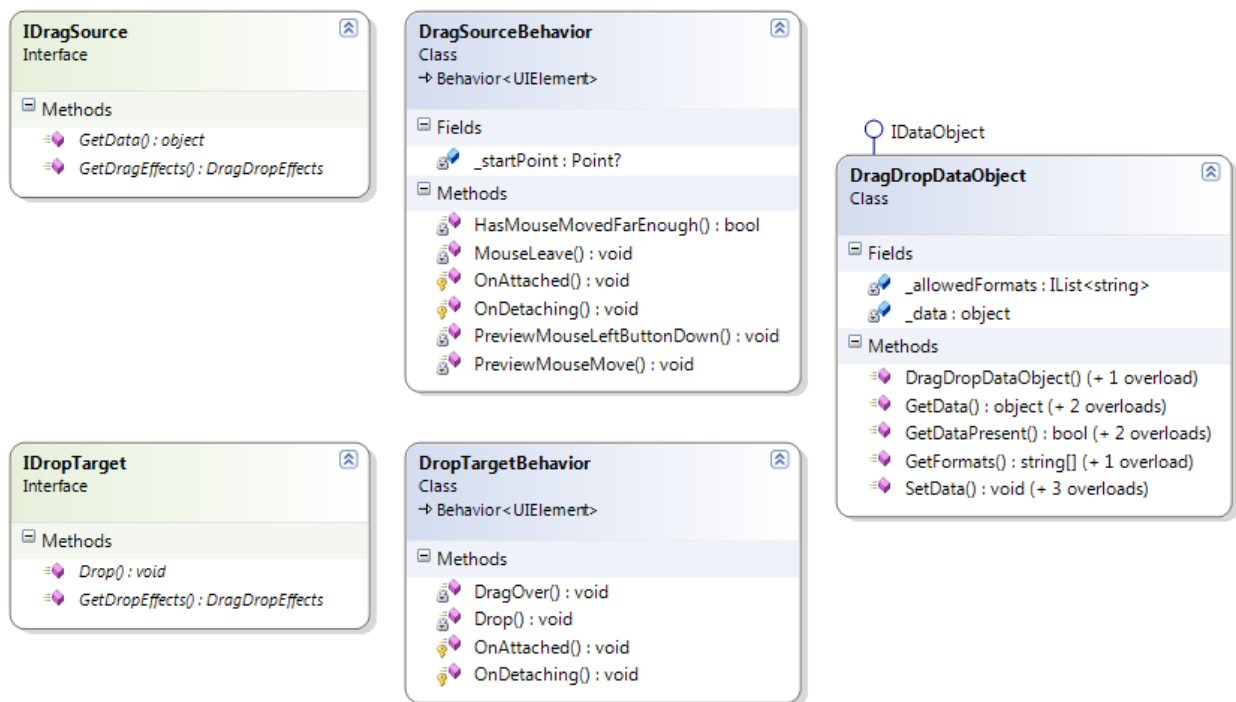


Figure 54: The classes making up the Drag-and-drop functionality.

The *DragSourceBehavior* is attached to all the *UIElements* that should support dragging, which in the modeling section of the application are the *ToolboxObjectView* (to enable dragging from the toolbox) and the *ScenarioView* (to enable moving scenarios on the canvas). When the user starts dragging the element the behavior asks the dragged object's view model, the *IDragSource* implementation, for the element's *drag effects* as well as its *drag-data*. The drag effects is a WPF enumerable called *DragDropEffects* that describe whether the element can move or not. The drag data is the data that should be carried as payload and eventually dropped to some location, wrapped in a *DragDropDataObject* instance. The behavior then calls the static method *DoDragDrop* in the *DragDrop* class from the *System.Windows* namespace which initiates the underlying WPF drag-drop event sequences. When the user releases the mouse on a *UIElement* that uses the *DropTargetBehavior* the *Drop* method of the behavior will be called. This method retrieves the mouse position in which the element was dropped and calls the *Drop* method on the underlying view model, i.e. the *IDropTarget* implementation. The *IDropTarget* then retrieves the data object from within the *DragDropDataObject* and uses this object in the logic. In the modeling implementation the data object retrieved from the *DragDropDataObject* is the *DraggableCanvasObjectViewModel* that is to be placed on the canvas (the *DropTargetBehavior* and *IDropTarget*). The process is described in Figure 55. For further details about how the element is added to the canvas once dropped see Figure 33 in 4.2.2.5 *Canvas*.



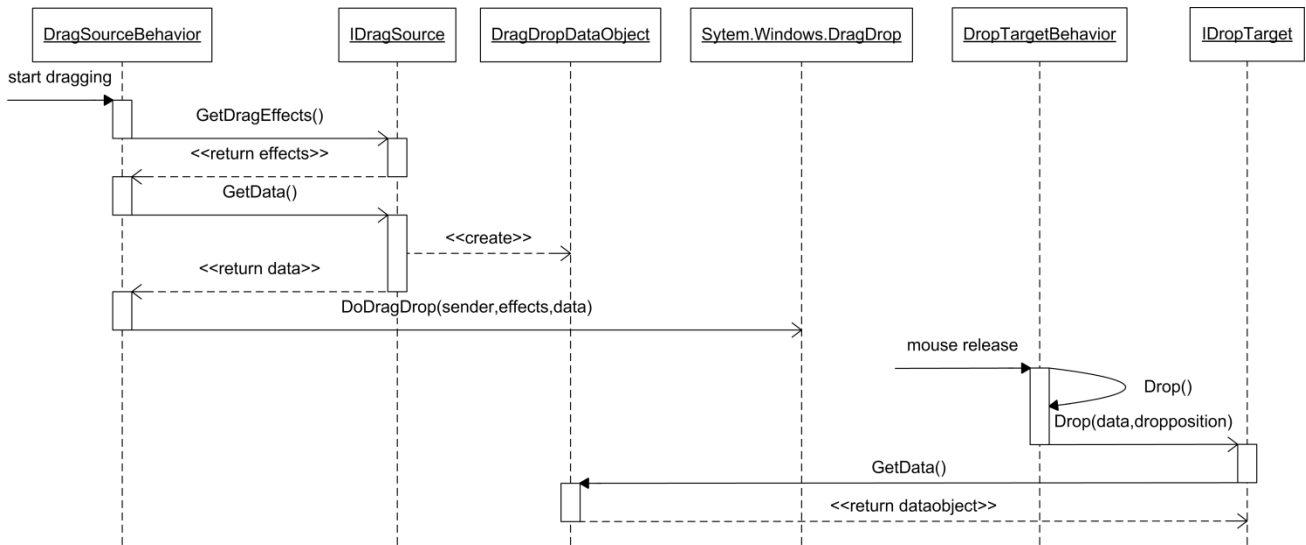


Figure 55: A sequence diagram displaying how the drag-drop classes work together with the WPF features to create drag and drop functionality.

### 5.2.2.2. Edit Settings on Canvas Elements

In order to edit the properties on one of the elements on the canvas, the user can press on the edit button (A pencil icon) at the top left corner of the icon of the element or by right-clicking on the element. It is also possible to edit a component on the canvas through the ribbon navigation, if a component or scenario is selected on the canvas. When doing so the user gets prompted with a new window which contains a number of controls containing the values of the properties of the selected element. The window is visually exactly the same as the edit toolbox element window which can be seen in Figure 50. The main difference is that when saving the changes this window only changes the values within the model class of the selected element and does not change the values in the XML files. Therefore only the specific instance of the element on the canvas is changed. Reflection is used in the same way as in the *edit-* and *create toolbox element* windows to edit the model.

### 5.2.3. Undo/Redo

As described in the Requirements chapter (see 3.7 *Functional Requirements*), undo and redo functionalities are important in tools like this. The functionality has been implemented using the *Command pattern* and the implementation is placed in the *Common* project such that all modules have access to the functionality. The implemented classes can be seen in Figure 56.

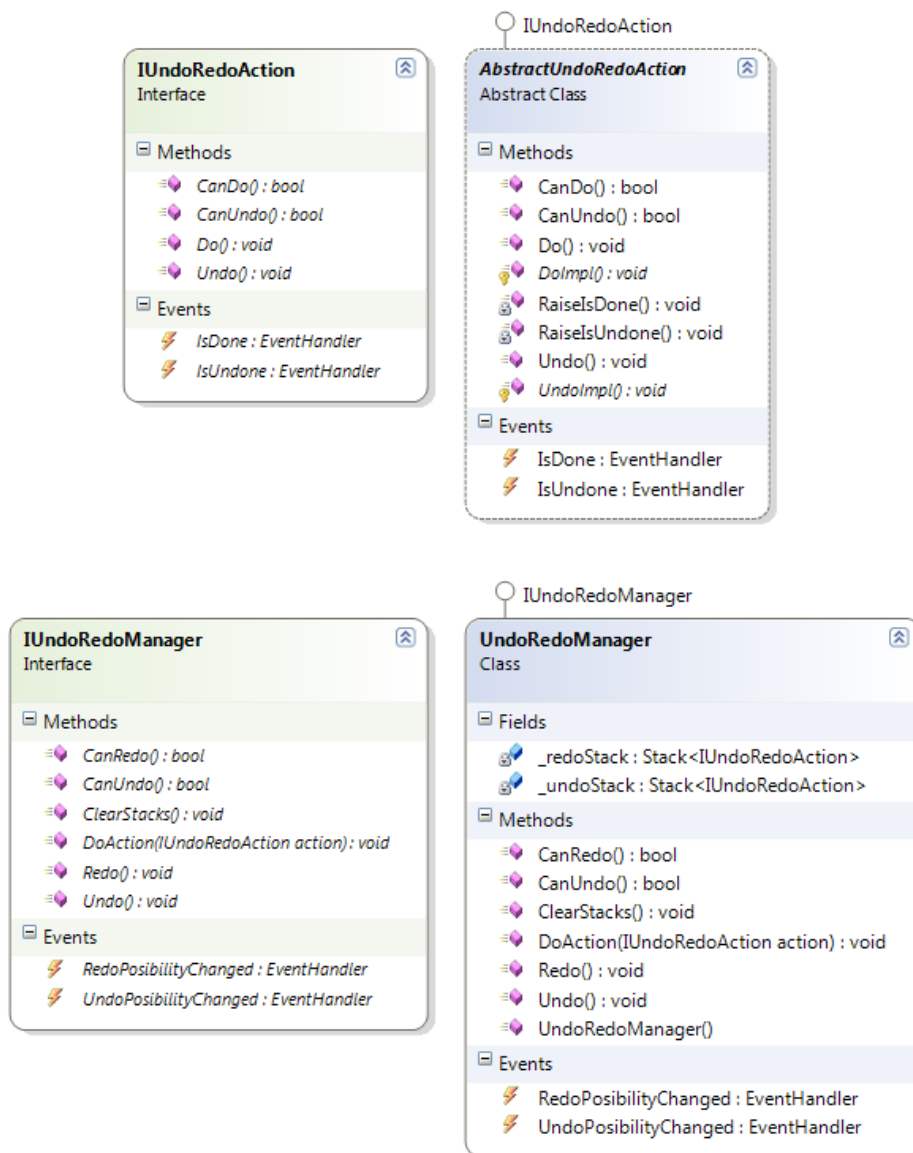


Figure 56: The classes used to provide undo-redo functionality to the application.

An undoable action must implement the `IUndoRedoAction` interface. This interface contains a `Do` method and an `Undo` method that performs and rolls back the action, respectively. It also contains events that are raised when the action is done and undone. In order to make the implementation of such actions as easy as possible an `AbstractUndoRedoAction` class has been implemented that provides the structure of raising the events correctly when `Do` and `Undo` are called as well as implement default implementations of `CanDo` and `CanUndo` that simply return `true`. The class implements `Do` and `Undo` as shown in Code Snippet 13:

```
public void Do()
{
    DoImpl();
    RaiseIsDone();
}

protected abstract void DoImpl();
```

Code Snippet 13: The `AbstractUndoRedoAction` class' `Do` method implementation that ensures that events are always raised.

This way the classes extending the *AbstractUndoRedoAction* class only need to implement the *DoImpl* and *UndoImpl* methods and the rest of the functionality will be handled by the base class.

In order to enable the actions to be done and undone in a correct order an *UndoRedoManager* has been developed. It contains three key functionalities: Do an action, undo an action and redo an action. The manager contains two stacks; one for actions that can be undone and one for actions that can be redone. Whenever an action is performed for the first time it is placed on the undo stack and the redo stack is cleared. This ensures that undesired situations where redoing an action collides with the newly performed action are avoided. Similarly to the action, the manager raises events when the undo and redo stacks change such that a GUI can display the status of the stacks correctly.

When a request to undo the latest action is sent to the manager it first checks whether this is possible by checking the size of the undo stack and calling the *CanUndo* method on the action on the top of the stack. Then it calls the *Undo* method on the action which rolls back the effects of the action, removes it from the undo stack and places it on the redo stack. Redo is performed in a similar way.

It is possible to use multiple undo-redo managers in the application that each provide undo-redo functionality in different areas. This can be done by registering an *UndoRedoManager* instance in the container with a unique key. In the current implementation this is done only for the canvas which registers an instance of the manager using “*modelling*” as key. This manager is then used in all undo-redo scenarios related to modelling of scenarios.

### 5.3. The Summary Area

As described in the Design chapter (see 4.2.5.2 *Summary*) a graphical summary of the estimated consumption of the scenarios on the canvas is shown in a view to the right of the canvas (see Figure 46). The summary contains a section for each scenario on the canvas in which the consumptions of the components connected to the scenario is summed up and grouped by their category. The estimated consumptions are also displayed as a pie chart along with an estimated total yearly consumption for the scenario. Each category has its own color which is defined in the *CategoryColorConverter* class and it is used to color the categories in the text as well as the pie chart and even the lines on the canvas connecting the components to the scenario. This makes it a lot easier for the user to identify which components are members of which category such that the reasons for a high consumption in a particular category can be identified quickly. An example of the summary view with two scenarios can be seen in Figure 57.

### 5.3.1. Implementation of the Summary Module

The summary is implemented in the Summary Module. It consists of a view and a view model as well as the two model classes *ScenarioConsumption* and *Consumption*. The module also consists of an implementation of *IModule* that registers the view in the Shell's Summary-region and thereby adding it to the application.

The summary view model subscribes to the global EventAggregator's event *CanvasModelChangedEvent* which is raised whenever the canvas model changes. This happens when a new project is loaded or if an element is added to or deleted from the canvas or if the settings of an element on the canvas are changed. When this event is raised the view model resolves the current *CanvasModel* from the container and runs through all the scenarios in it, creating a *ScenarioConsumption* instance for each of them. Each *ScenarioConsumption* creates a *Consumption* instance for each category of components connected to it and calculates the estimated consumption by retrieving the *YearlyConsumption* property from each component. This property is defined as a read-only property in the *IComponent* interface and each implementation of the interface describes how to estimate the consumption in the property's get-method. In all the current implementations the method retrieves the consumption using the same methods as the simulation process, but without any scheduling. This way the estimations are as accurate as possible while still keeping a high performance. To further improve the performance the property simply retrieves the consumption for all days in January and scales this value up to match a full year without considering prices or weather data.

The pie chart is created using the *DataVisualization* charting framework from the WPF Toolkit as described in the Design chapter (see 4.2.5 *Data Visualization*). In the summary only basic functionality is required and therefore no modifications are necessary to the framework.

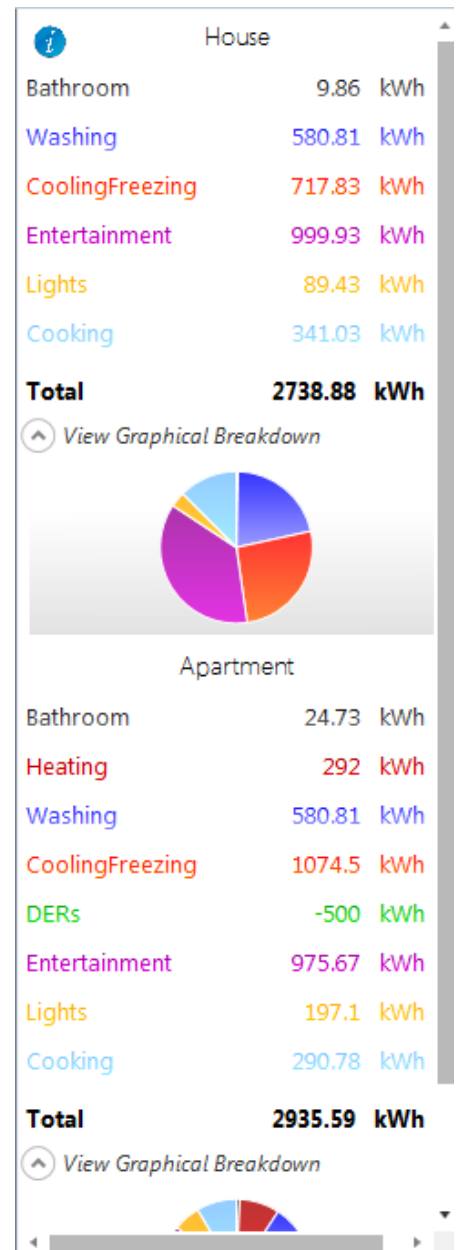
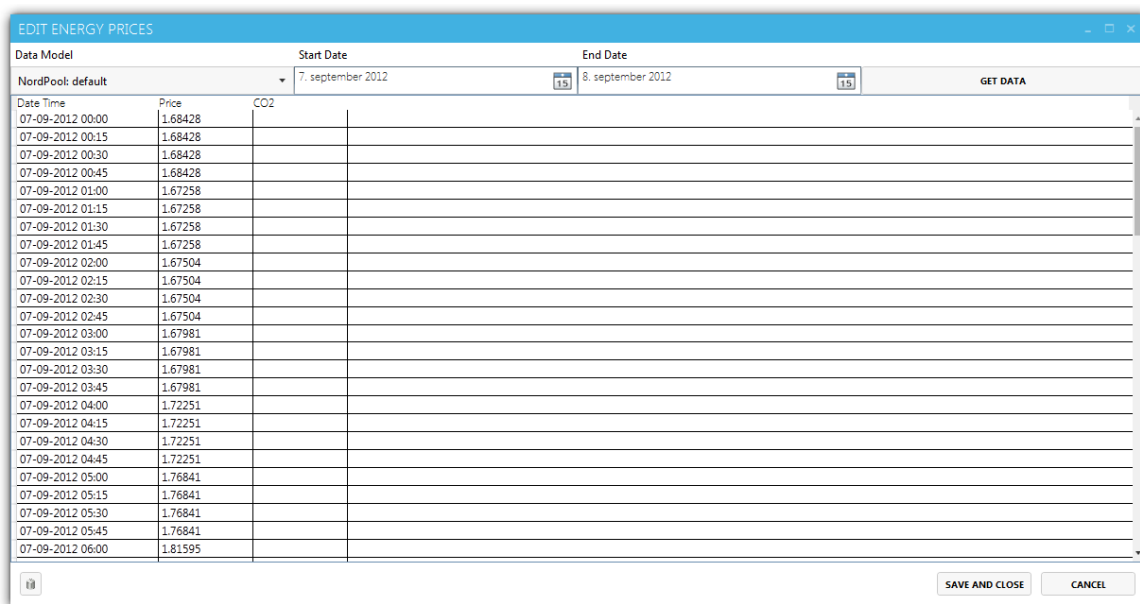


Figure 57 The summary displaying breakdown of the consumption from two scenarios.

## 5.4. Input Data

The functional requirements (see section 3.7) specify that the application must integrate real power and weather data. Section 4.2.3 *Data Handling* described the design of such functionality and in this section the implementation will be presented.

There are currently two types of input windows in the solution, one for weather data and one for power data. It is two very similar windows in which it is possible to retrieve data from a selected data provider. The window can be accessed through the buttons from the input tab in the ribbon navigation. A screenshot of the “Edit Power Prices” window can be seen in Figure 58.



Date Time	Price	CO2
07-09-2012 00:00	1.68428	
07-09-2012 00:15	1.68428	
07-09-2012 00:30	1.68428	
07-09-2012 00:45	1.68428	
07-09-2012 01:00	1.67258	
07-09-2012 01:15	1.67258	
07-09-2012 01:30	1.67258	
07-09-2012 01:45	1.67258	
07-09-2012 02:00	1.67504	
07-09-2012 02:15	1.67504	
07-09-2012 02:30	1.67504	
07-09-2012 02:45	1.67504	
07-09-2012 03:00	1.67981	
07-09-2012 03:15	1.67981	
07-09-2012 03:30	1.67981	
07-09-2012 03:45	1.67981	
07-09-2012 04:00	1.72251	
07-09-2012 04:15	1.72251	
07-09-2012 04:30	1.72251	
07-09-2012 04:45	1.72251	
07-09-2012 05:00	1.76841	
07-09-2012 05:15	1.76841	
07-09-2012 05:30	1.76841	
07-09-2012 05:45	1.76841	
07-09-2012 06:00	1.81595	

Figure 58: The window in which the user can edit the energy prices and create specific power models to be used by the simulation.

There is a ComboBox in the top of the window containing the data providers (and saved input data models) from which data can be retrieved. Next to the ComboBox are two DatePickers where the user can select the start and end date for the period that he want to retrieve data for. The “Get Data” button simply binds to a command that through the selected provider gets the data. The data will then be retrieved and filled into the DataGrid. There will in the current implementation be no data in the CO<sub>2</sub> field as there is currently no accessible source for that information.

To create custom data models the user can edit values in the fields in the DataGrid, by selecting the field with the mouse cursor and then typing the new value into the field. This enables the user to create their own examples of data for special events like storms or increasing oil prices.

After having edited the data the user can choose to close the window by pressing the “Cancel” button or to press “Save and Close” button to save the changed data as a custom data model. If the user chooses to save a new model, they get prompted with a dialog that allows the user to either override an existing model or save a new model with a chosen name.

## 5.4.1. Implementation of Input Data

As described in 4.2.3 *Data Handling* a shared structure for the data modules has been implemented. It is placed in the Common.Input project and both the WeatherModule and PowerModule inherit functionalities from it. This way the duplication of code is prevented and the weather- and power module can focus on implementing the functionality specific to their needs.

### 5.4.1.1. Caching

One thing that the shared structure does not handle is caching of data for better performance. The usage of the cache is part of the shared *IInputManager* (see Figure 37 for details), but the specific implementations of the cache are different. This cannot be handled by the shared structure as the data retrieved from the two types of sources are different and since the caching is done in a local database the data has to match specific table definitions. Therefore both the WeatherModule and the PowerModule are implementing their own versions of the *IInputStorageManger* (the caching manager) interface, which only the input manager has knowledge about.

The implementations of the *IInputStorageManager* used in this prototype are using local SQL Server Compact Edition Databases to cache the data. To get easy access to this database through C# the SQL Metal tool [90: Microsoft] was used, which can auto generate “LINQ to SQL”-classes that provides functionality to easily query the database with the use of LINQ.

If the data requested from the provider is not in the cache database when the data is needed, the input managers need to retrieve the data from the provider. Each of these providers is implemented separately as the source and format of the data can be very specific to each source.

### 5.4.1.2. The NordPool Power Provider

The NordPool Power Provider retrieves energy price data from Nordpool spot, the Nordic energy market. It is possible to get the energy price for each hour without tariffs, but in order to provide energy prices closer to the actual prices paid by customers the implementation automatically adds tariffs (estimated to 1,6 DKK) to the price before displaying and storing them. The NordPool data is available through a spreadsheet which can be downloaded from their homepage. The sheet contains data for a year and is updated daily at noon to include the energy prices for the following day. Below the URL structure of the spreadsheet can be seen, where “YYYY” represents that year you want to download data for.

[http://nordpoolspot.com/PageFiles/1654/Elspot%20Prices\\_YYYY\\_Hourly\\_DKK.xls](http://nordpoolspot.com/PageFiles/1654/Elspot%20Prices_YYYY_Hourly_DKK.xls)

Figure 59: The URL format of the NordPool spot energy price spreadsheet files.

When the data was inspected it was noticed that the spreadsheet was merely a mask and the real data being stored was in the form of a regular HTML Table. With the help of the *HTMLAgilityPack* [91: Mourier] the data was parsed from the table into a decent C# format.

The *HTMLAgilityPack* is an agile HTML Document Object Model (DOM) parser, which combines the DOM structure with LINQ, thereby making the DOM queryable. So when parsing the NordPool resource the HTML package can simply filter out all the unnecessary content using LINQ and only return the requested data.

In order to find the data the provider examines the *Table* object in the HTML Table and loops through each of its rows (*TR* objects). Within the columns (*TD* objects) of the rows the application finds the columns displaying the date and time for the entry and the price. These values are then added to a new *PowerEntry* object and added to the list which is returned.

#### 5.4.1.3. The DTU Weather Provider

To enable the simulation to calculate how much power the DERs provide the program needs to access information about how the weather was or will be for a given period. For this purpose Esben Larsen (Associate Professor at DTU Electrical Engineering) provided a spreadsheet containing weather information collected from a DTU weather station which is placed at Technical University of Denmark in Lundtofte, Denmark. The data contains entries for each hour of the year and contains data for all of 2011. So when looking at a date, the application can see how the weather normally would be at that specific hour of the year. As this is a prototype the application uses the weather data for 2011 regardless of what year is chosen.

By using the *DocumentFormat.OpenXml* package [92: Microsoft] the application can open the excel sheet and query it with LINQ. As a spreadsheet is just a wrapper of XML data, this package can easily parse the information into a useful object format for C#. The requested period is split into *DateTimes* separated in time by the interval specified in *StaticResources* (see 5.1.1.6 *Data Representation*) and for each *DateTime* the weather data is retrieved from the excel sheet by examining the hour in which the *DateTime* belongs.

## 5.5. Simulation

An essential part of the application is the simulation. Section 3.7 *Functional Requirements* specifies that the tool must be able to create simulations of modeled scenarios. This functionality was designed in 4.2.4 *Simulation* and the functionality and its implementation will be presented in this section.

To perform a simulation the user has to navigate to the Simulation tab, which can be seen in Figure 60. In this tab the user can either inspect previously performed simulations (See section 5.6 *Reporting*) or perform a new simulation, as described in this section. Before starting a new simulation the user has to consider the values in the five settings controls to the left of the “Run Simulation”-button.

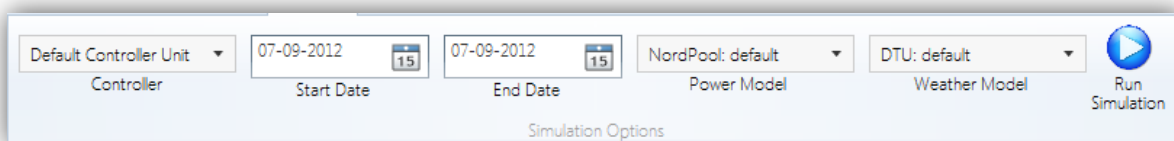


Figure 60: A part of the simulation tab of the ribbon navigation menu.

The users start off by choosing which type of controller they want to use for their scheduling. Then the users need to choose the period to run the simulation for, by setting a start and end date. Finally it is possible for the users to select which power and weather data they want to use for the simulation. Here it is possible to select the earlier created power and weather models (see 5.4 *Input Data*), such that they can simulate a user-constructed setup. To run the simulation the user presses the “Run Simulation”-button and waits for the program to notify that the simulation is finished. While waiting the user can see that the

simulation is being processed in the status bar in the bottom of the window or he can see the progress of the simulation in the window that opens when the simulation is started.

## 5.5.1. Implementation of the Simulation

All simulations are based on the five parameters shown above in addition to the canvas model containing the scenarios that should be simulated. As described in the design section the simulation is performed by the selected controller unit based on the chosen parameters. It is the intention that the selection of controller unit should behave as it would in the real world. Therefore things that are not affected by a controller unit in the real world should not yield different results when being simulated with different controller units. A controller unit can control *when* a component is turned on, but it can e.g. not control how much energy the component consumes in that period of time. This is entirely up to the component as it is component-specific information. Therefore this kind of data is calculated by the components and made available to the controller units. This also allows component manufacturers to implement component-specific information in the models that are added to the application.

### 5.5.1.1. Components Specific Data

As described in the design there are four logical types of components: DERs, Non-schedulable-, Periodic- and Constant components. See Table 5 in 3.4 *Modeling Components* for examples of which components belong in which groups. As described in 4.2.2.1 *Models* abstract implementations have been implemented for three of the four types in order to create common functionality to be used during the simulations and to let the controller units distinguish the component types. These three types will be examined in the following sections along with the DER components.

#### 5.5.1.1.1. Non-Schedulable Components

The Non-schedulable components all derive from the *AbstractNonSchedulableComponent* class (see Figure 61). As the component cannot be scheduled it is not affected by a controller unit and therefore it only exposes one public method for simulation called *GetUsageForDay* which returns a *ComponentResult* containing the consumption breakdown of the component for that given day.

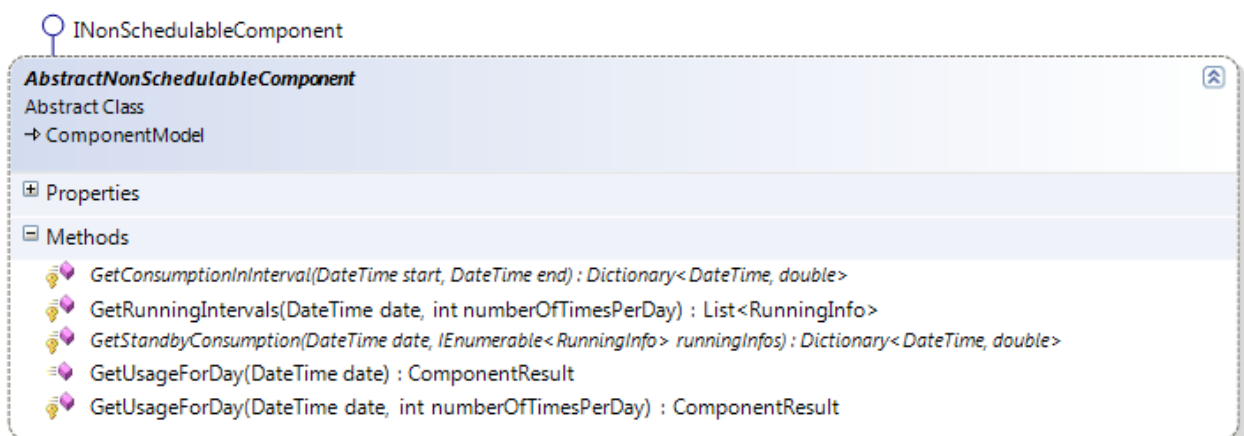


Figure 61: The *AbstractNonSchedulableComponent* class from which all non-schedulable components inherit.

The daily calculations are based entirely on the properties set on the component such as wattage, on-periods per day etc. Internally the method uses a few sub-methods called *GetUsageForDay(date,*



*numberOfTimesPerDay*) and *GetRunningIntervals* that are implemented with default implementations, but are marked as protected virtual for easy override in derived classes, as well as two abstract methods that must be implemented in deriving classes: *GetStandbyConsumption* and *GetConsumptionInInterval*. *GetConsumptionInInterval* returns the energy consumption in the chosen interval as a dictionary of DateTimes and doubles such that the energy consumption in each unit of time during the execution can be seen (see 5.1.1.6 Data Representation). *GetStandbyConsumption* returns a similar data structure describing the standby consumption for every unit of time on that given day. This way each specific type of non-schedulable component can implement a specific consumption pattern, such that TVs have a different standby consumption as well as energy consumption pattern than e.g. light bulbs. The current implementations just calculate the total amount of energy consumed in the period and distributes it equally over the time interval. The results of all of these methods are, however merged in the public *GetUsageForDay* method and returned as a single *ComponentResult* to the calling controller unit.

#### 5.5.1.1.2. Schedulable Constant Components

The schedulable constant components (related to heating or cooling) are turned on all the time (see 3.4 Modeling Components) and they expose a lot more functionality as they can be scheduled by the controller units. All the methods can be seen in Figure 62.

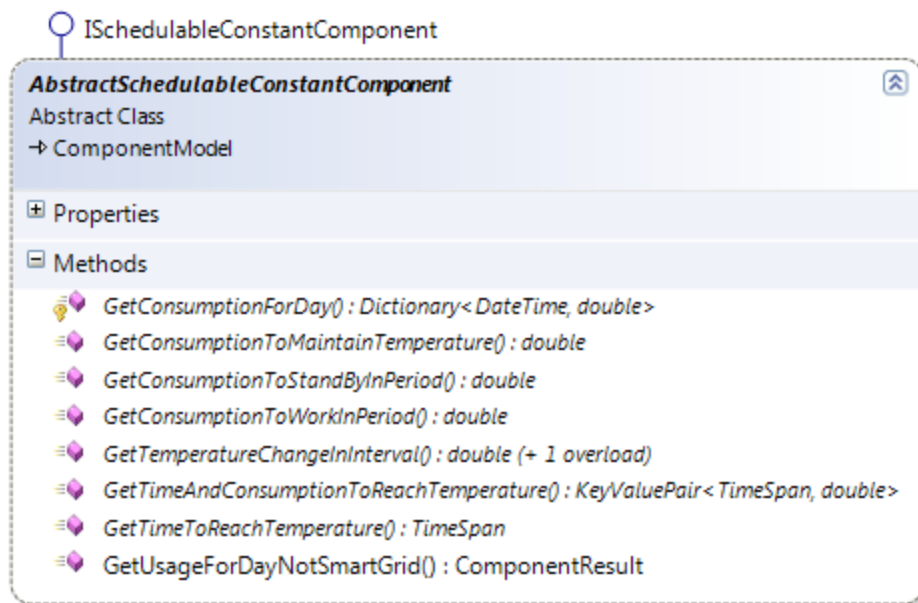


Figure 62: The `AbstractSchedulableConstantComponent` class from which all the schedulable constant components inherit.

As specified in chapter 3 Requirements, the user should be able to compare smart grid enabled scenarios with non-smart grid scenarios. This means that the components have been implemented such that they can still work in a non-smart grid environment by exposing a method called *GetUsageForDayNotSmartGrid* that works just like the *GetUsageForDay* method in the Non-schedulable components. In order to let the controller units schedule the components correctly in a smart grid environment they expose several methods that can be used to determine how fast they can change the temperature in their domain and how much energy they consume when doing so. All these methods are marked as abstract in the base class because the implementations will differ considerably between components such as refrigerators and radiators. The current implementations of the *CoolingFreezingModel* and the *HeatingModel* provide very

generic implementations of the abstract methods. The *CoolingFreezingModel's* consumption when cooling what is inside of it depends on what that material is. This is way too complex to describe for the user during modeling so the current implementation assumes that the content is 20 kg of water which appears to give reasonable consumption levels compared to data provided by manufacturers. Similarly the heating model assumes that the content of the room is air and the size of the room is passed to the model through the controller unit. Based on these two assumptions, basic thermodynamics are used to calculate the consumptions. If multiple heating models are connected to the house the controller should ensure that only a fraction of the house's size is passed to each heating model to ensure that not all heating models attempt to heat the entire house. How this is managed is up to the controller. A suggestion to an extension of how heating calculations are performed in the simulation is presented in chapter 8 *Future Work*.

#### 5.5.1.1.3. Schedulable Periodic Components

The schedulable periodic components are the components that run periodically such as washing machines and dish washers. They provide two public methods (see Figure 63). One is the *GetUsageForDayNotSmartGrid* which is similar to the same method found in the schedulable constant components. The other one is *GetConsumptionForExecutionForDate*. The component holds a property called *ConsumptionForExecution*. This property describes the energy consumption pattern during the execution of the component by returning a dictionary of DateTime and doubles that for each time unit during the execution describes how much energy is consumed in that time. The property is marked as abstract and must be overridden by the deriving classes to provide a model specific consumption pattern. The only periodic component of the current implementation, *WashingModel*, returns an evenly distributed consumption over the entire execution period. The *GetConsumptionForExecutionForDate* method uses this method to return the actual energy consumption pattern of the component for the given day.

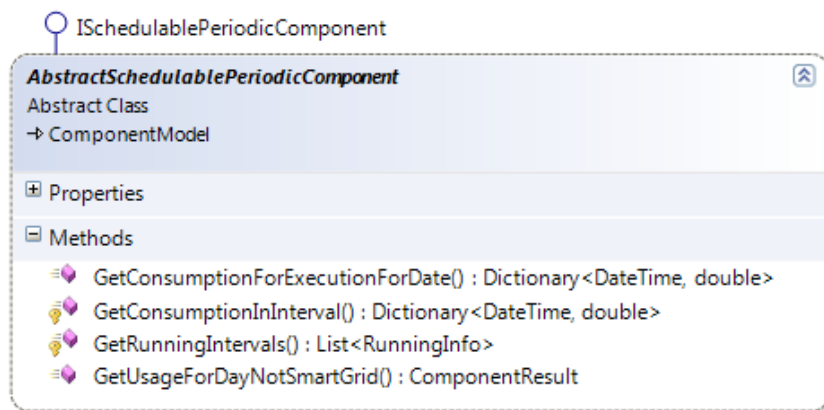


Figure 63: The *AbstractSchedulablePeriodicComponent* class from which all schedulable periodic components inherit.

#### 5.5.1.1.4. Wind Turbines

The DER components are so specific that they do not share an abstract implementation of the *IDERComponent* interface. The interface requires that the model implements a method called *GetGeneratedPower* that based on a weather model can calculate the power generated by the DER.

Power generation from wind turbines vary depending on the wind speed as seen in Figure 64.

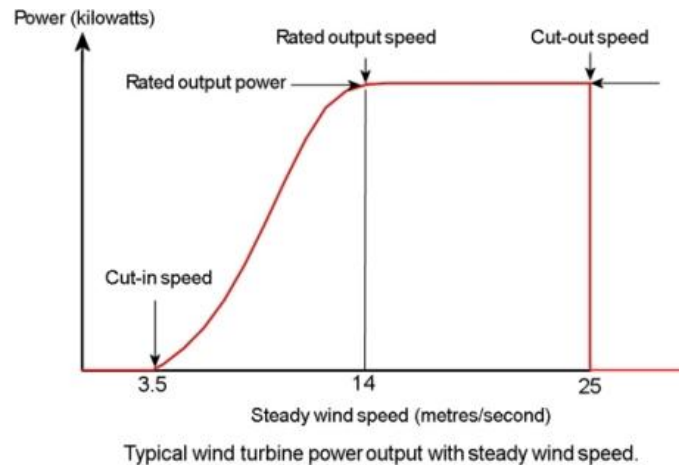


Figure 64: A chart showing how the power generated by a wind turbine depending on the wind speed. Source: [93: Wind Power Program]

When the wind is below a certain threshold (the cut-in speed) the wind turbine does not generate any power. Up until another threshold called the rated output speed the wind turbine generates power at a reduced rate. Finally, the wind turbine generates its maximum power when the wind speeds are between the rated output speed and the cut-out speed which is the maximum wind speed the wind turbine can handle. If this threshold is exceeded no power is generated [93: Wind Power Program]

The power available in an air stream can be calculated using the formula:

$$\frac{1}{2} \rho U^3 \frac{\pi d^2}{4}$$

where  $U$  is the wind speed,  $\rho$  is the density of the air stream and  $d$  is the diameter of the rotor [93: Wind Power Program].

Each wind turbine in the application must define the cut-in speed (Minimum wind speed), the rated output speed, the cut-out speed (Maximum wind speed), the rated output power (the maximum power generated under optimal conditions) and the rotor diameter (see Figure 65). Based on these values the formula above has been implemented in the *WindTurbineModel's* method *GetGenerationInWatts*, taking the actual wind speed and air temperature from the weather model. The air density is calculated based on the air temperature using numbers from DTU Risø [94: Friis] to give a reasonable estimate of the air density.

The percentage of available power converted into energy is called the wind turbines *efficiency*. According to *Betz limit*, which is a theoretical maximum limit for how much of the available power that can be used by a wind turbine, a wind turbine can never have an efficiency above approximately 59% [95: Wind Power Program]. The efficiency is calculated in the wind turbine model by dividing the rated output power (i.e. the power generated when running at optimal wind speed as specified by a property) with the available power calculated using the optimal wind speed. If this efficiency is greater than 59% the application will simply use 59% instead. The actual generated power is then calculated by multiplying the actual available power with the efficiency, and if the wind speed is below the rated output speed (i.e. it is on the slope of the curve) the value will also be multiplied with the actual wind speed divided by the rated output speed, thereby

assuming that the power generation increases linearly between the cut-in speed and the rated output speed.

The *GetGeneratedPower* method calls the *GetGenerationInWatts* method for each *DateTime* entry in the weather model in order to get the power generation over the required time interval. A *ComponentResult* is then created and returned containing the generated power for each entry with a negative sign, indicating that the power is generated rather than consumed.

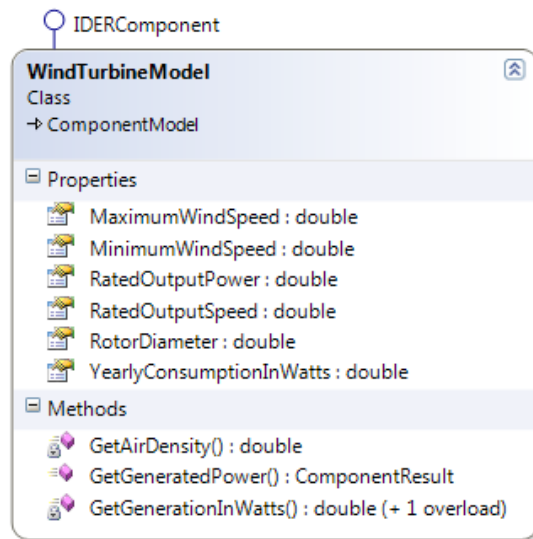


Figure 65: The *WindTurbineModel* representing the model version of a wind turbine.

A number of wind turbines have already been added to the toolbox based on actual wind turbines from the manufacturer ProCure [96: ProCure]. All of these wind turbines achieve a calculated efficiency of around 20% which fits very well with actual home wind turbines [97: Kidwind Science Snack].

The estimated yearly consumption used by the summary (see 5.3 *The Summary Area*) has to be calculated without performing a simulation, which means that the value has to be found without examining the actual weather data. In order to provide a reasonable estimate the average wind speed and air temperature of Denmark has been retrieved from the Danish Meteorological Institute (DMI) [98: Danmarks Meteorologiske Institut]. The power generation is then calculated once for the average wind speed and temperature and multiplied by the number of hours in a year.

#### 5.5.1.1.5. Solar Panels

Solar panels are typically described by a wattage per square meter which represents how much radiation it can convert into electrical power if radiated by 1 kW/m<sup>2</sup> [99: Moll]. This value is typically between 100 and 250 W/m<sup>2</sup> [100: RenSMART] and is in the implementation of a solar panel called the *WattagePerSqMeter*. Based on this value the *efficiency* of the solar panel is calculated by dividing the wattage with 1000 [99: Moll]. The electrical power generated by the solar panel is direct current. In order to convert it into an alternating current that can be used in a home, an inverter must be attached. The efficiency of this inverter is called the *InverterEfficiency* and is typically between 80% and 95%, depending on the amount of power being converted [101: Solar Choice]. The *SolarPanelModel* implemented in this thesis can be seen in Figure 66.

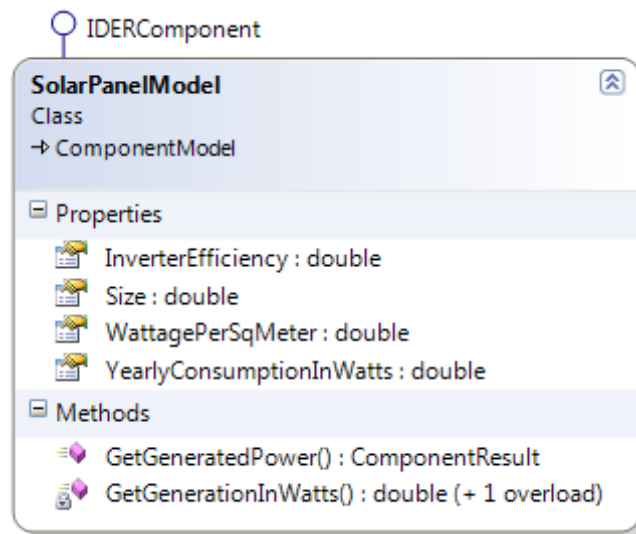


Figure 66: The `SolarPanelModel` class which represents the model for all solar panels in the application.

The solar panel model works exactly like the wind turbine model when it comes to calculating generated power, however the method used for calculating the actual power generated for each time entry in the simulated interval, `GetGenerationInWatts`, is of course different. The calculation is based on the solar irradiance retrieved from the weather model for the specific point in time. The generated power is calculated as the product of the Sun's radiation (in  $\text{W}/\text{m}^2$ ), the size of the solar panel (in  $\text{m}^2$ ), the efficiency of the solar panel and the efficiency of the inverter.

#### 5.5.1.2. The Controller

As mentioned earlier the application can contain multiple implementations of controllers that all use the models described above to perform simulations, but the way in which they perform the simulations can differ significantly. This thesis has created a default implementation of a controller unit which tries to schedule the modeled components using all the available information that the solution offers. This version of a controller unit should also be seen as an inspiration to programmers who might create additional controller units in the future.

The current default implementation performs a simulation by examining each scenario and for each scenario examining each day separately. It examines the components one type (using the Abstract model classes described in 5.5.1.1 *Components Specific Data*) at a time and calculates the production or consumption of each component of the given type. It starts by calculating how much energy the DERs have generated. A list of the generated power is passed on to the next components, so the scheduling can attempt to place energy consumption in the periods where energy is being generated from the DERs. The elements in the house will attempt to use this energy first and if anything is left after all components are scheduled it will be sold and send back to the grid. Below the six steps of a simulation for one day are shown.

### 1. DERs

First it gets the generated power from the DERs. It uses the weather model together with the concrete calculations for each DER. This step is a prerequisite for the scheduling of the components and must therefore be done first.

### 2. Non Schedulable Components

Secondly it gets the consumption per day per component for the non-schedulable components. Since the components cannot be scheduled they do not depend on the other steps either. By running them first ensures that the power from the DERs is consumed by these components when they need it. This way other components are not falsely scheduled based on a high DER consumption that is consumed by the non-schedulable components.

### 3. Schedulable Constant Components - Not Smart Grid Enabled

It then gets the consumption per day per component for the non-smart grid enabled constant components. As the components run constantly there is only one running interval which contains the entire day and the consumption is spread evenly all over this interval. As these are not smart grid enabled the same applies to them as to the non-schedulable components.

### 4. Schedulable Periodic Components - Not Smart Grid Enabled

The fourth step is to get the consumption per day per component for the non-smart grid enabled periodic components. As the component runs X number of times per day, the running intervals are placed evenly within the on-period of the day. As these are not smart grid enabled the same applies to them as to the non-schedulable components.

### 5. Schedulable Constant Components - Smart Grid Enabled

Then the consumption per day per component for the schedulable constant components is calculated. It uses the *GetSchedulableConstantConsumption* function (Read more about later in this section) to find the times of the day where the component is allowed to turn off depending on its own temperature. It find these interval by looking at when it is most expensive to be turned on based on the daily energy prices and any leftover generated power. As they consume energy most of the time it is harder to schedule these to consume energy at the most beneficial times and therefore they are run before the periodic components such that they can use the DER power. If any DER power is left it is easier to schedule the periodic components to run when the energy is available than to schedule the constant components.

### 6. Schedulable Periodic Components - Smart Grid Enabled

Finally it gets the consumption per day per component for the schedulable periodic components. It uses the *GetSchedulablePeriodicConsumption* function (Read more about later in this section) to find the times of the day where it is cheapest for the component to be turned on based on the daily energy prices and any leftover generated power.

The *GetSchedulableConstantConsumption* function is used by step number 5 to find the intervals of the day where the constantly active components can be turned on/off to save energy. It starts by finding the most expensive intervals of the day and then uses this information to find out how much time it needs to heat/cool to the highest/lowest allowed temperature depending on the type of the component (Heating / Cooling). Then it looks at how much on-time is required in order to cool/heat to compensate for the temperature lost/gained during the planned off times. The cheapest periods matching the required on-time

are then found and scheduled as on-periods. Finally it calculates the remaining times where it has to be in standby to keep the temperature steady.

The ***GetSchedulablePeriodicConsumption*** function is used by step number 6 to find out when a periodic component should be turned on. The controller knows that the component is smart grid enabled, how many times it needs to be run each day, when it is allowed to run and its execution duration. With this information the controller goes through each entry in the daily energy prices and gets the prices for running the component in that interval throughout the allowed period of the day. This way the controller gets a list of potential intervals and the price for running the component in that interval. By sorting this list in ascending order of price, it takes the first interval, schedules an execution in this period by creating a running info, and removes the intervals in the list which are overlapping the selected interval. It repeats this action as many times as the component needs to run each day. These running intervals are then returned.

For each type of component a list of *ComponentResults* is collected containing the scheduled consumption for each component for each day. These component results are added to the scenario result of the scenario which is added to the final simulation result and returned when the simulation has been completed.

#### 5.5.1.2.1. Performance during Simulations

As mentioned in 3.6.4 *Performance* the performance of the simulation is a potential issue. To get the simulation to run as smooth and fast as possible the application runs some parts of the simulation in parallel. By using the *System.Threading.Tasks* namespace, which was added in .NET Framework 4, you can produce concurrent foreach loops in a very simple way. Microsoft has added simple methods that handle the thread pools, killing of threads and scheduling of threads. The main class in this namespace is the *Task* class, which represents an asynchronous operation that can be waited on and cancelled. This class is used to retrieve power and weather data simultaneously before starting the scheduling. The static *Parallel* class from the namespace contains a *ForEach* method that takes an Enumerable (list) and an action as input. It performs the action on each element inside the Enumerable in parallel.

To speed up the simulation the controller uses this method to run the simulation of all the scenarios concurrently and inside the action for each scenario the simulation of each day in is run in parallel, resulting in two nested *Parallel.ForEach* calls. The simulation can run the days concurrently as the days do not affect each other with the current implementation, but other implementation could handle this differently and would then not be able to run days concurrently. The components do not run in parallel as they should be able to take the generated power into consideration and react on how other elements are scheduled such that they all are not scheduled to run at the same time.

## 5.6. Reporting

Based on every simulation performed, one or more reports can be made. As mentioned in the Design chapter (see 4.2.5.1 Reporting) a report consists of an overview of the simulation result along with a number of charts. The reporting window lets the user see and edit the reports of a simulation. The reporting window can be found either by clicking on the “Show Report” button when a simulation is finished or by choosing an older simulation in a drop down menu in the ribbon and clicking on the “Inspect Selected Simulation” button (see Figure 67).

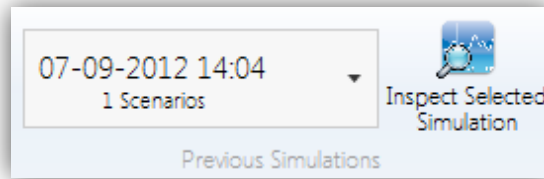


Figure 67: A selection of the simulation tab in the ribbon displaying the functionality to let the user examine previous simulations.

The reporting window can be seen in Figure 68.

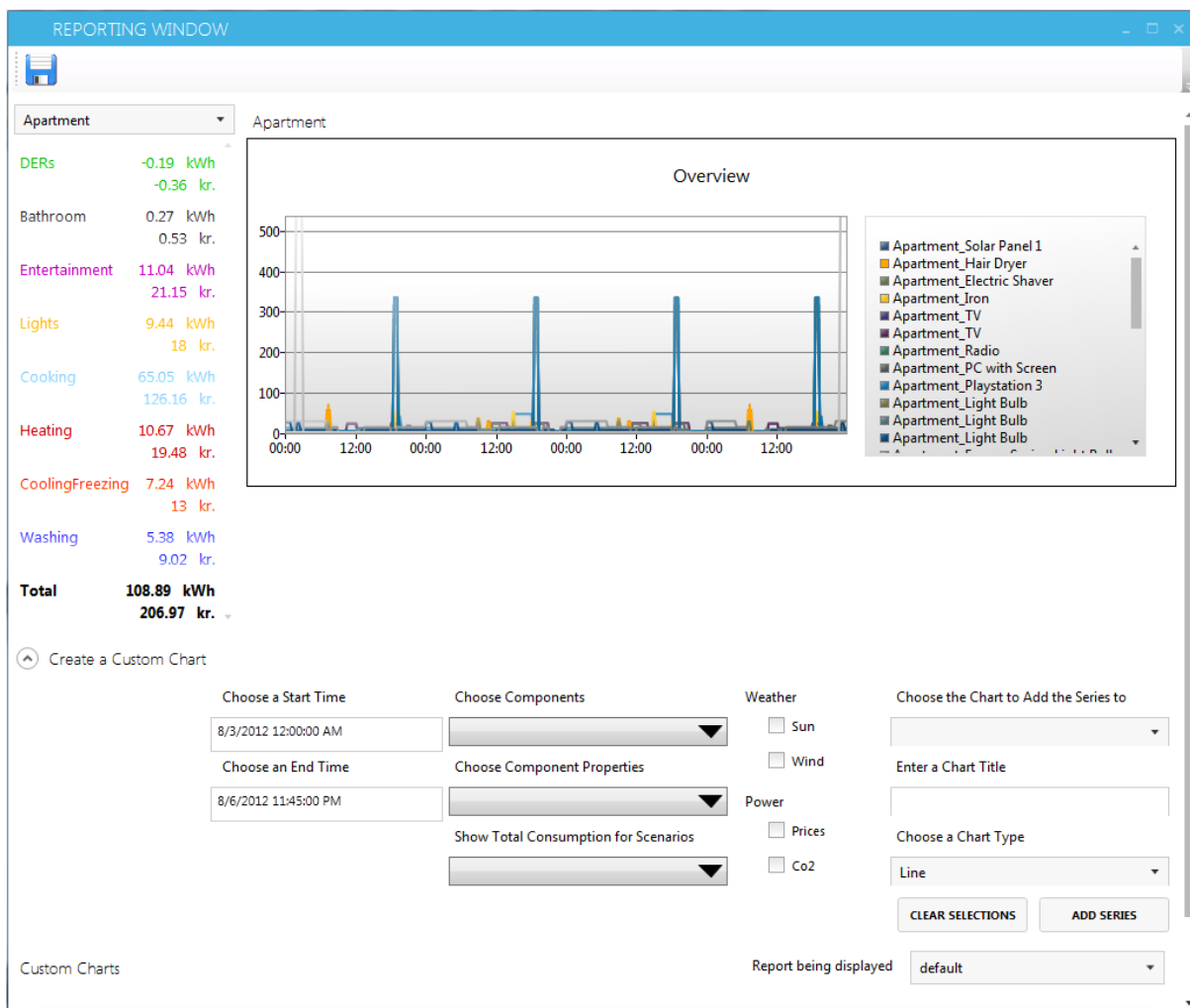


Figure 68: The reporting window used to analyze the results of a simulation.



At the top left of the window is a drop down menu saying “Apartment” in. This menu lets the user choose which scenario in the simulation result to show the overview for (in this screenshot the name of the chosen scenario is “Apartment”). The chart called “Overview” shows the consumption pattern of each of the components of that scenario in the simulated interval. The x-axis shows the time and the y-axis shows the consumption at the given time in Wh. To the left of the overview chart is a view very similar to the summary. This area works similarly, but instead of showing the estimated yearly consumption it shows the consumption in the simulated interval for each category as well as the total price for the energy used by each category. These values are from the actual simulation rather than estimations and should therefore be more accurate than the ones found in the summary in the main window (this is examined in 6.2.3.2 *Component Validation*). These categories are also shown in the specific color of the category as in the summary. Upon mouse-over of each category an estimation of the yearly consumption, price and CO<sub>2</sub> emission can be seen, calculated by scaling up the simulated results to match a year. Therefore the longer the simulation period is, the more accurate these results become (see Figure 69).

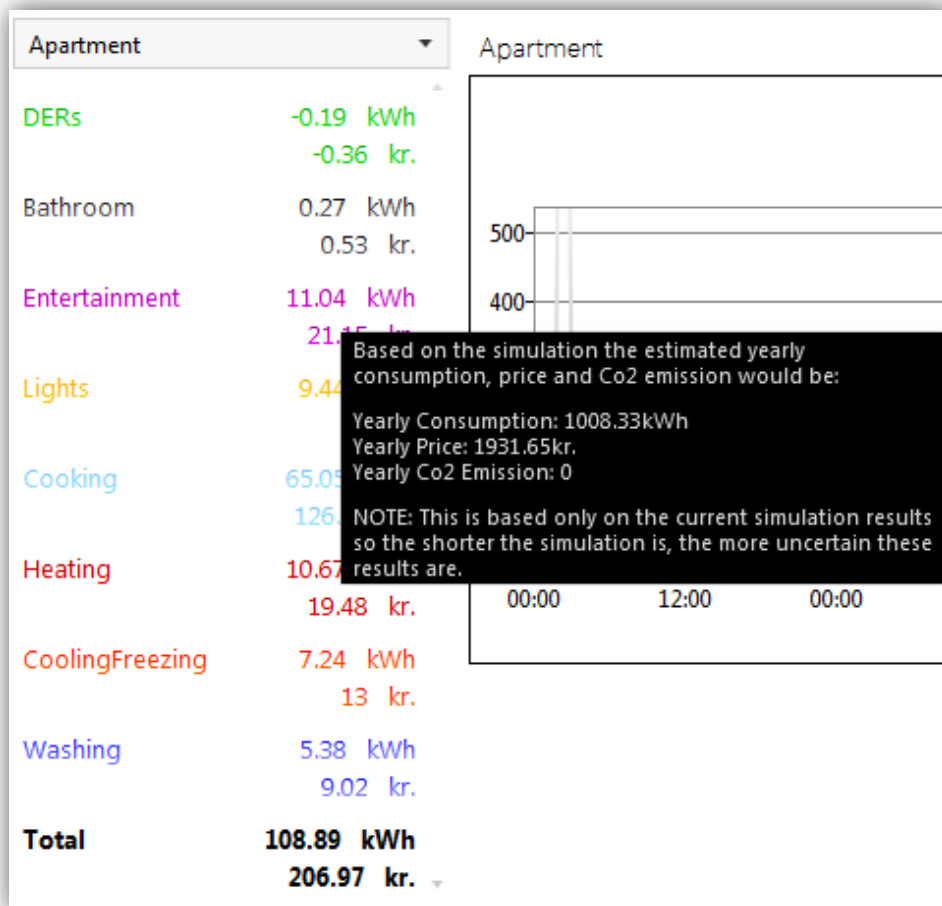


Figure 69: A screenshot showing the tooltip displayed when hovering the mouse of a category in the overview.

Below the overview is a region called “Create a Custom Chart”. It provides the user with controls to create charts based on the simulation results (see Figure 70).

Figure 70: A selection of the reporting window that shows the panel in which the user can create custom charts.

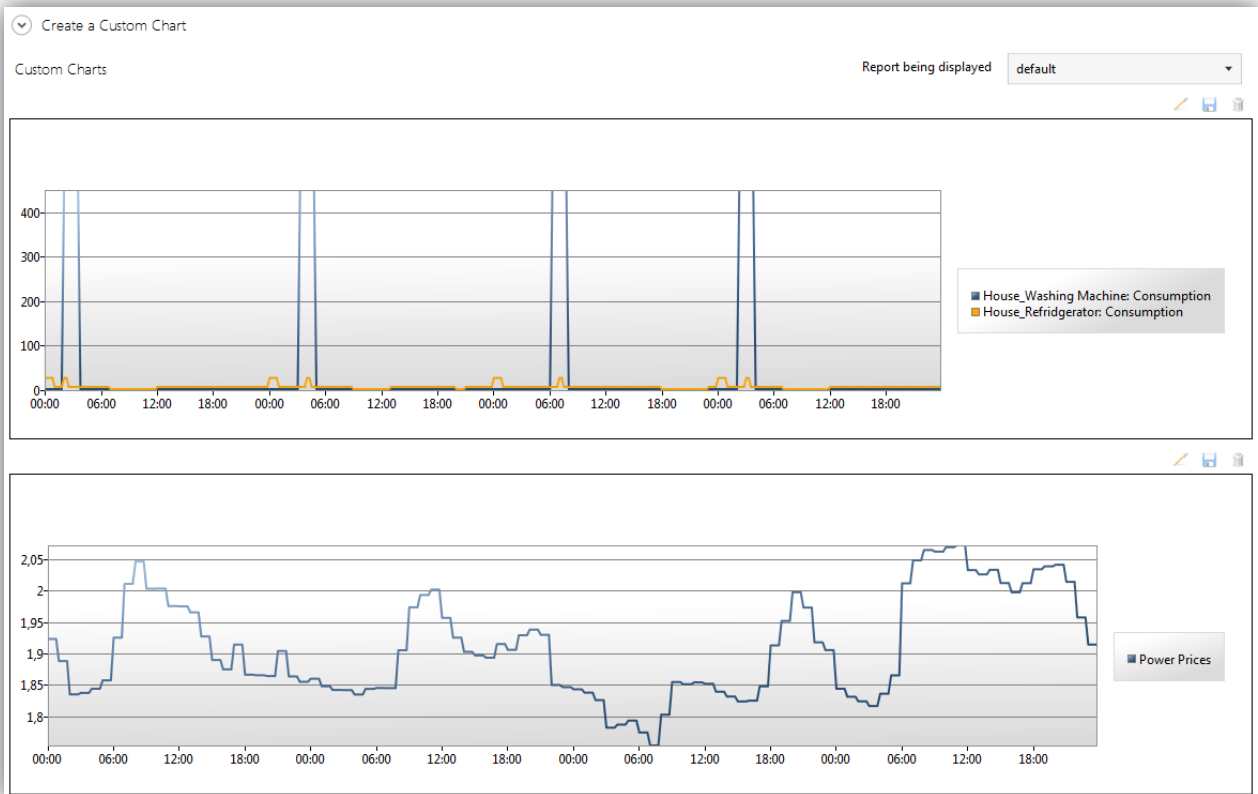
The charts can be generated from a wide range of parameters. The first column of controls lets the user choose the start and end time of the data for the new chart. By default this is the first and last data entry of the simulation result. The second column lets the user select which data series to add from the model. In the first drop down the user can select which components to include in the chart. The second drop down holds the properties to be displayed for the selected components such as price, CO<sub>2</sub> emission, consumption or on-time intervals. The last drop down lets the user add a data series to the chart that shows the total consumption of the scenarios selected in this drop down. All three drop down menus allow for multiple elements to be chosen at the same time.

In the third column the user can choose which input data to display in the chart such as wind speeds, energy prices etc. By adding these it becomes easy to see how the components have been scheduled compared to the fluctuating prices and weather. The final column specifies information about the chart that should hold the series specified in the other columns. The first drop down lets the user add the series to an existing chart or to choose to add them to a new chart. If “New Chart” is chosen (i.e. leaving the value empty) the user can specify a title and a chart type (line, bar, pie etc.) for the chart. Clicking the “Add Series” button creates the series and/or chart specified and adds them to the “Custom Charts” region.

Figure 71 shows two examples of charts that have been created. The first one contains series showing the consumptions over time for the refrigerator and washing machine in the scenario. The second chart shows the fluctuating energy prices in the same period. It can be seen that the washing machine has been run when the energy prices were lowest<sup>2</sup>. It is important to note that all types of series (consumption, prices, wind speeds etc.) can be combined in the same charts. This creates a very strong flexibility when creating charts, but also means that the y-axis of the custom charts is without unit as it can at the same time be kr., Wh, m/s and more. Additionally, some series, such as consumption of washing machines, will show very

<sup>2</sup> Note that the x-axes the two charts are shifted compared to each other due to the sizes of the chart areas

large values whereas some series, such as energy prices, will show very low values, making these series impractical to have in the same chart. Therefore it can be a good idea to separate them into two charts and compare these as done in Figure 71.



**Figure 71:** A selection of the reporting window showing the Custom Charts section in which the charts created by the user are displayed. Note that the charts have different widths so the x-axes are shifted compared to each other.

In the upper right corner of each chart are three buttons: Edit, save and delete. The delete button removes the chart from the report. The save button is intended to save the chart as an image, but the functionality has not been implemented in this prototype so the button is currently disabled. The edit button opens an *Edit Chart* window (see Figure 72) in which the user can change the name and type of the chart as well as remove series from the chart by clicking on the trash can next to the series.

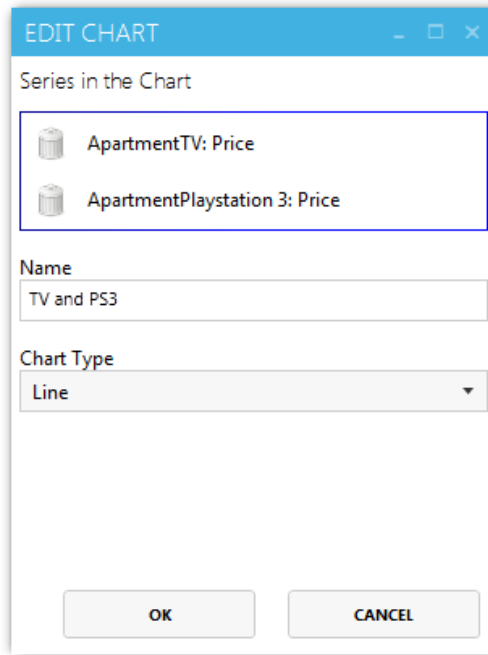


Figure 72: The view in which the user can edit an existing chart.

When a report has been created with the desired charts it can be saved using the save icon in the upper left corner of the reporting window. This lets the user choose a name for the report and when it has been saved the user can toggle between which report to show, using the drop down menu just above the region with the custom charts. Since reports are connected to a single simulation result only reports connected to the same simulation result can be displayed this way. If reports connected to another simulation result should be displayed the reporting window must be closed and the desired simulation result should be found in the drop down menu in the ribbon using “*Inspect Selected Simulation*” as explained earlier in this section. It is, however, possible to have two reporting windows open at the same time showing different simulation results if comparison across simulations is required.

### 5.6.1. Implementation of the Reporting Module

The most important aspect of the Reporting Module is the classes representing the charts. As mentioned in the Design chapter (see 4.2.5.1 *Reporting*) a report is represented by a *ReportInfo* object. Each custom chart in the report is represented by a *ChartInfo* object and each series in the chart is represented by a *SeriesInfo* object (see Figure 43). These classes, however, only contain information about what is stored in the charts, not the actual data being displayed. The actual data is stored in the simulation result and when a chart needs to be displayed the *ChartInfo* object must be converted into an object that contains the actual data. In order to facilitate, this three classes have been implemented called *ChartDataSet*, *DataSeries* and *DataEntry*. These classes correspond to the aforementioned Info classes as seen in Figure 73.

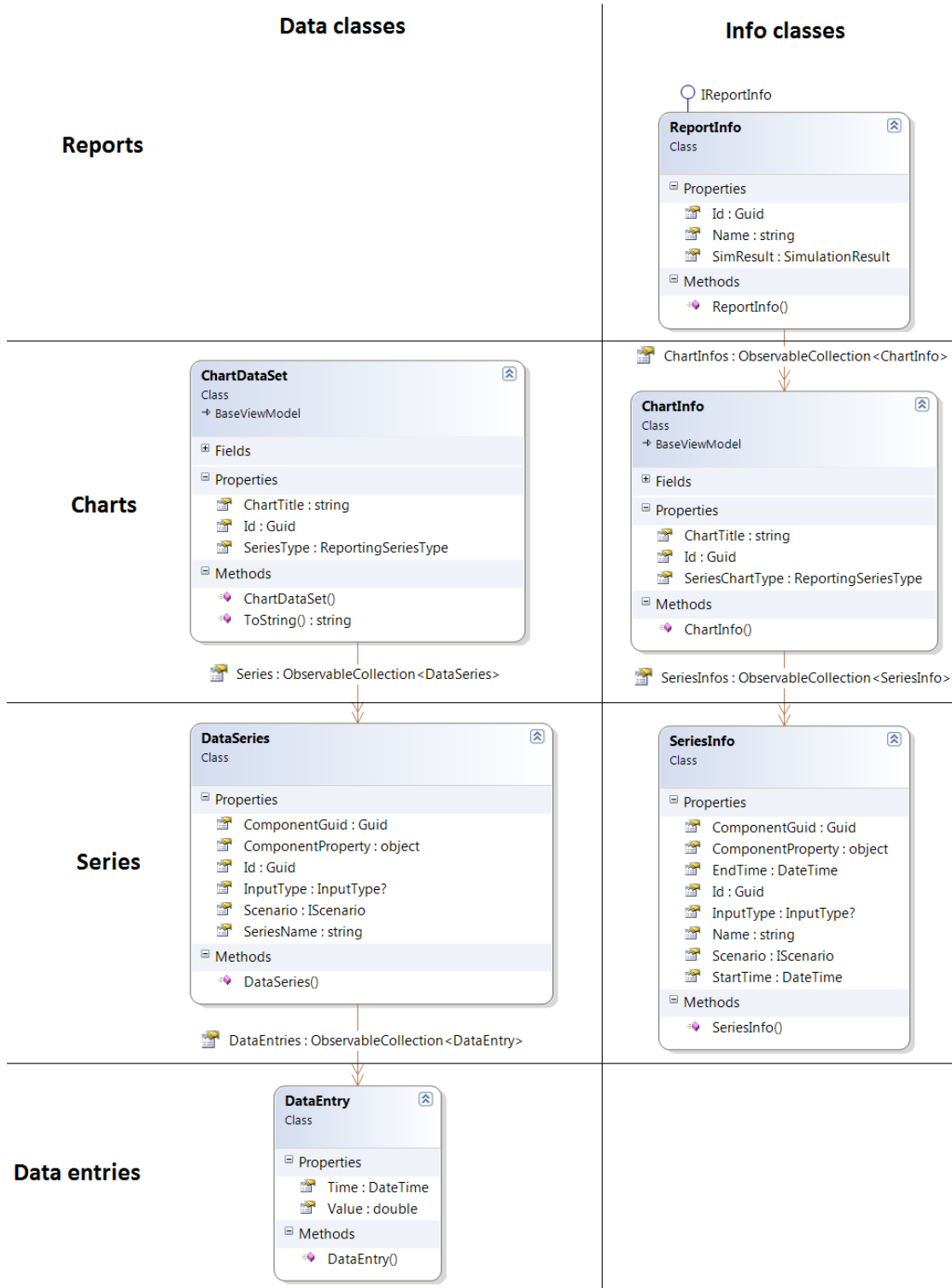


Figure 73: The classes used to represent reports, charts and series as well as the classes used to represent the actual data.

*DataEntry* represents one data point in a series and consists of a *DateTime* representing the point in time the data point represents and a value (*double*) describing the value of the point. Using a class to represent data points makes it a lot easier to data bind the charts the data rather than using dictionaries of *DateTime*, *double*. Additionally, it is better from a software engineering standpoint in that it encapsulates the

behavior of the element in an object making maintenance and extensions easier. All the data points describing a data series are stored in a collection of data entries in a *DataSeries* object. This class represents the series being displayed in a chart and therefore it contains the data as well as metadata such as a name of the series as well as information about what the data represents, e.g. an Id of a component. If the *Scenario* property of the *DataSeries* is set it means that the series displays data for a scenario rather than just a component. Similarly if the *InputType* is set it means that the series represents input data such as energy prices or weather data.

Finally all the series stored in the same chart are saved as a collection of *DataSeries* in a *ChartDataSet* object. Like the *DataSeries* class it contains metadata about the chart such as the type of the series in the chart (line, bar etc.) and a title of the chart. By storing the type of the series in the *ChartDataSet* rather than in the *DataSeries* class it is easier to enforce that all series in a chart are of the same type. This prevents situations where the user creates a line series in a pie chart etc., which does not make any sense and would result in unpredictable behavior.

#### 5.6.1.1. *Initializing the Reporting View Model*

As described above the reporting window depends on the simulation result from the simulation. This object is injected into the reporting view model on creation by using the view creation pattern described in 5.1.1.4 *Displaying Views as Windows* to display the window. When the simulation result is injected into the view model the first thing it needs to do is to merge the component results in the simulation result. The controller unit has created a separate component result for each component for each day of the simulation so all component results that are connected to the same component must be merged. Then the view model creates the overview chart and summary. The summary is created by grouping the merged component results based on the components' categories and creating *Consumption* and *ScenarioConsumption* instances for each group, much like in the Summary module. These are, however, not the same classes used in Summary as they need different functionalities and properties. The approach is the same, though. For each group the price of using the energy in the pattern defined by the simulation in the given interval is calculated by multiplying each entry in the energy consumption in kWh with the price per kWh retrieved from the power model (CO<sub>2</sub> emission is calculated similarly). Next each group estimates the yearly consumption, prices and CO<sub>2</sub> emission by examining the number of seconds simulated and scaling the results up to match a year.

The overview chart is created by creating an instance of the *ChartDataSet* class that contains a *DataSeries* object for each component in the scenario. The data in the data series is the consumption over time for the component and is retrieved from the merged component results as dictionaries of DateTimes and doubles. This is converted into a collection of *DataEntry* objects when added to the data series in order to make it easier for the charting framework to display the data as described above.

#### 5.6.1.2. *ChartHelper*

In order to display the chart in the view the WPF charting framework is used as mentioned in 4.2.5 *Data Visualization*. The framework does not support data binding the series in the chart to a collection, and therefore a *ChartHelper* class has been implemented, inspired by an implementation by Jeremiah Morrill [102]. This chart helper provides static dependency properties that can be attached to the chart such that it can data bind to collections of series as seen in Code Snippet 14.

```

<chartingToolkit:Chart DataContext="{Binding Path=SelectedScenario.OverviewDataSet}"
    local:ChartHelper.DependentValueBinding="Value"
    local:ChartHelper.IndependentValueBinding="Time"
    local:ChartHelper.Title="{Binding Path=ChartTitle}"
    local:ChartHelper.SeriesType="Line"
    local:ChartHelper.Palette="{DynamicResource ChartPaletteWithNoDataPoints}"
    local:ChartHelper.SeriesSource="{Binding Path=Series}"
    Height="300" />

```

Code Snippet 14: Using the *ChartHelper* class to data bind the chart to a collection of *DataSeries*.

Whenever the collection of data series changes for a chart in the view model the *SeriesSource* dependency property in the *ChartHelper* class is notified and an event handler is invoked. This event handler clears the series on the chart connected to the dependency property and adds the series that are in the collection bound to the dependency property.

All the series will use the *Palette* specified by the *Palette* dependency property. This property specifies how the series should look. Although using different colors for each series is default functionality in the charting framework this functionality is lost when you change any appearance aspect of the palette. The charts in the reports, however, simply have too many data points and having each data point displayed as a large dot on the line is not useful. Therefore it was necessary to implement a new palette that provides various colors for the series as well as removes the data points and attach this palette to the chart using the chart helper. Removing the dots have the unfortunate effect that it is no longer possible to hover the mouse over the line to see the value at the specific point, but the graphical benefits make up for this loss.

### 5.6.1.3. Creating Custom Charts

When the user defines a custom chart using the controls in the reporting window he can select a start and end time of the data to display. These dates are entered in standard text boxes. Ideally a date and time chooser would be used, but as such a control does not exist in the WPF toolbox a text box was used rather than developing a new control. The text entered into the text box is validated as a date time value and must therefore adhere to the standard American way of representing dates and time, i.e. the format:

MM/DD/YYYY HH:MM:SS AM/PM

Whenever a value is changed in any of the custom chart controls the value is pushed to properties in the view model using data binding, so when the user clicks “Add Series” the reporting view model extracts the data entered in the controls through these properties and creates a *ChartInfo* instance containing *SeriesInfos* for each series defined. If for example the user chooses one component and two component properties, such as price and consumption, two series will be created containing the price over time for the component and the consumption of the component, respectively. By first creating a *ChartInfo* object the code used to create the overview chart and to recreate saved charts in loaded reports based on *ChartInfos* can be reused to create the custom charts. This *ChartInfo* instance is passed to the *CreateChartDataSet* method which creates a *ChartDataSet* object based on the information in the chart info by extracting the corresponding data from the simulation result and saving it in *DataSeries* instances. Finally this chart data set is added to the *CustomDataSets* collection.

A separate control has been created to represent a custom chart in the view. It contains a chart using the chart helper as described for the overview chart and three buttons binding to Edit-, Save- and DeleteChart commands in the reporting view model. By having each chart as a separate view control an infinite list of

charts can easily be displayed using an `ItemsControl` in the reporting view as seen in Code Snippet 15. This control binds to the `CustomDataSets` collection and displays all the `ChartDataSet` instances as charts using the custom chart control.

```
<ItemsControl ItemsSource="{Binding Path=CustomDataSets}">
  <ItemsControl.ItemsPanel>
    <ItemsPanelTemplate>
      <StackPanel Name="ContentPanel"/>
    </ItemsPanelTemplate>
  </ItemsControl.ItemsPanel>
  <ItemsControl.ItemTemplate>
    <DataTemplate>
      <Views:CustomChartView Margin="0,5,0,0"/>
    </DataTemplate>
  </ItemsControl.ItemTemplate>
</ItemsControl>
```

Code Snippet 15: The `ItemsControl` displaying all the `ChartDataSets` in the `CustomDataSets` collection as instances of the `CustomChartView` control that uses the `Chart` control from the `Data Visualization` namespace.

#### 5.6.1.4. Saving a Report

When the user saves the report all the custom charts, stored as `ChartDataSets`, are converted into `ChartInfos` and the underlying `DataSeries` are converted into `SeriesInfos`. If the user chose to overwrite the currently open report these `ChartInfos` are added to the active `ReportInfo` in the reporting view model and if not, they are added to a newly created `ReportInfo` object which is then stored as the currently active report in the view model.

## 5.7. Projects

As described in 3.7 *Functional Requirements* the application must provide a way of saving and loading projects. The definition and design of this functionality was described in 4.2.1.3 *Projects*. This functionality will be presented in this section.

The program enables the users to save their projects so that they can return to them at a later point in time. This is done by using the `ProjectModule` which keeps track of the status of the active session. The `ProjectManager` in the `ProjectModule` contains the property called `ActiveProject`, which is of the type `Project`. The `Project` class contains a reference to the currently active `ICanvasViewModel` (containing the modeling content on the canvas) and a collection of `SimulationResult` which is all the simulations that have been performed in this project.



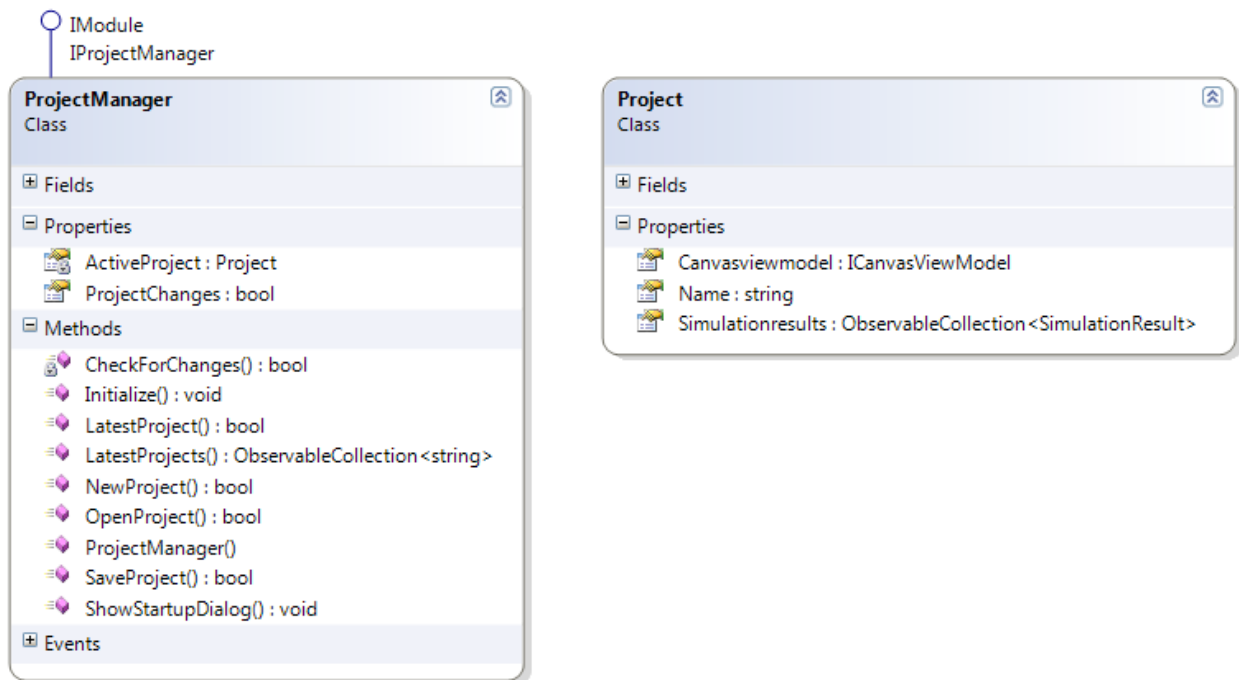


Figure 74: The classes in the *ProjectModule* responsible for controlling sessions.

There are four main functionalities within the *ProjectModule*; Saving, Loading, Creating (new) and keeping track of the latest opened projects. It is possible for the user to access these functionalities from three places in the program, which can be seen in Figure 75. Additionally the functionality can be accessed through keyboard shortcuts following the standard Windows conventions:

- Ctrl+N: New Project
- Ctrl+S: Save Project
- Ctrl+O: Open Project

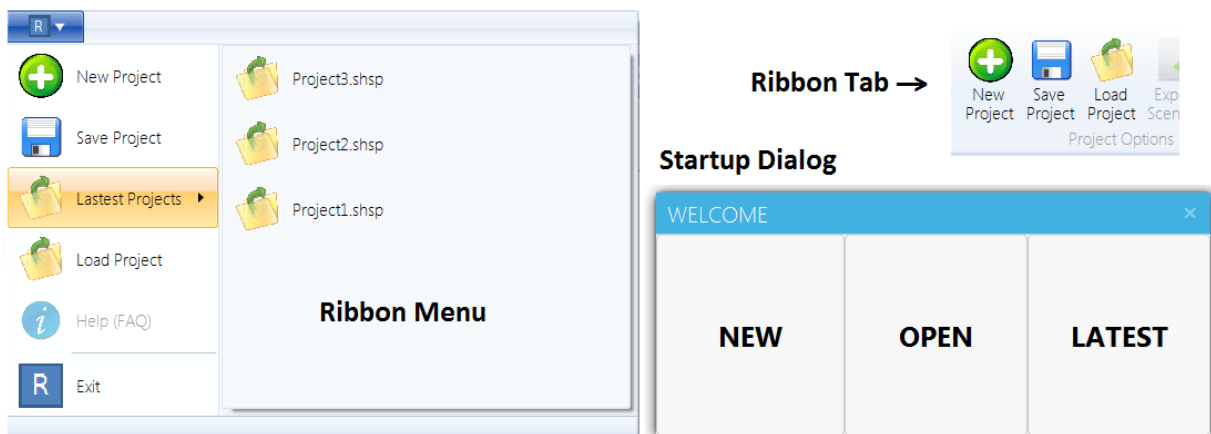


Figure 75: The three different places in which the projects can be created, saved or loaded.

It is possible to access these functionalities from a number of places as the buttons in the menus just bind to composite commands in the *GlobalCommands* class which is available to all modules. The *ProjectManager* then simply attaches implemented functionality to these commands.

As mentioned earlier the *ProjectManager* keeps track of information about the open project by having an *ActiveProject* property. When the user saves or loads a project, the *ProjectManager* uses serialization to serialize the *Project* instance into a file. In order to serialize the project and all of its content every class in the class hierarchy from the *Project* and down to the component models and component results have been marked as *Serializable*. The properties/commands which are not serializable are marked with the *[NonSerializable]* attribute and all of these will be initialized in a Initialization method, which will be called by the module when loading when the project has been deserialized.

When a project is saved it uses the container to retrieve the currently active instance of the *ICanvasViewModel* and the *ISimulationManager* through which it can retrieve the performed simulations. When the program loads a saved project it uses the EventAggregator in Prism to publish events (*CanvasViewModelReplacedEvent* and *SimulationResultReplacedEvent*) containing the loaded canvas model and simulation results stored in the saved project. The two other modules (Canvas and Simulation) are subscribed to these events and when the events are published they replace the current canvas model and simulation results with those published with the event.

Lastly the *ProjectManager* is responsible for keeping track of changes to the project since the last save operation. This enables the application to ask the user if he wants to save the changes before closing a project. The Project manager is notified of changes by the canvas- and the simulation modules when changes have been made to canvas content or when a simulation has finished. This is done in the respective modules by resolving the *IProjectManager* interface from the container and setting the *ProjectChanges* property to *true*. The *ProjectManager* then examines this property when the user asks to close the current project.

## 5.8. Usability Adherence

Thesis goal 11 of this thesis specifies that chosen user interface design standards must be adhered to. In section 3.6.5 *Usability* elements from the ISO standard 9241-171 were introduced along with other usability concepts as the guidelines for the user interface design. This section examines how the usability requirements specified in section 3.6.5 *Usability* have been met in the current prototype implementation and how well it complies with the ISO 9241-171.

The eight suggestions from the ISO standard that were presented in 3.6.5 *Usability* and the way they are covered in the current implementation, and thereby how the non-functional requirement *usability* is fulfilled, can be seen below:

- **Meaningful element names**

The names of buttons such as save, load, new, copy etc. have been used in coherence with the Windows platform to ensure that functionality of the button is obvious to the user and that the same word does not have multiple meanings in the application.

- **Location of elements to be platform coherent**

The application is designed such that commonly known visual elements are placed, ordered and named as they are in Windows (The only supported platform), to make it more familiar to the user. An example is the buttons for saving and closing windows, which are placed in same order as in the dialogs from Windows.

- **Never only color to convey message**

The application uses colors to convey messages on three areas: Energy production vs. consumption of components, categories of components on canvas and to differentiate series in charts. Energy production is also marked with a negative sign to convey the message through other means than color. The category of a component can still be seen by editing the component and examining the Category property. Differentiating series in a chart can, although, only be done through colors as nothing else is supported by the chosen charting framework.

- **Optimize task steps**

To minimize the user's steps when performing regular tasks such as modeling and simulation the application provides easy access to the relevant functionality in just a few steps. Simulations can e.g. be performed by setting a start and end date in the ribbon and clicking "Run Simulation". The task steps involved in modeling are reduced by using drag and drop.

- **Undo/redo**

The application has implemented undo/redo functionality in the modeling section as described in section 5.2.3 *Undo/Redo*.

- **Alternative inputs methods**

All across the program both combo boxes and calendar selectors are being used for selection data with limited value options. For example are all enumerable values selectable through combo boxes.

- **Clear error messages**

To inform the user of errors or issues when using the application, some error handling has been made which in a simple matter notifies the user of what is wrong. An example of this is in the editing and creating windows of elements, where text fields get highlighted with a red border, if the input is not of a valid format.

- **Multiple access to functionality**

The application provides multiple ways of accessing the same functionality. An example of this is when editing elements on canvas which can be done from both the ribbon or from a context menu when right clicking. Another example is when opening/saving projects which can be done from both the ribbon, from the program menu and through keyboard shortcuts (See section 5.7 *Projects*).

- **Group navigation controls based on functionality**

By using the ribbon navigation method, the functionalities are divided logically into separate ribbon tabs. For example are all simulation functionalities placed in the *Simulation tab* (See section 4.2.1.2 *Navigation*).

An overall evaluation of the application's adherence to the ISO 9241-171 standard has been conducted and the full conformity checklist can be seen in Appendix E. The goal has not been to provide full conformity as the application is a prototype, but the application still conformed well in categories such as "Names and labels for user-interface elements (8.1)", "General control and operation guidelines (8.4)" and "Window appearance and behavior (10.5)". Areas in which the application can be improved in terms of conformity to the standard include "User preference settings (8.2)", "Compatibility with assistive technology (8.5)" and "Support services (11.2)".

Section 3.6.5 *Usability* also described five ways in which usability could be improved described by other sources than the ISO standard. How these points have been met is described below:

**1. Drag and drop**

Drag and drop has been used in the modeling section of the application to enable users to model their homes by dragging elements from the toolbox onto the canvas as described in 5.2.2.1 *Drag and drop*.

**2. Keep users informed**

The status bar described in section 4.1.4 *Modules* informs the user when simulations are running. Similarly the simulation status window which is opened when a simulation is started shows step by step how far the simulation is such that the user can see that the application is not stuck.

**3. Match between real world and system**

The meaning of the words “scenario” and “component” might be a bit unclear to new users, but the lack of precise, real world definitions led to the use of these words. By using user-defined names for the actual components and scenarios the names can be specified to match real appliances and units, though. Additionally words such as “energy prices”, “weather information” and “simulation” have been used in accordance to the real world meanings of the words in order to prevent confusions.

**4. Avoid unnecessary information in dialogs such as rarely used fields etc.**

It would be useful to provide advanced and simple view versions of e.g. the *Create Component* window. Currently it contains a lot of information and default values for the advanced settings could be provided and hidden in the simple version. This has, however, not been implemented in this prototype due to time restrictions, but should be considered before a release.

**5. Walkthrough Usage guide**

The application is partially implementing guidance through the program, which could be improved in further development. The application is currently guiding the user by writing a simple message on the canvas on how to start modeling a scenario with components.

Although this is a prototype the adherence to usability requirements has been important and many points and suggestions have been met. Some areas are, however, not implemented yet as they either do not apply to this type of application or they do not make sense to spend time on until the application is extended to be released.

## 5.9. Conclusion

Based on the design decisions made in chapter 4 a prototype of an application has been developed that fulfils most of the requirements set in chapter 3.

The chapter began by introducing the general concepts and implementations that provide the foundation of the application such as describing how the Prism library is used, the role of Common shared project and a description of shared resource classes such as the *IconProvider*. Then the mechanisms for how the application opens and closes windows without breaking the Model-View-ViewModel pattern were described. A design pattern developed by this thesis was introduced for opening and closing windows that ensures that all logic remains in the view models and is thereby kept out of the view.

To fulfil the non-functional requirements of *customizability* and *extensibility* the application can load externally made DLL assemblies containing new content in various areas of the application. The assemblies just need to implement a given interface (e.g. *IControllerUnit*) and the application can then use the classes in the DLL based on that interface. In order to enable dynamic addition of new components to be used during modelling the elements are saved in an XML format and are parsed into view models using reflection. This also allows external parties to extend the application with their components without making any code changes.

To perform as precise simulations as possible the simulation will require detailed information about the components and contain very specific calculations for each type of components. Therefore the components inherit functionalities from an abstract layer that contains calculations for how the different types of elements consume energy. The controller unit can then depend on each component to calculate its own consumption and only worry about scheduling the components correctly. The chapter describes how the simulations are performed by describing the calculations within the components and how the default implementation of the controller unit's scheduling is done by looking at price peaks and possible generated power from Distributed Energy Resources.

Finally the chapter summed up how the non-functional requirements *usability* described in 3.6.5 *Usability* has been met in the current prototype implementation and how well the application conforms to the ISO 9241-171 standard about user interface design.



## 6. Test

---

In order to verify that the application performs the implemented functionalities correctly the application has been tested in multiple ways. Testing can be done in many ways and each type of test has an area in which it is suitable. The application developed in this thesis has been tested using unit tests, manual tests, case studies and lastly it has been demonstrated to GreenWave Reality for feedback. This chapter will describe the approach taken to each type of testing and some of the results will be analyzed.

The developed tests are based on the functional requirements defined in *3.7 Functional Requirements* in order to ensure that the application implements the required functionality.

The main focus of a prototype is to prove how the functionality can be implemented rather than to ensure that the functionality is 100% robust. As this application is a prototype it has therefore been tested mostly using positive tests. Positive tests use positive inputs are used to ensure that the module behaves correctly and yields the right results when being used as intended. The negative input tests ensures that the application does not crash or reach an unsafe state when the user does something unintended, a connection unexpectedly close etc. Some negative tests have been created, but the focus has been on positive tests.

### 6.1. Unit Tests

Unit tests are highly useful for testing elements that yield predictable results. Therefore a large number of unit tests were developed that test various situations in the models. These tests were mostly done in the modeling module as this module contains all the logic of the components and scenarios and these are obvious candidates for unit testing. Both black box tests and white box tests have been created. Black box tests test the functionality as seen from outside of the module, i.e. tests the public methods [103: Software Testing Fundamentals] whereas the white box tests test the internal workings of the module, i.e. the private methods [104: Software Testing Fundamentals]. White box tests have been created using generated “*accessor*” classes that provide the test files with access to the class’ private methods.

It is never possible to test all combinations of input and therefore the test cases have to be chosen with great thought and input values that could potentially break the application (or in the case of performance tests, slow it down unreasonably) should be identified. Negative input values that has been used often in the tests are null values, empty lists (such as in the case of adding a component to a canvas without a scenario) retrieving non-existing Ids from lists etc.

Unit tests have been performed for the following modules:

- Common
- Common.Modelling
- ModellingModule
- CanvasModule

- ToolboxModule
- PowerModule
- SimulationModule
- WeatherModule

The other modules do either not contain any unit testable content or are covered by other unit tests or test approaches described later.

140 unit tests have been created and the overall results and the number of unit tests in each module can be seen in Figure 76. The full results showing the name and success of each unit test can be seen in Appendix F.

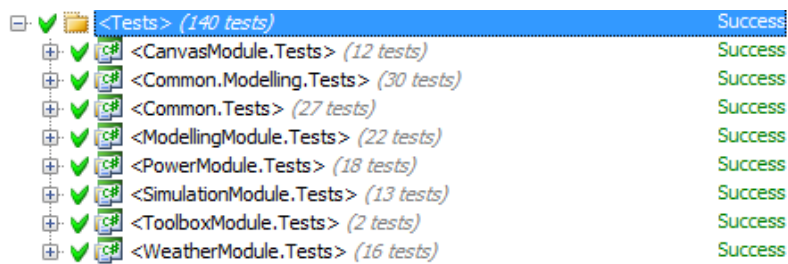


Figure 76: Overview of the 140 unit tests that have been created to test the application.

The unit tests were developed for each module after the initial implementation of the module. As the modules were mostly developed one at a time it meant that the unit tests of all modules could be run after a new module was created or a new feature was implemented in order to ensure that the functionality still worked.

### 6.1.1. Sorting Performance

Due to the Z-Index bug on the canvas causing the connections to be drawn on top of the scenarios and components, sorting of the elements on the canvas was introduced as described in 5.2.2 *Canvas*. In order to ensure that this does not hurt the performance, tests were developed to test the time it takes to sort the elements in various setups. These tests were developed as unit tests and placed in the *Common.Tests'* *StaticResourcesTest* class because the sorting method is part of the *StaticResources* class. The sorting was tested in seven different setups and the time it took to sort the elements was calculated using the *StopWatch* class. The results can be seen in the table below measured in milliseconds. It is clear that in all realistic situations which will probably include a couple of scenarios with up to 100 or so components on each the performance is still very good. Sorting 3 scenarios with 100 components connected to each scenario only took 4 milliseconds which will have no effect on the perceived effect from the user's point of view.

Number of Scenarios	Number of Components per Scenario (results in same number of connections as well)	Number of Total Elements on Canvas	Sorting Time (in ms)
1	1	3	1
1	100	201	2
1	1000	2001	37
2	3	8	1
3	100	603	4
10	50	1010	10
100	100	20100	3620

Table 8: The time it takes to sort the components, scenarios and connections on the canvas in various setups. The tests were run on a Lenovo T510 laptop.



## 6.2. Manual Tests

Some features in the application are not suitable for unit testing. This is primarily the graphical functionalities such as opening or closing a window, expanding a menu, ensuring that a chart is rendered correctly; basically all functionalities that cause a graphical change, but does not represent a concrete value in the models or view models. Another area of functionality that does not work well with unit tests is the simulations. For this kind of situations this thesis has used a manual testing approach where the tests have been performed “by hand”.

In this section the manual tests that have been performed are presented and the results of these are evaluated.

### 6.2.1. GUI Tests

In order to test that the graphical user interface works as expected tests were performed manually by running the application and examining how the application reacted in various situations. Common situations that were tested includes clicking buttons, moving windows, tabbing between windows, closing windows, resizing windows, opening drop down menus and selecting elements in them etc. All expected workflow paths through the application were attempted multiple times using different inputs in order to verify that the GUI behaved correctly in each situation.

One of the things identified during the GUI tests was that when resizing the main window the data bindings in the ribbon were lost. This turned out to be a known bug of the ribbon class provided by Microsoft. This would be very difficult to find during unit tests, but was quickly discovered using manual tests and a workaround for the bug was created.

### 6.2.2. Chart Performance

As described in *4.2.5.1 Reporting* the currently used charting framework does not perform very well on large datasets, but was the best option available that did not require purchase of a license (see Figure 44). When the datasets become large, which typically happens when displaying the overview of a long running simulation in the reporting view, the GUI thread blocks while rendering the chart. This cannot be prevented as the chart must be rendered on the GUI thread in WPF. Sadly it can take a very long time to render the chart as it tries to draw every single point in the dataset which in the case of a simulation lasting a year is 35,040 data entries per component. A more intelligent approach from the charting framework would be to combine multiple adjacent data points into one when they will not be distinguishable as separate data points in the chart anyway. These merged data points would then be passed to the WPF renderer that would then only draw those data points rather than drawing all of the 35,040 points per component. When zooming in on the chart the framework could provide a new dataset with unmerged data points to still provide the necessary detail level. The actual approach used by the commercial charting frameworks is unknown as the companies obviously do not want to expose this information, but it is likely that some of them are using an approach like the one mentioned above to improve the performance over what can be achieved by the basic WPF charting framework used in this thesis.

In order to illustrate the performance issues with the charting framework a number of tests have been conducted in which the reporting view was opened using datasets of varying sizes to see when the performance drops to the unbearable. The time it took from the Show Report button was clicked until the

reporting view was showing the chart and was responsive was measured. The tests were performed multiple times for each setup and an average time was found. The results of these tests can be seen in Table 9.

Number of Components:	Days Simulated:	Number of Data Points:	Time to Render Chart:
1	1	96	< 0.5 sec
1	7	672	2 sec
1	30	2,880	11 sec
10	7	6,720	1 min 16 sec
10	30	28,800	6 min 4 sec
10	365	350,400	Out of Memory
50	365	1,752,000	Out of Memory

**Table 9: The time it takes to render the reporting view. Only tests with one scenario have been created as the overview chart only shows data for one scenario. The number of data points is calculated as 4 quarters/hour \* 24 hours/day \* days simulated \* number of components.**

As can be seen from the results the rendering speed is rather low when the number of data points go up and anything beyond 5,000 data points seems impractical for real use. Many real homes will, however, have 50 or so components which means that only 1 day of consumption can be rendered at a remotely satisfying speed. Additionally, the application runs out of memory when attempting to render data for a year with 10 components (350,400 data points). Based on these results it is obvious that an improved charting framework is of high priority if the application is to be developed further with the prospect of a future release.

### 6.2.3. Simulation Tests

The simulations performed in the application rely considerably on large data sets and semi-random and random elements that make it very hard to predict the precise output of the simulation. This means that unit tests have no concrete value to compare the simulated output to, in order to evaluate if the unit test passed or failed which makes the simulations very hard to test properly using unit tests. Therefore this functionality has been tested by stepping through the code during execution to ensure correct behavior and the results provided by the simulation have been compared to approximated consumption results of the components from real life use. An example of a source for such information is natlex.dk [105: Natlex]. Additionally the simulations were tested using case studies of real homes from which the actual yearly consumption was known in order to verify that the simulations provide realistic results. For more information about the case studies see 6.3 Case Studies.

#### 6.2.3.1. Performance Benchmarking of the Simulation

One of the focus areas identified in the Requirements chapter (see 3.6.4 Performance) for applications such as this is performance as simulations in general can be very time- and resource consuming. Therefore performance benchmarking of the simulations was conducted in the end of the project period by recording the time it took to perform simulations with different scenario sizes and simulation periods.

The benchmarking was conducted on a fast internet connection with 95.16 Mbps download and 126.28 Mbps upload on a cabled network. Stationary desktop computers were used with the following specifications.

1. Intel Core i5 660 @ 3.33GHz
2. 3.49 GB RAM
3. Normal 5400 rpm HDD.
4. Windows XP Professional with Service Pack 3

The results of the benchmarking can be seen in Table 10. The number of scenarios (s) and components (c) can be seen horizontally and the simulation periods can be seen vertically. The results are in the format *minutes:seconds.milliseconds*.

Days\S-C	1s 10c	1s 20c	1s 40c	1s 60c	1s 75c		2s 5c	2s 10c	2s 20c	2s 50c
1 day	0:13:42	00:15.02	0:13.69	0:15.45	0:15.30		0:14.23	0:13.99	0:13.37	0:13.73
1 week	0:14.06	00:13.86	0:15.69	0:14.85	0:15.21		0:17.02	0:13.93	0:13:55	0:14.77
1 month	0:18.17	0:18.97	0:22.99	0:25.72	0:27.53		0:18:42	0:21.03	0:23:95	0:33.31
6 months	2:04.66	2:20.31	5:03.23	6:02:68	6:46.84		2:11.90	3:13.47	5:21.13	10:38.35
1 year	6:15.05	10:34.45	18:23.66	25:11.25	28:03.74		8:51.38	11:21.42	19:49.17	44:06.47

Table 10: The performance tests of simulations of various setups.

All simulations were conducted without using the cache such that all the simulations had to retrieve the power and weather data from the providers and thereby make the test results comparable. It was recorded that the fetching of the power data in general took around 7 seconds and parsing the weather and power data files took around 1.20 minutes for one year. The time to parse the data scales with the amount of days, but the retrieval of power data does not, as the entire year has to be retrieved every time (see 5.4.1.2 *The NordPool Power Provider*). It should be noted that the same dates were used for all setups of the same length to ensure comparability. Finally it was tested that the order of how the simulations were run did not influence the results e.g. that running the *1 day* simulation before the *1 year* simulation did not yield different results than running the *1 year* simulation first.

When examining the benchmarking results it can be seen that when simulating only one day the number of elements are not affecting the simulation time, but when simulating for longer periods the number of elements becomes a huge factor. When keeping the simulation time constant (examining the table horizontally) the time to fetch the input data is constant as the same input data has to be retrieved in all the scenario configurations. This indicates that the overhead associated with retrieving data overshadows the time it takes to perform the simulation when performing short simulations such as 1 day or 1 week. When simulating longer periods, however, it can be seen that the time to perform the simulation increases significantly when increasing the number of components, indicating that in longer simulations the number of components become the determining factor, overshadowing the overhead of retrieving input data.

The currently implemented controller unit is creating a thread for each scenario and for each scenario it creates a thread for each day of the simulation. Inside the “daily” thread it calculates the consumption for each component sequentially. The daily thread will take longer time to execute the more components the scenario is containing and this increased time for each thread will then be multiplied with the number for days and number of scenarios when simulating bigger modeled setups. This partly explains why the large setups take so long to simulate.

Another explanation of the increasing simulation time in larger setups is a technical bottleneck in the simulation. This bottleneck is a *lock* on the final simulation result list which each daily component result is

added to when the daily threads/calculations are done. To prevent synchronization problems the application is creating a critical region which locks the result list while one thread adds its component result to it. The time inside this critical area grows when the amount of components (and thereby component results to add) increases and therefore the time other threads have to wait is also increasing. The longer waiting time will therefore slow down the thread pool in its scheduling and starting of new threads for the following days of the simulation.

### 6.2.3.2. Component Validation

The simulation results rely highly on the accuracy of the implementation of each component's consumption pattern. As mentioned in the scope of the thesis (section 1.4) the focus of this thesis was not to make accurate implementations of the components as this would take far too much time, but rather to create some general implementations that provide reasonable results and let accurate implementations be a possible future development. It is, however, still interesting to ensure that the components' consumptions are not too far from the truth. Therefore manual tests have been performed in which the most important components (at least one of each model type) has been simulated for a year and the estimated consumption used by the summary as well as the simulated consumption is compared to general consumption estimations from the real world retrieved from natlex.dk [105: Natlex], the JRC European Commission [106: JRC European Commission] and Renewable Energy UK [107: Renewable Energy UK]. The results can be seen in Table 11.

Component	Estimated Consumption (kWh)	Simulated Consumption (kWh), smart grid enabled	Reference Consumption (kWh)
Dryer (2150 W, 1h / week)	137.33	123.62	112
Washing Machine (1800 W, 4h / week)	344.9	330.75	281
Electric Radiator (600 W)	1606	2131.9	1752
Refrigerator (100 W)	206.83	182.7	228
Chest freezer (110 W)	347.3	276	361
Electric Shaver (15 W 5m / day)	0.45	0.45	0.5
TV (100 W, 3h / day)	127.7	124.55	110
PC with screen (150 W, 3h 20m / day)	226	200.8	183
Energy saving light bulb (11 W, 5h / day)	19.7	20	20
Light Bulb (40 W, 5h / day)	71.75	73	73
Electric Kettle (2000 W, 6m30s / day)	77.7	79	79
Oven (2000 W, 15m / day)	179.4	178	183
Solar Panel (6 m <sup>2</sup> )	-1037.2	-1039.7	-1140
Wind Turbine (400 W)	-162.5	-266.6	-294

**Table 11: The estimated and simulated consumptions of most of the components compared to values from external sources. The wind turbine reference value is from Renewable Energy UK [107: Renewable Energy UK], the solar panel value is from JRC European Commission [106: JRC European Commission] and the rest of the values are from natlex.dk ([105: Natlex]. The rows marked with grey are the ones that provide the least accurate results.**

It is important to note that the consumption depends highly on the properties set on the components such as wattage and on-time. As natlex.dk [105: Natlex] allows for setting both wattage and on-time on the components it is very suitable for comparing the values to the ones found in the simulation.

Table 11 shows that most of the components provide very reasonable results for estimated consumption as well as the actual simulated consumptions. For most of the components the simulated results are closer to the reference source than the estimated results, which is to be expected due to the increased level of detail available when performing simulations compared to static estimations. A couple of components stand out though. The electric radiator (heating model), the refrigerator (cooling/freezing model) and the chest freezer (cooling/freezing model) provide reasonable results in the estimations, but are somewhat off when

being simulated. This is most likely due to the fact that the simulations are performed in a smart grid environment in which the components are turned on and off based on the energy prices. This means that the internal cooling/heating cycle of the components is modified. As no smart grid enabled heating or cooling/freezing models exist in the commercial market it is difficult to anticipate how they will work and react to external changes to the cycle. Therefore the simulations are difficult to make accurate for such component types. The estimations are per definition made on non-smart grid enabled environments as smart grid results cannot easily be estimated without performing simulations and are therefore closer to the reference values for these kinds of components.

It should also be noted that the wind turbine provide inaccurate estimations, but accurate simulated values. This is due to the fact that the power generated by a wind turbine depends on the current wind speed, but it does not scale linearly with this. In order to make fast estimations, however, the estimated values are done using a mean wind speed value which means that it expects a constant wind speed of around 5 m/s. This does not provide the same output as you would get with varying wind speeds as in the simulation, but is a rough estimation. It can be seen in the table that the simulated values are much closer to the reference values retrieved from Renewable Energy UK [107: Renewable Energy UK] and are within an acceptable range.

Lastly, it can be seen that the solar panels provide values reasonably close to the real values. The estimated and simulated values are very close to each other as they use the same weather data and do not have the same issue as the wind turbine when using averaged data in the estimation. Although the values are approximately 100 kW lower per year than the real values they must be considered acceptable as this depends highly on the weather information used for the given year and is therefore expected to vary.

All in all the components provide reasonable results and as the focus of the thesis was not to provide accurate simulations for each type of component, but rather to provide an implementation that can easily be extended with accurate implementations of later on the results are satisfying.

### 6.3. Case Studies

As described in the vision of the thesis (see 1.2 *Vision*) the application attempts to help inform the customers of potential effects of the smart grid which is presented as a prerequisite to rolling out the smart grid by *Smart Grid Netværket's* suggestion 7 [5: Klima- Energi- og Bygningsministeriet]. In order to show how the application can be used for this purpose two case studies have been created based on real homes. The real energy consumptions of both homes are known and can be used to validate the results provided by the simulation. Both cases have been simulated over a period going from the 1st of January 2011 until the 31<sup>st</sup> of December 2011, such that the results can be compared to the values found on the real energy bills.

The case studies are models of two real homes, one family house and one three room apartment. Every component in the house was recorded together with its wattage and how its usage patterns were. All information was then modeled in the application with the recorded settings. Lists of the components within each household can be seen in Appendices G and H. The two screenshots of the summary window below show the estimated yearly consumption values of each category.

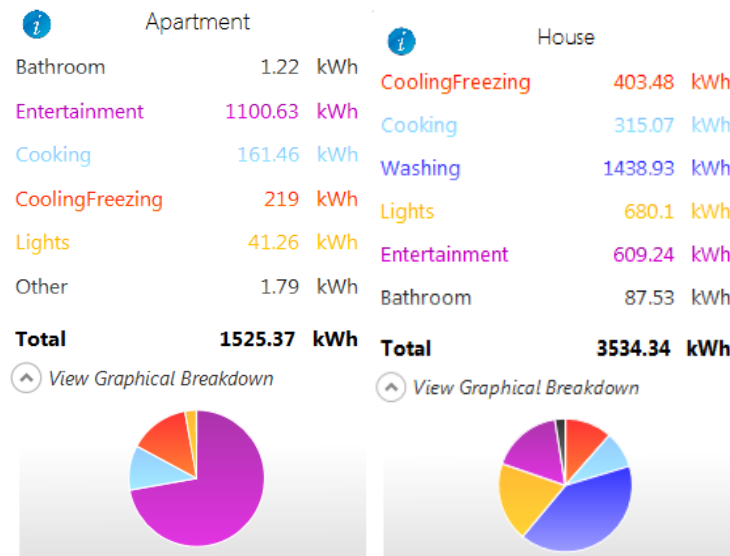


Figure 77: The estimated yearly consumption of the two case studies as seen in the Summary view of the application.

The actual consumption from the energy bills for the two cases was 3100 kWh for the **House** and 1500 kWh for the **Apartment**. When comparing the estimated summary values with the actual consumption for the home it can be seen that the estimations are not that far off and the results are acceptable when noting that the application is a prototype and therefore still has work to do in terms of details of each component’s consumption simulation.

For each of the case studies five setups were created that can be compared in order to identify the financial differences between smart grid and non-smart grid setups. The setups reflect the stages in which the homes can be during a transition from a regular setup to a fully smart grid enabled setup. The five setups can be seen below.

1. Without Smart Grid enabled appliances and with flat rate energy prices
2. Without Smart Grid enabled appliances and with fluctuating energy prices
3. With Smart Grid enabled appliances, with fluctuating energy prices
4. With Smart Grid enabled appliances, with fluctuating energy prices and with one solar panel
5. With Smart Grid enabled appliances, with fluctuating energy prices and with one wind turbine

When looking at the results of the simulation the consumption values vary from simulation to simulation. This is caused by random factors in the controller unit which are implemented to reflect real life decisions and actions of the consumer. Therefore the results of the simulations vary slightly in the categories that can be controlled by the controller unit. The results are presented in the next two sections for each case study.

### 6.3.1. Case Study: House

Figure 78 shows the results of the simulations performed on the five setups of the house case study. It can be seen that simply upgrading to fluctuating energy prices (going from setup 1 to 2) can save the house for 581.17 DKK per year<sup>3</sup>. If the home also upgrades to smart grid enabled appliances (setup 2 to 3) the house

<sup>3</sup> It should be noted that using variable energy prices currently comes with an additional yearly fee which with all likelihood outweighs this saving (see 1.1.1.2 The Power Market)

would be saving an additionally 536 DKK per year. This is a total saving of 1117.70 DKK per year which is a quite significant saving, but it requires the owners of the house to buy both smart grid enabled appliances and purchase and setup a home area network that contains a controller unit.

An example of how the scheduling of the appliances in the house has resulted in cheaper consumption (moving from setup 2 to 3) can be seen in the *CoolingFreezing* category where average price per kWh is 1.96 DKK/kWh in setup 2 and 1.92 DKK/kWh in setup 3 where the consumption is moved to the cheapest periods. Similarly the *Washing* category has a price of 1.99 DKK/kWh in setup 2 and 1.85 DKK/kWh in setup 3.

1)		2)		3)	
Cooking	307.84 kWh 646.47 kr.	Cooking	335.34 kWh 674.18 kr.	Cooking	283.34 kWh 572.44 kr.
Lights	691.95 kWh 1453.08 kr.	Lights	692.86 kWh 1382.19 kr.	Lights	691.26 kWh 1378.96 kr.
Entertainment	620.43 kWh 1302.91 kr.	Entertainment	621.01 kWh 1231.91 kr.	Entertainment	618.39 kWh 1226.85 kr.
Bathroom	89.06 kWh 187.03 kr.	Bathroom	89.06 kWh 176.58 kr.	Bathroom	89.06 kWh 176.58 kr.
CoolingFreezing	403.48 kWh 847.21 kr.	CoolingFreezing	403.48 kWh 790.78 kr.	CoolingFreezing	346.39 kWh 667.16 kr.
Washing	1516.08 kWh 3183.76 kr.	Washing	1396.45 kWh 2783.63 kr.	Washing	1340.75 kWh 2481.55 kr.
<b>Total</b>	<b>3628.83 kWh</b> <b>7620.46 kr.</b>	<b>Total</b>	<b>3538.2 kWh</b> <b>7039.29 kr.</b>	<b>Total</b>	<b>3369.18 kWh</b> <b>6503.54 kr.</b>
4)		5)			
DERs	-1039.27 kWh -2077.77 kr.	DERs	-266.56 kWh -521.81 kr.		
Cooking	312.47 kWh 631.08 kr.	Cooking	328.97 kWh 657.91 kr.		
Lights	692.41 kWh 1381.25 kr.	Lights	693.13 kWh 1382.55 kr.		
Entertainment	619.13 kWh 1228.07 kr.	Entertainment	621.98 kWh 1233.49 kr.		
Bathroom	89.06 kWh 176.58 kr.	Bathroom	89.06 kWh 176.58 kr.		
CoolingFreezing	346.39 kWh 667.16 kr.	CoolingFreezing	346.39 kWh 667.16 kr.		
Washing	1339.3 kWh 2475.58 kr.	Washing	1420.72 kWh 2638.9 kr.		
<b>Total</b>	<b>2359.48 kWh</b> <b>4481.94 kr.</b>	<b>Total</b>	<b>3233.68 kWh</b> <b>6234.78 kr.</b>		

Figure 78: The results of each of the five setups for the case study of a house. The results are simulated values and retrieved from the reporting window of the application.



The effect of investing in a DER for the house has been examined in setup 4 and 5. The solar panel added in setup 4 is only 6 m<sup>2</sup> which is a rather small solar panel, but it still saves the house 2077.77 DKK per year. The wind turbine added in setup 5 is a small 400W turbine from *ProCure* which costs 12,000 DKK [108: ProCure]. The yearly savings using this turbine has been found to be 521.81 DKK using setup 5. These savings can be seen by looking at the DERs category in the above figure.

### 6.3.2. Case Study: Apartment

In the Apartment case study the same effects of the simulation can be seen as the benefits of upgrading to both fluctuating prices and smart grid enabled appliances lowers the total electric bill. As the apartment does not contains any of the obvious smart grid enabled appliances such as washing machines and electric heating units, the benefits of upgrading will be less significant than in the House case study. Below the results of the simulation for the five setups can be seen.

1)		2)		3)	
Bathroom	1.31 kWh 2.76 kr.	Bathroom	1.31 kWh 2.51 kr.	Bathroom	1.31 kWh 2.51 kr.
Entertainment	1010.69 kWh 2122.46 kr.	Entertainment	1010.69 kWh 2003.84 kr.	Entertainment	1010.69 kWh 2003.84 kr.
Cooking	163.46 kWh 343.27 kr.	Cooking	163.67 kWh 328.67 kr.	Cooking	160.72 kWh 322.56 kr.
Lights	42.03 kWh 88.27 kr.	Lights	42.03 kWh 83.64 kr.	Lights	42.05 kWh 83.68 kr.
Other	0 kWh 0 kr.	Other	2.5 kWh 5.03 kr.	Other	2.5 kWh 5.02 kr.
CoolingFreezing	219 kWh 459.9 kr.	CoolingFreezing	219 kWh 428.51 kr.	CoolingFreezing	219.05 kWh 421.5 kr.
<b>Total</b>	<b>1436.51 kWh</b> <b>3016.66 kr.</b>	<b>Total</b>	<b>1439.21 kWh</b> <b>2852.2 kr.</b>	<b>Total</b>	<b>1436.33 kWh</b> <b>2839.11 kr.</b>
4)		5)			
DERs	-1039.27 kWh -2077.77 kr.	DERs	-266.56 kWh -521.81 kr.		
Bathroom	1.31 kWh 2.51 kr.	Bathroom	1.31 kWh 2.51 kr.		
Entertainment	1010.69 kWh 2003.84 kr.	Entertainment	1010.69 kWh 2003.84 kr.		
Cooking	160.7 kWh 322.48 kr.	Cooking	165 kWh 331.63 kr.		
Lights	42.03 kWh 83.64 kr.	Lights	42.03 kWh 83.64 kr.		
Other	0 kWh 0 kr.	Other	1.5 kWh 3 kr.		
CoolingFreezing	219.05 kWh 421.5 kr.	CoolingFreezing	219.05 kWh 421.5 kr.		
<b>Total</b>	<b>394.52 kWh</b> <b>756.20 kr.</b>	<b>Total</b>	<b>1173.03 kWh</b> <b>2324.32 kr.</b>		

Figure 79: The results of each of the five setups for the case study of the apartment.



It can be seen that upgrading to fluctuating prices will lower the total yearly cost with 164 DKK and by adding a controller and smart grid enabling the existing appliances a mere saving of 13 DKK can be obtained. This is due to the fact that the apartment only contains the refrigerator which can be upgraded to a smart grid enabled unit and therefore the basis for saving money on moving the consumption time is minimal. The results of upgrading the household with DERs are the same as with the House case study, as it is the same units added to the apartment and the wind data used to calculate their energy production is the same due to the simulation period being the same.

## 6.4. Demonstration and Feedback

The final aspect of the testing process was to demonstrate the application to a company that works within the field of smart grids and home automation in order to receive useful feedback and verify that the application addresses real issues faced by the industry. The Danish company *GreenWave Reality* was chosen for this as they are geographically accessible, they have already created a commercial home automation system and they are working on creating controller units for homes in smart grid scenarios.

The demonstration was performed during a meeting on the 16th of August 2012 in GreenWave Reality's offices with sales engineer, Oliver Dufour, and the CFO, Peter Wilmar Christensen. The feedback received was very positive and the general opinion was that the application is very close to being a commercial product that can be used in real scenarios. They noted that the application could be highly useful for people selling solar panels and DERs in general in order to convince people that investing in the solar panels would be a good idea. This confirms that the application in fact fulfills the goal of targeting salesmen as described in the Requirements chapter (see 3.1.2 *The Advisor/Salesman*).

Besides the application's obvious use to measure energy consumption GreenWave Reality noted that the application already contains a lot of the information needed to measure the *flexibility* of a home (as described in 1.1.3.4 *Flexibility*) and that it with some extensions could provide such calculations. This would be highly useful for utility companies as the higher a house' flexibility is the more valuable a member it is to the smart grid. Therefore the utility companies could in the future let the consumer's energy prices vary based on his home's flexibility. In such a situation it would be highly useful to be able to simulate the flexibility of the home, both for the utility company and for the consumer. In relation to the flexibility they noted that it would be very useful to add heat pumps to the application as they provide a significant amount of flexibility by being able to heat the house when the smart grid requires higher consumption and turn off when the grid is loaded. Heat pumps were, as seen in 3.4 *Modeling Components*, also originally a planned component in the application, but did not make it into the prototype implementation due to the time restrictions.

Another component suggested by GreenWave Reality was the electric vehicle. It was not included in the current prototype as the current infrastructure in Denmark provided by BetterPlace does not let the charging stations connect to the home and therefore they will not be part of the scheduling in the home (as mentioned in 1.1.3.2.1 *Electric Vehicles (EVs)*). GreenWave expects that this will change in the future and multiple types of infrastructures will be available; some of which will be connected to the home and therefore be highly useful to the home when shifting the energy consumption to cheaper periods.

Lastly, the idea of integrating historical data of the consumer's usage patterns was brought up. This could be used to provide more precise simulations based on how the consumer uses energy today.

Overall the meeting with GreenWave Reality illustrated that the developed application provides functionalities and simulations that are very useful when addressing the issues faced in the industry and tools such as this can be used by a number of groups in the field of smart grids and smart homes. As such the application is highly relevant and further development should be encouraged.

## 6.5. Conclusion

In this chapter the testing philosophy of this thesis has been documented. The results of each of the testing strategies has been discussed and evaluated.

Unit testing was used to verify the code in the program and to test how the application handles certain types of input correctly. Several unit tests were created during implementation for each part of the program such that the code could be validated during implementation, which helped identifying if new features broke earlier implemented features. Unit testing was also used for testing code performance such as sorting and calculations.

Throughout the implementation of the program manual testing has been a crucial part of the process as the application contains a lot of elements which only manual testing can verify the results of. Both the graphical user interface and the simulation contains content which cannot be evaluated automatically and therefore require manual tests. Examples of features that have been manually tested are the performance of the simulations and of the charts, neither of which perform well under the current conditions. This thesis suggests an upgrade to a commercial and better performing chart package, to replace the current free charting package from Microsoft which is performing very poorly when working with large datasets.

The simulated and estimated consumptions of the components in the application were compared to external reference values in order to validate the accuracy of the implemented consumption algorithms. In Table 11 the estimated and simulated values can be seen along with the reference consumptions. It can be seen that only the Heating/Cooling/Freezing components differ more than what is reasonable and they would need to be improved in future development.

The application was demonstrated to GreenWave Reality in order to get feedback from a party which is involved in the smart grid and controller unit industry. Their feedback illustrated that there is a need for such a simulation tool in the industry and that they could see many uses for the application.

# 7. Conclusion

---

In this thesis the goal was to create a simulation tool for smart homes that can help shedding light on the effects of implementing smart grid technologies and initiatives in a home such as shifting the energy consumption based on fluctuating energy prices. This chapter contains the conclusion of thesis and will present it by evaluating the fulfillment of the thesis goals presented in section 1.3 and then sums up the work done in this thesis in an overall conclusion.

## 7.1. Conclusion on the Thesis Goals

Section 1.3 *Thesis Definition* described 14 thesis goals that define what the objectives of the thesis are. This section will examine each of the thesis goals and describe which sections of the report describe the fulfillment of the goals in order to verify that the thesis in fact covers what it was intended to.

- 1. Present the required background knowledge about the existing power grid, the Smart Grid, the power market, Virtual Power Plants (VPP) and Distributed Energy Resources (DER)**

The required background knowledge of the existing power grid was described in section 1.1.1 and the power market was described in 1.1.1.2. The smart grid and VPPs were presented in 1.1.2 and DERs were presented in 1.1.3.1.

- 2. Analyze the concepts of a smart meter and Home-Area-Network (HAN)**

An analysis of Home Area Networks and smart meters has been conducted in section 1.1.3 and in section 2.2. In these sections the thesis has gone through all the core concepts of HAN including the smart meter.

- 3. Examine the existing approaches for communication within the smart home**

In section 2.2.4 the approaches used for communication within smart homes were analyzed and a number of protocols were identified and compared.

- 4. Survey existing state-of-the-art controller units and projects for home automation**

Section 2.2.5 found that not many controller units currently exist for smart homes but three were found and compared. It was found that controller units do not all work in the same way and as such they cannot be ignored in a simulation tool, leading to the inclusion of controller units in this application.

- 5. Survey existing state-of-the-art simulation solutions**

Section 2.1 provides a presentation of four existing smart home simulation tools. The tools have been compared and necessary features not available in the current tools were identified.

- 6. Present the target audience of the application**

The requirements chapter identifies the target audience in section 3.1 based on the identified focus areas of the solution. Three main target groups were identified: The consumer (section 3.1.1), the advisor/salesmen (section 3.1.2) and the educator/educatee (Section 3.1.3).

### **7. Identify aspects of a private home that are relevant for the simulation**

The components in a home that are relevant to a smart home simulation tool were identified in the requirements chapter in section 3.4 and grouped in categories for use in the application.

### **8. Identify functional and non-functional requirements for the simulation tool**

The most important non-functional requirements of the application such as usability, customizability and extensibility were presented in 3.6 and section 3.7 presented the functional requirements of the application. These requirements led to use case descriptions presented in section 3.8.

### **9. Find an appropriate design architecture for the system**

This goal has been covered in section 4.1.1 where the choice of creating a desktop application with no server connection is presented and the Prism library is used as a foundation for best practice implementation.

### **10. Design the simulation tool with extensibility in mind**

To allow the application to be extended easily with new feature it was designed using a modular approach. In section 4.1.3 the decisions regarding the module structure are discussed and in section 4.1.4 the functionality of the application was divided into logical parts. To help the implementation follow the design guidelines and to increase extensibility, interfaces and abstract implementations were used extensively and the use of these is described in section 4.1.5.

### **11. Adhere to chosen standards of user interface design**

In section 3.6.5 relevant sections of the usability standard *ISO 9241-171* is presented along with a number of other usability advices from other sources. The adherence to these standards is presented in section 5.8.

### **12. Implement a prototype of the designed simulation tool as a proof of concept**

The designed tool was implemented as a prototype and the implementation of the tool is described in chapter 5 along with a presentation of the features.

### **13. Use existing design patterns to guide a best practice implementation of the tool**

The application uses a number of existing design patterns such as Model-View-ViewModel and Command pattern to guide the implementation of the tool. The use of these design patterns is described in section 4.1.2.

### **14. Create case studies that show how the application can be used to discover the effects of a smart grid on a private home**

In section 6.3 case studies were performed that show how a real private home would be affected by using fluctuating prices and smart grid enabled components as described by the simulations of the created tool. The case study of a regular house revealed a potential 1100 DKK saving per year.

## **7.2. Overall Conclusion**

As described in the vision of the thesis (see *1.2 Vision*) the Danish workgroup, *Smart Grid Netværket*, suggested that the knowledge of the smart grid should be increased at consumer level in order to increase the acceptance of the smart grid. This thesis helps reach this goal by allowing consumers to model their own home and see how their energy consumption would change if the appliances were smart grid enabled and the house used fluctuating energy prices rather than flat rate prices. Additionally, a number of other areas have been identified in which the application would prove valuable. This includes salesmen that wish to illustrate the financial benefits of their product (see *3.1.2 The Advisor/Salesman* and *6.4 Demonstration*

*and Feedback*) and utility companies that wish to measure the flexibility of the homes connected to the grid, as identified during the meeting with GreenWave Reality (see *6.4 Demonstration and Feedback* and *1.1.3.4 Flexibility*). All in all the application fulfills multiple target groups' requirements in the area of smart grids that are all currently unmet.

During the analysis of the thesis it was found that a number of controller units for smart homes already exist and that they can work in different ways (see *2.2.5 Controllers for Home Automation*). As this has an impact on the simulation results, the application has been created such that it can load multiple different controller units and the user can choose which one to use. The analysis also found that a wide range of communication protocols could be used in a smart home. As these do not necessarily cooperate, the application enables the consumer to choose a communication protocol for each of their components and future developers can implement actions in the application based on these choices, e.g. ensuring that all components use the same communication protocol.

An analysis of the state-of-the-art simulation tools of smart homes showed that very little work has been done in the area and that the existing tools are mostly non-graphical and therefore not very easy to use. Additionally they do not provide much opportunity to model specific homes and the incorporation of real energy and weather data is non-existent. Lastly the graphical analysis and comparison of different simulations were not present in any significant extent in any of the tools. These missing features provided the foundation for the simulation tool developed in this thesis.

The application developed in this thesis is a prototype and therefore is not developed to a state where it can be released to the end users. The program, however, still shows how a number of useful features can be used to provide consumers with detailed information about their potential energy consumption and how the application can support future extensions in a simple way. First of all the application offers more features than any of the other competing smart home simulation tools. It also has more focus on the end user by focusing on high usability and a personalized experience by allowing the user to model their own house. The application uses real input data in the simulations such as energy prices and weather information from different data source which makes the simulations more realistic. For the developer the solution is very easy to extend with new features as it has been developed using a module structure where functionality has been divided into separate, loosely coupled, logical modules.

There are some shortcomings of the application that could be improved in future development. Currently the simulations do not use much information about the physical house and its structure and such information could be used to enable more accurate simulations. Information about room structure and wall insulation could e.g. be used to provide better results when simulating how heating elements are consuming energy. There are also some performance issues when running simulations and displaying the results of the simulations. These are obviously important issues, but are inevitable: Because the simulations consist of heavy calculations it would require a centralized server solution to perform the simulations faster, and to improve the slow performance when displaying simulation results as charts would require a better commercial charting tool that can cope with the large amount of data.

As mentioned in the three sections, *1.1.3.3 - The Effect of the Current Energy Pricings on Smart Homes*, *2.2.2.1 - Legislative Restrictions on Privacy of Smart Meter Data* and *2.2.5.5 - Closed Standards for Appliances*, there are some technical and legal barriers surrounding the smart grid and thereby the smart

homes that need to be solved before the application can be finished and be ready to provide completely realistic results. This includes the lack of a chosen standard for how elements should communicate in a home which effects the use of controller units; and the issue of how the fluctuating prices (specifically the tariffs) in Denmark will be structured (see more in *1.1.3.3 The Effect of the Current Energy Pricings on Smart Homes*) which effects the energy prices and ultimately whether there is any money to save by moving the energy consumption. This tool has been developed assuming that these barriers are not present.

As identified in this thesis the application has multiple uses. One of them is to answer the question: “How will the smart grid affect my energy bill?”. This question can be answered by creating a model of the home and running a simulation using no smart grid enabled components and a flat rate power model and running a simulation using smart grid enabled components and fluctuating energy prices. This has been done in the case studies performed in section *6.3 Case Studies* and the results show that in one of the cases the consumer can save 581 DKK per year by changing from flat rate to fluctuating energy prices and a total of 1117 DKK per year if the components are also smart grid enabled. A report made by The Danish Competition Authority states that the main motivation for consumers when considering changing power suppliers is the energy prices, but that the savings have to be around 750 DKK per year before 30% of the consumers would consider changing. According to the report such savings are difficult to obtain as most Danish households have an energy consumption of around 4000-4500 kWh per year [6: Konkurrencestyrelsen] which is around 8500-10000 DKK. The results of the case studies in this thesis shows that such savings could be obtained by switching to smart grid enabled components and using fluctuating energy prices, which indicates that people who would be made aware of this could be interested in switching to smart grid enabled appliances.

## 8. Future Work

---

As mentioned in the Requirements chapter (see 3.6.2 *Extensibility*) the goal of this thesis was to create the application such that it would be simple and easy to develop further later on. Therefore one of the main focus areas of the application was extensibility. The application was therefore designed such that the functionality was split into logical groups of components which were loosely coupled with as simple communication between them as possible. In the current implementation of the prototype some extensions have already been prepared for and the application has been developed such that they can easily be incorporated.

One of the most substantial preparations made is the automatic loading of third party assemblies to be used in the application. This is used to allow external parties to extend the program with additional content, such as new controller units or data providers for weather or power data. The program is created to go through folders and find the DLL assemblies that implement the given interface provided by the *Common* projects of the application. This way e.g. controller manufacturers can implement and try out their own scheduling methods before trying them with their hardware or they can use it to make users aware of their controller unit and let them try it out in their home in the simulation.

Another prepared extension is the smart meter module. It currently does not contain much implementation other than interfaces to be used and a manager to load and control the added smart meters, but it is intended to handle integration to real smart meters and thereby give access to realistic and historic data. With this data the application could provide an extra level for the simulations when scheduling the components by taking the user's energy consumption patterns into consideration.

Besides the mentioned prepared extensions there is some functionality that was planned in the requirements and/or the design of the application, but did not make it into the prototype due to the limited time span. The features are presented below:

- A list of **improving component upgrades** could be a part of the reporting. The application could suggest which components in the user's setup could be replaced in order to lower their consumption and save them money in the long run. The application could compare the total consumption of the modeled component with the total consumption of the other components of the same type available within the program. With this list the user could get an idea of which elements they could upgrade. This could be made as a separate module which could access the components in the model through the container and also get all the available components through the modeling manager.
- **Validation checks** of the current content on the canvas could be implemented in which the application looks at which components are added to the canvas and then evaluates whether there is a reasonable amount of consumption in each of the defined categories. This could help inform the user if they are missing some normally occurring elements in their setup which would skew the simulation results. This could be made as a part of the summary module as it could easily check it's

already contained data against a defined scheme of how the share of consumption for each category normally is.

- **Integration with the GridLab-D** simulator could give the user some advanced options in the simulation of the home such as isolation of the home. By allowing the user to export their scenarios to a GLM file (GridLab-D's file format), they could be able to perform simulations with their setup with a simulator which is looking at different parameters than this thesis. (Read more about GridLab-D in section 2.1.4 *GridLab-D*). GridLab-D can be run as a command line tool and can therefore also be called directly from C# with the static *Process* class.
- The **types of modeling components** should be extended with the types defined in the requirement chapter that are not in this version of the prototype. This includes the heating pump and water heater types as they both consume a considerable amount of energy and could be scheduled to use their energy more efficiently.
- **A number of use cases** that were listed as *nice to have* or *cosmetic* in the Requirements chapter (see 3.8 *Behavioral Requirements (Use Cases)*) were not implemented in this prototype and are therefore likely candidates for implementation if extending the prototype later on. This list of discarded use cases can be seen below:
  - Print report of simulation (Use case #22)
  - Duplicate a scenario (Use case #26)
  - Export Scenario from a project (Use case #30)
  - Import Scenario to a project (Use case #31)
  - Print Model (Use case #32)
  - Recreate model from simulations (Use case #33)
  - Popup hints displayed (Use case #34)
  - Hide popup hint (Use case #35)
  - Select specific historic weather forecast (Use case #36)
  - Save chart as image (Use case #37)

A lot of interesting features and functionalities were identified during the development that would be obvious to implement as it would give the user a better experience when using the application. A list of the extensions can be seen below, but the extent of each of the extensions has not been taken into consideration as it would require a more thorough analysis.

- Adding functionality for the user to create **dependencies between the components** would make the simulations more realistic as they would be able to schedule/simulate the running times of the components more accurately. An example would be if the user adds both a TV and a PlayStation 3, then the PlayStation should generally not be turned on when the TV is not turned on. This could be resolved by either introducing a "depends on"-property list in the components or by just creating wrapper components which have increased property values that match the merged components' summed up values, such as the case might be for speakers, a PC screen and the PC itself combined into one component.
- Allowing the user to set **a list of "on periods"** on components so that the user can set that a component can be turned on in multiple intervals such as morning and evening, but not in between. In the current implementation only one "on period" interval can be set.



- Rethinking the modeling part of the application could allow the user to model their house exactly as it is, by defining a **floor plan of the house** and then placing the components in the correct locations. This would allow both the user and the application to simulate the consumption in the house more accurately, as the simulation could take the house's interior and component arrangement into consideration for e.g. heating calculations. This however, has to be considered very carefully as it can easily become too complex to use which would discourage many users from using it.
- While using the application the user could be guided through the program by a **tutorial** which shows the user how to model their home and suggests the "next moves" for the user. It would introduce the user to the functionalities of the program and make them aware of the options in the application. This extension would improve the usability of the program drastically as the user would be introduced to the program while using it rather than relying on the user to read a guide first or trying with trial and error (see 3.6.5 *Usability*).
- By integrating the user's **historic consumption information** the program could give the user more accurate simulations as the simulation could take the user's usage patterns into consideration. This extension requires that such information has been stored for a longer period of time prior to the use of the data. Many existing in-home displays provide such data and could possibly be integrated with the application [109: Tendril; 110: GreenWave Reality; 111: HAI Smartgrid Solutions]. This extension was suggested by GreenWave Reality at a meeting with them in August 2012 (See 6.4 *Demonstration and Feedback*).
- Converting the application so that it runs as a **web based application** would make the application more accessible for the user compared to installing it as a desktop application. This would require a lot of reprogramming of some of the parts, but if Silverlight is chosen as the technology to use, a lot of the code can be reused without changes as Silverlight is basically just a web-version subset of WPF. Converting the application to a web based solution also enables the solution to include some features such as a central database and more computational power when simulating. Such a solution could be made with either HTML5 or Silverlight as front end and a backend consisting of services to integrate data from other sources, control created models etc. which would all be placed in the Cloud.
- The **default controller** can still be improved using information that is already available to it. The controller does e.g. not handle the situation that the prices for selling and buying energy could differ. If the prices are different the controller could determine if it was profitable to sell the generated power from the DERs rather than using it internally in the house.
- The simulation tool uses many of the same calculations needed by a controller unit. This makes it reasonable to consider using parts of the simulation tool to provide functionality that can **control real components** and thereby be used as a controller unit in a home. This way it can also work as an in-home display (IHD) that shows the consumption of the home using the same reporting tool already available in the application.

The application is developed as a prototype and therefore a lot of further development can and will be expected before the application is in a state where it can be released. In this chapter a list of some of the planned extensions has been introduced together with a list of extensions which was not considered while planning, but became clear during the development process. The majority of these extensions requires

quite a lot of programming and would definitely require both planning and designing before being implemented. Some of them are also very comprehensive and would require a change to the base structure of the program in order to work properly, such as converting the application to a web based solution.

When extending the application it is important to consider that if one makes the application too detailed, containing too many advanced features, it could easily make it difficult for the user to perform even simple tasks and thereby discourage the user from using the application. As long as the *consumer* is part of the target audience the application must be kept at a complexity in which consumers (i.e. nonprofessionals) are capable and willing to operate. If the fundamental workflows in the application such as modeling and simulating are to be increased in complexity the developers should first consider whether the target audience, and thereby focus, of the application should be changed.

# References

---

1. Microsoft. *Microsoft Hohm Beta*. 2012 [cited 2012 6/6]; Available from: <http://www.microsoft-hohm.com/>.
2. Belkin. *Conserve Energy-Saving Products*. 2012 [cited 2012 6/6]; Available from: <http://www.belkin.com/conserve/products/>.
3. Google. *Google PowerMeter: A Google.org Project* 2011 [cited 2012 6/6]; Available from: <http://www.google.com/powermeter/about/>.
4. GreenWave Reality. *Solutions*. 2012 [cited 2012 6/6]; Available from: <http://www.greenwavereality.com/solutions/>.
5. Klima- Energi- og Bygningsministeriet, *Smart Grid Netværkets Arbejde - Sammenfatning og anbefalinger*, 2011.
6. Konkurrencestyrelsen, *Konkurrencen på detailmarkedet for el*. 2009.
7. Energinet.dk. *Elproduktion og -forbrug*. 2012 29/4 [cited 2012 13/2]; Available from: <http://energinet.dk/DA/KLIMA-OG-MILJOE/Elsektorens-miljoepaavirkninger/Elproduktion-i-Danmark/Sider/Elproduktion-og-forbrug.aspx>.
8. Klima- Energi- og Bygningsministeriet. *Smart Grid Netværkets Anbefalinger*. 2011 [cited 2012 9/9]; Available from: [http://www.kemin.dk/da-DK/KlimaogEnergipolitik/danmark/energiforsyning\\_og\\_effektivitet/smartgrid/smartgrid\\_anbefal/Sider/Forside.aspx](http://www.kemin.dk/da-DK/KlimaogEnergipolitik/danmark/energiforsyning_og_effektivitet/smartgrid/smartgrid_anbefal/Sider/Forside.aspx).
9. Schaldemose, B. *Teknologisk Institut med til at skabe fremtidens elnet*. n.d. [cited 2012 13/1]; Available from: <http://www.teknologisk.dk/31044>.
10. Ricci, A., et al. *Power-Grid Load Balancing by Using Smart Home Appliances*. in *Consumer Electronics, 2008. ICCE 2008. Digest of Technical Papers. International Conference on*. 2008.
11. Haas, K. *Peaking Power Plants*. 2009 4/11 [cited 2012 13/2]; Available from: <http://www.energyandcapital.com/articles/peaker-plants-bridge-the-gap/991>.
12. Illinois Environmental Protection Agency. *Peaker Power Plant Fact Sheet*. 2011 [cited 2012 13/2]; Available from: <http://www.epa.state.il.us/air/fact-sheets/peaker-power-plant.html>.
13. Electric Power Research Institute. *EPRI Smart Grid Resource Center*. n.d. [cited 2012 14/6]; Available from: <http://www.smartgrid.epri.com/Demo.aspx>.
14. Xu, Z. and Q. Wu, *Review on Integration of Distributed Energy Resources (DER) in Power Systems*, 2011, Centre for Electric Technology, the Technical University of Denmark.

15. Oxford Dictionaries. *Smarth Home*. Oxford Dictionaries n.d.; Available from: <http://oxforddictionaries.com/definition/english/smart%2Bhome>.
16. Kok, K., et al., *Field-Testing Smart Houses for a Smart Grid*. 21st International Conference on Electricity Distribution, 2011.
17. Energy Saving Trust. *Wind Turbines*. 2012 [cited 2012 13/2]; Available from: <http://www.energysavingtrust.org.uk/Generate-your-own-energy/Wind-turbines#2>.
18. Wesoff, E. *Update: Solar Junction Breaking CPV Efficiency Records, Raising \$30M*. 2011 15/4 [cited 2012 13/2]; Available from: <http://www.greentechmedia.com/articles/read/solar-junction-setting-new-cpv-efficiency-records/>.
19. BBC News. *It's Payback Time for Home Generation*. 2010 22/6 [cited 2012 13/2]; Available from: <http://news.bbc.co.uk/2/hi/science/nature/8751862.stm>.
20. BetterPlace. *FAQ - Ladestander*. n.d. [cited 2012 23/7]; Available from: <http://danmark.betterplace.com/faq/ladestander/>.
21. Flaim, T. and R. Levy, *Dynamic Pricing in a Smart Grid World*, 2010, NARUC Webinar.
22. Control4. *Introducing the Control4 EMS 100*. 2012 [cited 2012 24/2]; Available from: <http://control4.com/energy/products/>.
23. Dansk Energi. *Grøn Omstilling Kræver Lav Afgift og Dynamiske Tariffer på Strøm*. 2010 7. juli 2010 [cited 2012 6/6]; Available from: [http://www.danskeenergi.dk/Aktuelt/Arkiv/2010/Juli/10\\_07\\_07A.aspx](http://www.danskeenergi.dk/Aktuelt/Arkiv/2010/Juli/10_07_07A.aspx).
24. Det Energipolitiske Udvalg, *Redegørelse om Muligheder for og Virkninger af Ændrede Afgifter på Elektricitet med Særlig Henblik på Bedre Integration af Vedvarende Energi (Dynamiske Afgifter)*, 2010, Skatteministeriet.
25. Go Energi. *Forstå Din Elregning*. 2012 15/2 [cited 2012 6/6]; Available from: <http://www.goenergi.dk/forbruger/alt-om-energiforbrug/elforbrug/forstaa-din-elregning/index.html>.
26. Elpristavlen. *Mulige Elprodukter*. n.d. [cited 2012 6/6]; Available from: <http://www.elpristavlen.dk/Elpristavlen/Soegeresultat.aspx?kwh=4000&postnr=2800&netcompany=DONGnet&customer-group=private&ratetype=FlatRate>.
27. Software Development Methodologies. *Software Development Methodologies*. n.d. [cited 2012 6/6]; Available from: <http://softwaredevelopmentmethodologies.org/>.
28. Schwaber, K. and J. Sutherland, *The Scrum Guide*, 2011, Scrum.org.
29. Conte, G., et al. *A Simulation Environment for the Analysis of Home Automation Systems*. in *The 15th Mediterranean Conference on Control & Automation*. 2007. Athens, Greece.
30. EnergyPlus, *Getting Started with EnergyPlus*, in *Basic Concepts Manual - Essential Information You Need about Running EnergyPlus* 2011, US Department of Energy.

31. PowerMatcher Smartgrid Technology, *PowerMatcher Simulation Tool* 2010, Energy Research Centre of the Netherlands.
32. GridLAB-D. *GridLab-D Simulation Software*. 2011 [cited 2012 22/2]; Available from: <http://www.gridlabd.org/>.
33. PowerMatcher Smartgrid Technology. *Basic Structure and Agent Roles*. n.d. [cited 2012 6/6]; Available from: <http://www.powermatcher.net/get-technical/powermatcher-technology/basic-structure-and-agent-roles/>.
34. GridLAB-D. *Getting Started Using GridLAB-D*. 2011 12/11 [cited 2012 22/2]; Available from: [http://sourceforge.net/apps/mediawiki/gridlab-d/index.php?title=Getting\\_Started\\_Using\\_GridLAB-D](http://sourceforge.net/apps/mediawiki/gridlab-d/index.php?title=Getting_Started_Using_GridLAB-D).
35. Silver Spring Networks. *Silver Spring Networks*. 2012 [cited 2012 6/6]; Available from: <http://www.silverspringnet.com/>.
36. Farhangi, H., *The Path of the Smart Grid*. IEEE Power & Energy Magazine, 2010.
37. Trilliant. *Open, Standard Smart Grid Communications Networks*. n.d. [cited 2012 6/6]; Available from: <http://www.trilliantinc.com/mobile-home>.
38. International Electrotechnical Commission, *IEC 62056-21: Electricity Metering - Data Exchange for Meter Reading, Tariff and Load Control - Part 21: Direct Local Data Exchange* 2002.
39. Kamstrup. *Elektricitet*. n.d. [cited 2102 6/6]; Available from: <http://www.kamstrup.dk/111/Elektricitet>.
40. Echelon. *Smart Metering*. 2012 [cited 2012 6/6]; Available from: <http://www.echelon.com/applications/smart-metering/>.
41. Trilliant. *Smart Grid Communications Network*. 2012 [cited 2012 6/6]; Available from: <http://trilliantinc.com/>.
42. IEEE Spectrum. *Privacy on the Smart Grid*. 2010 6/10 [cited 2012 6/6]; Available from: <http://spectrum.ieee.org/energy/the-smarter-grid/privacy-on-the-smart-grid>.
43. NIST. *NIST Finalizes Initial Set of Smart Grid Cyber Security Guidelines*. 2010 2/9 [cited 2012 6/6]; Available from: [http://www.nist.gov/public\\_affairs/releases/nist-finalizes-initial-set-of-smart-grid-cyber-security-guidelines.cfm](http://www.nist.gov/public_affairs/releases/nist-finalizes-initial-set-of-smart-grid-cyber-security-guidelines.cfm).
44. Federal Office for Information Security, *Protection Profile for the Gateway of a Smart Metering System*, 2011.
45. Huq, M.Z. and S. Islam. *Home Area Network Technology Assessment for Demand Response in Smart Grid Environment*. in *Universities Power Engineering Conference (AUPEC), 2010 20th Australasian*. 2010.
46. California Energy Commission and Charles River Associates, *Statewide Pricing Pilot: Summer 2003 Impact Analysis*, 2004, Charles River Associates.

47. Michaels, H. and K. Donnelly, *Architecting the Smart Grid for Energy Efficiency*. ACEEE Summer Study on Energy Efficiency in Buildings, 2010(11): p. 163-173.
48. EnergyHub. *EnergyHub*. n.d. [cited 2012 6/6]; Available from: <http://www.energyhub.com/>.
49. SmartHomeUSA.com. *Commstar Home Automation Controller*. 2012 [cited 2012 24/2]; Available from: <http://www.smarthomeusa.com/ShopByManufacturer/JDS-Technologies/Item/CS308-IP/>.
50. SmartHomeUSA.com. *HAI Omni Home Control Systems*. 2012 [cited 2012 24/2]; Available from: <http://www.smarthomeusa.com/Shop/pc-home/HAI/HAI-Omni-Home-Control-Systems/>.
51. SmartHome.com. *SmartLinc - INSTEON Central Controller*. 2012 [cited 2012 24/2]; Available from: <http://www.smarthome.com/2412N/SmartLinc-INSTEON-Central-Controller/p.aspx>.
52. DONG Energy. *eFlex*. 2012 [cited 2012 8/9]; Available from: <http://www.dongenergy-distribution.dk/da/Privat/Kundeservice/Elkunde/Testkunder%20%E2%80%93%20e-Flex/Pages/testkundere-Flex.aspx?override=web>.
53. DTU Elektro. *iPower*. n.d. [cited 2012 8/9]; Available from: <http://www.ipower-net.dk/>.
54. EcoGrid. *EcoGrid EU*. n.d. [cited 2012 8/9]; Available from: <http://www.eu-ecogrid.net/>.
55. Tendril. *Tendril Connect*. 2012 [cited 2012 24/2]; Available from: <http://www.tendrilinc.com/platform/connect/>.
56. Barki, H. and J. Hartwick, *User Participation and User Involvement in Information System Development*. Emerging Technologies and Applications Track, IEEE Computer Society Press, 1991. **4**: p. 487-492.
57. McGilla, T. and J. Klobas, *User developed application success: sources and effects of involvement*. Behaviour & Information Technology, 2008. **27**(5): p. 407–422.
58. eMeter Strategic Consulting, *PowerCentsDC Program Final Report*, 2010.
59. GreenResearch, *Consumer Smart Grid Insights 2011*, 2011.
60. Capion, K.E., *Prisfleksibelt Elforbrug i Kølesystemer*, 2008, Technical University of Denmark.
61. Elmodelbolig. *Elmodelbolig Statistik*. n.d. [cited 2012 23/7]; Available from: <http://elmodelbolig.energiraadgiver.com/>.
62. Elsparefonden. *Selvtjek Bolig*. n.d. [cited 2012 23/7]; Available from: <http://www.selvtjekbolig.sparel.dk/>.
63. Go Energi. *Hvor bruger du mest strøm i dit hjem?* 2012 [cited 2012 23/7]; Available from: <http://www.goenergi.dk/forbruger/alt-om-energiforbrug/elforbrug/hvor-bruger-du-mest-stroem>.
64. Elmodelbolig. *Elmodelbolig Statistik - Nye apparaters elforbrug (2010)*. n.d. [cited 2012 23/7]; Available from: <http://elmodelbolig.energiraadgiver.com/NEM/NyeApp>.
65. ISO, *Ergonomics of human-system interaction*, in *Part 171: Guidance on software accessibility* 2008.

66. Tognazzini, B. *First Principles of Interaction Design*. n.d. [cited 2012 23/7]; Available from: <http://www.asktog.com/basics/firstPrinciples.html>.
67. Nielsen, J. *Ten Usability Heuristics*. 2005 [cited 2012 23/7]; Available from: [http://www.useit.com/papers/heuristic/heuristic\\_list.html](http://www.useit.com/papers/heuristic/heuristic_list.html).
68. Nielsen Norman Group. *Strategies to enhance the user experience*. n.d. [cited 2012 23/7]; Available from: <http://www.nngroup.com/>.
69. Spool, J., *Drag 'n Drop is Invisible To Users*, in *User Interface Engineering*, 2005.
70. Wang, M., *One Design Fits All? - Usability Check for Ribbon Interface*, in [www.infragistics.com](http://www.infragistics.com) 2011.
71. Hedges, G., *Behind the Scenes: Ribbon UIs – Love Them or Hate Them?*, in [creo.ptc.com](http://creo.ptc.com) 2011.
72. Garofalo, R., *Building Enterprise Applications with Windows® Presentation Foundation and the Model View ViewModel Pattern* 2011, Sebastopol, CA: O'Reilly Media, Inc.
73. Gossman, J., *100 Model/View/ViewModels of Mt. Fuji*, in [blogs.msdn.com](http://blogs.msdn.com) 2005.
74. Microsoft. *Commanding Overview*. n.d. [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/ms752308.aspx>.
75. Smith, J. *WPF Apps With The Model-View-ViewModel Design Pattern*. 2009 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/magazine/dd419663.aspx>.
76. McCarter, R. *Problems and Solutions with Model-View-ViewModel*. 2010 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/magazine/ff798279.aspx>.
77. Microsoft. *Prism 4.1 - Developer's Guide to Microsoft Prism*. 2012 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/gg406140.aspx>.
78. Microsoft. *Desktop Development*. 2010 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/ff921343.aspx>.
79. Microsoft. *4: Modular Application Development*. 2012 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/gg405479%28v=pandp.40%29.aspx>.
80. Microsoft. *2: Initializing Prism Applications*. 2012 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/gg430868%28v=pandp.40%29.aspx>.
81. Microsoft. *9: Communicating Between Loosely Coupled Components*. 2012 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/ff921122%28v=pandp.40%29.aspx>.
82. Microsoft. *5: Implementing the MVVM Pattern*. 2012 [cited 2012 23/7]; Available from: <http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx>.
83. Kava, A., *MEF vs. Unity in composite application (Prism)*, 2011.
84. Microsoft. *Prism*. 2012 [cited 2012 02/08]; Available from: <http://msdn.microsoft.com/en-us/library/ff648465.aspx>.

85. Microsoft. *Microsoft Ribbon for WPF October 2010*. 2010 [cited 2012 02/08]; Available from: <http://www.microsoft.com/en-us/download/details.aspx?id=11877>.
86. Microsoft. *WPF Toolkit - February 2010 Release*. 2010 06/03 [cited 2012 02/08]; Available from: <http://wpf.codeplex.com/releases/view/40535>.
87. Eberhardt, C., *WPF Charting Performance Comparisons (the Battle Continues)*, in [www.scottlogic.co.uk](http://www.scottlogic.co.uk) 2012.
88. Aim Software. *Mahapps.Metro Documentation*. 2012 [cited 2012 21/8]; Available from: <http://mahapps.com/MahApps.Metro/>.
89. Kaluza, M. *WPF "Drag and Drop" – Doing it the MVVM Way*. 2010 [cited 2012 21/8]; Available from: <http://sharpfellows.com/post/WPF-Drag-and-Drop-e28093-doing-it-the-MVVM-way.aspx>.
90. Microsoft. *SqlMetal.exe (Code Generation Tool)*. n.d. [cited 2012 21/8]; Available from: <http://msdn.microsoft.com/en-us/library/bb386987.aspx>.
91. Mourier, S. *Html Agility Pack*. 2012 [cited 2012 21/8]; Available from: <http://htmlagilitypack.codeplex.com/>.
92. Microsoft. *Open XML Developer*. 2011 [cited 2012 21/8]; Available from: <http://openxmldeveloper.org/>.
93. Wind Power Program. 14. *Wind Turbine Power Output Variation With Steady Wind Speed*. n.d. [cited 2012 21/8]; Available from: [http://www.wind-power-program.com/turbine\\_characteristics.htm](http://www.wind-power-program.com/turbine_characteristics.htm).
94. Friis, P., *Energien og Effekten i Vinden og i Vindmøllen*, 2011, Risø DTU.
95. Wind Power Program. 17. *Maximum Wind Turbine Efficiency - the Betz Limit*. n.d. [cited 2012 21/8]; Available from: <http://www.wind-power-program.com/betz.htm>.
96. ProCure. *ProCure A/S*. n.d. [cited 2012 21/8]; Available from: <http://www.procure.dk/moeller.htm>.
97. Kidwind Science Snack. *Understanding Coefficient of Power (Cp) and Betz Limit*. n.d. [cited 2012 21/8]; Available from: [http://learn.kidwind.org/sites/default/files/betz\\_limit\\_0.pdf](http://learn.kidwind.org/sites/default/files/betz_limit_0.pdf).
98. Danmarks Meteorologiske Institut. *Vind i Danmark*. n.d. [cited 2012 21/8]; Available from: [http://www.dmi.dk/dmi/index/klima/klimaet\\_indtil\\_nu/vind\\_og\\_vandstand\\_i\\_danmark.htm](http://www.dmi.dk/dmi/index/klima/klimaet_indtil_nu/vind_og_vandstand_i_danmark.htm).
99. Moll, E. *How to Calculate Solar Cell Output*. n.d. [cited 2012 8/9]; Available from: [http://www.ehow.com/how\\_7827904\\_calculate-solar-cell-output.html](http://www.ehow.com/how_7827904_calculate-solar-cell-output.html).
100. RenSMART. *Solar PV Comparison Table*. n.d. [cited 2012 21/8]; Available from: <http://www.rensmart.com/Products/SolarPV>.
101. Solar Choice. *Optimizing Solar Panel System Efficiency through Inverter Sizing*. 2011 [cited 2012 21/8]; Available from: <http://www.solarchoice.net.au/blog/optimizing-solar-panel-system-efficiency-through-inverter-sizing/>.



102. Morrill, J. *Silverlight Charts: Binding multiple Series*. 2008 [cited 2012 21/8]; Available from: <http://jmorrell.hjtcentral.com/Home/tabid/428/EntryId/351/Silverlight-Charts-Binding-multiple-Series.aspx>.
103. Software Testing Fundamentals. *Black Box Testing*. 2012 [cited 2012 7/9]; Available from: <http://softwaretestingfundamentals.com/black-box-testing/>.
104. Software Testing Fundamentals. *White Box Testing*. 2012 [cited 2012 7/9]; Available from: <http://softwaretestingfundamentals.com/white-box-testing/>.
105. Natlex. *Natlex*. n.d. [cited 2012 3/9]; Available from: <http://www.natlex.dk/elenergi.html>.
106. JRC European Commission. *Photovoltaic Geographical Information System*. n.d. [cited 2012 8/9]; Available from: <http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php>.
107. Renewable Energy UK. *Calculate KWh Generated By Wind Turbine*. n.d. [cited 2012 3/9]; Available from: <http://www.reuk.co.uk/Calculate-kWh-Generated-by-Wind-Turbine.htm>.
108. ProCure, *WG400 for Batteriladning eller Net-tilslutning*, 2012.
109. Tendril. *Insight - In Home Energy Display*. 2012 [cited 2012 8/9]; Available from: <http://www.tendrilinc.com/energy-providers/product/insight/>.
110. GreenWave Reality. *Energy Management*. n.d. [cited 2012 8/9]; Available from: <http://www.greenwavereality.com/solutions/energymgmt/>.
111. HAI Smartgrid Solutions. *In Home Display*. 2012 [cited 2012 8/9]; Available from: <http://smartgrid.homeauto.com/SmartGrid/IHD.asp>.



## Appendix A: Glossary

**SG** = Smart Grid

**DER** = Distributed Energy Resources

**VPP** = Virtual Power Plant

**WAN** = Wide Area Network

**NAN** = Neighborhood Area Network

**HAN** = Home Area Network

**DLMS** = Device Language Message Specification

**COSEM** = Companion Specification for Energy Metering

**HEMS** = Home Energy Management System

**EMS** = Energy Management System

**AMR** = Automated Metered Reading

**AMM / AMI** = Automatic Meter Management / Advance Metering Infrastructure (Extends **AMR**)

**IHD** = In Home Display (Shows smart grid data)

**TOU** = Time of Use

**EV** = Electric Vehicle

**WPF** = Windows Presentation Foundation

**IoC** = Inversion of Control

**DI** = Dependency Injection

**MVVM** = Model-View-ViewModel

**GUI** = Graphical User Interface

**KWh** = Kilo Watt hour

**XSD** = XML Schema Definition

**Scenario** = Scenario is the application's description of a specific configuration of a household

**Component** = Component is the application's description of a specific energy consuming device

**Controller Unit** = A unit in the home that controls energy consumption of all the devices in the home.

**Ribbon** = A Microsoft developed navigation menu

**Charting Toolkit = MSChart = Data.Visualization.Chart** = A Microsoft developed charting toolkit

## Appendix B: HAN Communication Protocols

### Ethernet (Wired)

The Ethernet technology is built on the IEEE 802.3 Ethernet standard (IEEE Standards Association, 2008) and is the commonly used technology for connecting to the internet. It is a well-known, proven and quite mature technology that a lot of businesses and private homes are already using. An Ethernet solution requires installation or use of for example RJ45 network cables and RJ45 Network interfaces on each of the units connected to the Home Area Network and the cabling will require the appliances to be stationary as they always need a plugged in cable. In order to use this technology in a HAN the appliances need to integrate a web based interface since this technology uses web based protocols to communicate between appliances and controllers.

#### Advantages

- Mature and proven technology
- Reliable and allows multi physical technologies (copper, optical fiber)
- Adequate data rates for the smart grid requirements
- Well-developed security mechanisms

#### Disadvantages

- Often requires that the consumer installs new cables to each appliance
- Appliances and controllers will become very stationary when installed
- Will be more directly exposed to the internet and thereby internet criminals

### HomePlug (Wired)

A HomePlug solution reuses the existing internal power lines in the home for data communication. The technology is using the IEEE 1901 standard (IEEE Standards Association, 2008) that provides data rates between 4 – 10 Mbps between the units. The IEEE 1901 (Also known as HomePlug GP) is a low power and low speed version of the original HomePlug AV technology that could have data rates up to 200Mbps. HomePlug requires each appliance and controller unit on its network to have a HomePlug adapter which uses the TCP/IP protocol. As the communication is happening within a power cord, you will get a more noisy communication than other standards and it can therefore be more easily affected from outer interference.

#### Advantages

- No need for new cables as it runs with existing power lines
- Only a single connection to the appliance is needed – power cord serves two functions
- Data rates of 4-10 Mbps surpass smart grid requirements
- If AES-128 standard is implemented then it is a safe alternative to Ethernet
- Can support different kinds of protocols with the use of adapters

#### Disadvantages

- Outer parameters can give noisy communication
- Need to be configured well to not be viewable to others

- The amount of devices on the market is limited
- The acceptance of a HomePlug standard is only growing slowly

## Wi-Fi (Wireless)

With a Wi-Fi HAN solution you will have a network consisting of appliances and controllers that all use the well-known 802.11 standards. By using the state-of-the-art IEEE 801.11n (IEEE Standards Association, 2009) standard you will get a fast and stable wireless network that a lot of consumers and businesses are already using. As it is already a mature and proven technology, like Ethernet, a lot of security mechanisms have been developed that can handle the outside threats. Security specifications like WEP, WPA, WPA2 and AES are all integrated into the newer 801.11n technology. With n-technology you will get data rates up to 300Mbps and with other version like g- and a-technologies you will get 54Mbps.

### Advantages

- Popular wireless technology across smart phones, computers, home networks etc.
- IP based technology
- Mature security standards
- High data rate on 300 Mbps
- Cheap integration because of low price on chipsets

### Disadvantages

- Too high energy consumption for running on batteries in a reasonable period
- If in low power version the price will be too high

## ZigBee (Wireless)

The standard called ZigBee Home Automation (in this report called ZigBee) is developed as a low power and low data rate communication standard (ZigBee Alliance, 2012). It uses the IEEE 802.15.4 (IEEE Standards Association, 2011) standard for its layers. The main benefit of the ZigBee technology is that it can run on appliances which are driven by batteries due to its low energy consumption. It uses a lower frequency than other wireless alternatives which also gives a lower data rate, but it is still fulfilling the requirements of a HAN. ZigBee is created in such a low powered way that it can easily be integrated into a lot of simpler device as for example health care products, remote controls, retail services, telephone products and other simple electronic devices.

ZigBee has been developed specifically for a smart home (ZigBee Alliance, 2012) and therefore has some elements in its specification that are smart grid related. The technology consists of three main actors that need to be active on the network before it can operate:

- ZigBee End Device (ZED) is the simple devices that need to be controlled on the HAN, in other words the appliances. The device has a minimal amount of functionalities and can be battery powered.
- ZigBee Coordinator (ZC) which is a controller unit. Only one ZC can be connected to the network. It contains security models that accepts and enables the connect ZEDs.
- ZigBee Router (ZR) is a normal router that uses the signal from the ZigBee standard. It is responsible for routing the data between devices.

### Advantages

- Already a lot of commercial products on the market
- Low cost and power
- Specifically designed as a smart home technology

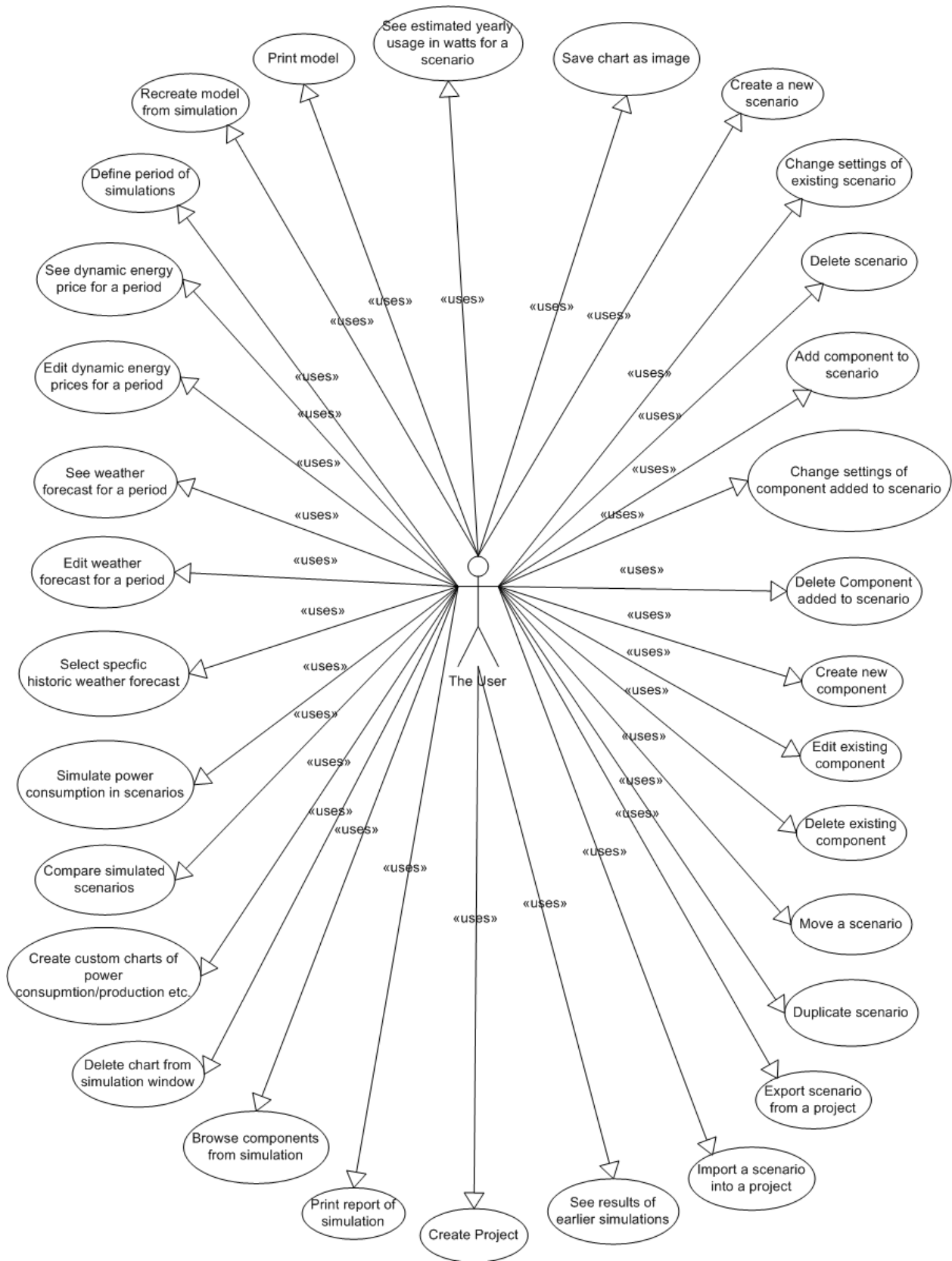
### Disadvantages

- Bandwidth limit on each channel on 20 Kbps which makes it unfit for larger networks
- Security mechanism can optionally be enabled
- Low range inside household

There are a lot of factors that need to be taken into consideration when comparing the presented technologies for the HAN communication. When looking at all the listed advantages and disadvantages it is clear that there is no one right answer to which technology to choose and each of the presented technologies is great for a specific scenario. ZigBee for example is perfect for an appliances-only HAN with no computers or handheld devices as ZigBee is built for a HAN structure and provides low cost communication. But ZigBee has a problem with networks with a demand for a higher distance range between the appliances. So in order to make the right decision on which technology to use in a specific scenario one should compare the specifications of the technology with the scenario in question. The most relevant specifications are listed in the table below.

	<b>Ethernet</b>	<b>HomePlug</b>	<b>Wi-Fi</b>	<b>ZigBee</b>
Data Rate	10-1000Mbps	4-8 Mbps	54-300 Mbps	320 Mbps in EU
Frequency	-	2-30 MHz	2.4GHz	868 MHz
Battery life	-	-	Weeks	Years
Security	On Device	56-bit DES	802.11i (WPA2)	128-bit AES
Range (In- & Outside)	100 m p. cable	300 m p. cable	100+ m	10 m to 75 m
Latency	Low	Low	Low	High

# Appendix C: Use Case Diagram



# Appendix D: Detailed Use Case Descriptions

See the 37 detailed use cases on the following 37 pages.



## [1] Create Project

Name	Create project	
Purpose	The user can create a new project	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	-	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Create one or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Create Project</i> button in the top menu	
2.		The system opens a dialog asking whether the user wants to save the currently open project before closing it
3.	The user clicks <i>Yes</i>	
4.		The project is saved and closed
5.		A new project dialog is displayed
6.	The user enters a name for the project and clicks <i>Create</i>	
7.		The project is created and opened
Postcondition	The currently open project is saved and closed and a new project is created and opened	
<b>Alternative flow (no project is currently open)</b>		
Step	The Actor	The System
2a.		No project is currently open: Skip to <u>step 5</u>
Postcondition	A new project is created and opened	

## [2] Create a New Scenario

Name	Create a new scenario	
Purpose	<b>The user</b> creates a new scenario on the canvas	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	-	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow (no scenarios exist)</b>		
Step	The Actor	The System
1.	<b>The user</b> drags and drops a house from the toolbox onto the canvas	
2.		A Create Scenario Wizard is displayed
3.	<b>The user</b> fills in information about name of scenario, number of residents, heating type, size, energy rating, etc. and clicks Finish	
4.		The system creates the scenario
Postcondition	The scenario is created	
<b>Alternative flow 1 (at least one scenario already exists)</b>		
Step	The Actor	The System
2a.		<b>The user</b> is given the option to bind the settings of the new scenario to the existing scenario
Postcondition	The scenario is created	
<b>Alternative flow 2 (top menu)</b>		
Step	The Actor	The System
1b.	<b>The user</b> clicks the <i>New Scenario</i> button on the top menu (continue at <u>step 2</u> )	
Postcondition	The scenario is not created	
<b>Exception flow 1 (invalid information filled in)</b>		
Step	The Actor	The System
4c.		<b>The user</b> is informed that the information is invalid and asks the user to type in different information (continue at <u>step 3</u> )
Postcondition	The scenario is not created	

### [3] Add Component to Scenario

Name	Add component to scenario	
Purpose	The user can add components from the toolbox to the existing scenarios	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	A scenario already exists on the canvas	
Related GUI Mock-up	Main Window	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> drags a component from the toolbox and drops it onto a scenario on the canvas	
2.		The component is added to the canvas and connected with a line to the scenario closest to where the component was dropped
3.		A text is displayed next to the component informing <b>the user</b> that he can change the settings of the component
Postcondition	The component is added to the canvas and connected to a scenario	

#### [4] Delete Component Added to Scenario

Name	Delete component added to scenario	
Purpose	The user can delete a component from the canvas that has been added to an existing scenario.	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	A component already exists on the canvas	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks the component and clicks the <i>Delete Component</i> button on the top menu or the <i>Delete</i> button on the keyboard	
2.		A message dialog appears asking the user if he is sure he wants to delete the component
3.	The user clicks <i>Yes</i>	
4.		The component is deleted
Postcondition	The component has been deleted	
<b>Alternative flow 1 (Right click)</b>		
Step	The Actor	The System
1a.	The user right clicks on the component and clicks on the <i>Delete</i> option that appears in the right click menu (continue at <u>step 2</u> )	
Postcondition	The component has been deleted	
<b>Alternative flow 2 (Mouse over)</b>		
Step	The Actor	The System
1b.	The user hovers the mouse over the component and clicks on the <i>Delete Component</i> button that appears on the component (continue at <u>step 2</u> )	
Postcondition	The component has been deleted	
<b>Exception flow (The user clicks No to delete)</b>		
Step	The Actor	The System
3c.	The user clicks <i>No</i> in the message dialog	
Postcondition	The component is not deleted	

## [5] Simulate Power Consumption in Scenarios

Name	Simulate Energy Consumption in Scenarios	
Purpose	The user can run a simulation of multiple scenarios and get the power consumption of each scenario as well as other data about the simulation	
Actors	<b>The user</b> of the application	
Frequency	Very often	
Preconditions	At least one scenario is created on the canvas and <u>Define Period of Simulation</u> has been performed	
Related GUI Mock-up	Simulation Window, Simulation ribbon	
Related Functional Requirement	Simulate energy consumption in a scenario	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> clicks on the <i>Run Simulation</i> button on the top menu	
2.		The system performs a simulation of each scenario on the canvas
3.		The results of the simulation are displayed in a window
Postcondition	The scenarios have been simulated	
<b>Exception flow</b>		
Step	The Actor	The System
2a.		Something goes wrong in the simulation
		An error message is displayed to the user informing him of what went wrong and how to fix it if possible
Postcondition	The scenarios have not been simulated	

## [6] Compare Simulated Scenarios

Name	Compare simulated scenarios	
Purpose	The user can compare overall results of each of the scenarios in a simulation	
Actors	<b>The user</b> of the application	
Frequency	Very often	
Preconditions	-	
Related GUI Mock-up	Simulation Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<u>Simulate power consumption in scenarios</u>	
2.		The overall energy consumption in the time period is displayed for each of the scenarios in the same diagram along with information about when each component has started and stopped. The total energy bill is displayed below.
3.	The user chooses the scenarios to compare in a dropdown menu	
4.		The diagram is updated and only the selected scenarios are displayed
Postcondition	The selected scenarios are displayed	

## [7] Change Settings of Existing Scenario

Name	Change Settings of Existing Scenario	
Purpose	<b>The user</b> can change the settings of a scenario on the canvas	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	A scenario already exists	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> double-clicks the house on the canvas	
2.		The settings window for the scenario is displayed
3.	<b>The user</b> changes the settings and clicks <i>OK</i>	
4.		The settings are validated and updated
Postcondition	The settings have been changed	
<b>Alternative flow 1 (Top menu)</b>		
Step	The Actor	The System
1a.	<b>The user</b> clicks on the scenario and clicks on the <i>Edit Scenario</i> button on the top menu (continue at <a href="#">step 2</a> )	
Postcondition	The settings have been changed	
<b>Alternative flow 2 (Mouse over)</b>		
Step	The Actor	The System
1b.	<b>The user</b> hovers the mouse over the scenario and clicks the <i>Edit Scenario</i> button that appears on the house (continue at <a href="#">step 2</a> )	
Postcondition	The settings have been changed	
<b>Alternative flow 3 (Right-click)</b>		
Step	The Actor	The System
1c.	<b>The user</b> right-clicks on the scenario and clicks <i>Properties</i> (continue at <a href="#">step 2</a> )	
Postcondition	The settings have been changed	
<b>Exception flow 1 (Invalid settings)</b>		
Step	The Actor	The System

4d.		The information is invalid and <b>the user</b> is asked to reenter correct settings (continue at <u>step 3</u> )
Postcondition	The settings have not been changed	



## [8] Change Settings of Component Added to a Scenario

Name	Change settings of component added to a scenario	
Purpose	The user can change the settings of a component that is connected to a scenario on the canvas.	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	A component exists and it is connected to a scenario	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario), Create custom components to use in the modelling	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> double clicks on the component	
2.		The settings window for the component is displayed
3.	The user changes the settings in the settings window and clicks <i>OK</i>	
4.		The settings for the component are updated
Postcondition	The settings have been changed	
<b>Alternative flow 1 (Right click)</b>		
Step	The Actor	The System
1a.	The user right clicks on the component and clicks <i>Edit Component</i> (continue at <u>step 2</u> )	
Postcondition	The settings have been changed	
<b>Alternative flow 2 (Top menu)</b>		
Step	The Actor	The System
1b.	The user clicks on the component and clicks <i>Edit Component</i> on the top menu (continue at <u>step 2</u> )	
Postcondition	The settings have been changed	
<b>Alternative flow 3 (Mouse over)</b>		
Step	The Actor	The System
1c.	<b>The user</b> hovers the mouse over the component and clicks on the <i>Edit Component</i> button that appears on the component (continue at <u>step 2</u> )	

Postcondition	The settings have been changed	
<b>Exception flow (Invalid settings)</b>		
Step	The Actor	The System
4d.		The system informs the user that the settings are invalid and asks him to enter correct settings (continue at <u>step 3</u> )
Postcondition	The settings have not been changed	

## [9] Delete scenario

Name	Delete scenario	
Purpose	Delete a scenario that exists on the canvas	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	A scenario already exists on the canvas	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> clicks on the scenario and clicks the <i>Delete Scenario</i> button on the top menu or the <i>Delete</i> button on the keyboard	
2.		A message dialog appears asking <b>the user</b> if he is sure he wants to delete the scenario
3.	<b>The user</b> clicks <i>Yes</i>	
4.		The scenario is deleted from the canvas
Postcondition	The scenario is deleted	
<b>Alternative flow 1 (Right-click)</b>		
Step	The Actor	The System
1a.	<b>The user</b> right-clicks the scenario and clicks <i>Delete</i> (continue at <u>step 2</u> )	
Postcondition	The scenario is deleted	
<b>Alternative flow 2 (Mouse over)</b>		
Step	The Actor	The System
1b.	<b>The user</b> hovers the mouse over the scenario and clicks the <i>Delete Scenario</i> button that appears on the house (continue at <u>step 2</u> )	
Postcondition	The scenario is deleted	
<b>Exception flow</b>		
Step	The Actor	The System
3c.	<b>The user</b> clicks <i>No</i> in the dialog box	
Postcondition	The scenario is not deleted	

## [10] Define Period of Simulation

Name	Define period of simulation	
Purpose	The user can define the dates between which the simulation should run	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	-	
Related GUI Mock-up	Simulation ribbon	
Related Functional Requirement	Simulate energy consumption in a scenario	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> chooses the start date using the date chooser in the top menu	
2.	<b>The user</b> chooses the end date using the date chooser in the top menu	
Postcondition	The dates have been set	

## [11] Create Custom Charts of Power Consumption/Production etc.

Name	Create custom charts of power consumption/production etc.	
Purpose	<b>The user</b> can add custom graph to the simulation window and thereby to the report	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	The user has a simulation open in the simulation window open	
Related GUI Mock-up	Simulation Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects a start and end period that lays within the period of the simulation.	
2.	<b>The User</b> selects the parameter that he wants to get displayed in the chart.	
3.	<b>The User</b> selects the component he wants to see statistics for.	
4.	<b>The User</b> selects “add chart”	
5.		The system adds a new chart to the simulation window with the selected data.
Postcondition	A new custom chart has been added to the simulation window	
<b>Alternative flow 1 (All components)</b>		
Step	The Actor	The System
3a.	<b>The User</b> selects the scenarios that he wants to get shown in the chart	
Postcondition	A chart is created	
<b>Exception flow 1 (Invalid input)</b>		
Step	The Actor	The System
5b.		The system checks if the given input are valid and can work together. It finds it invalid and it prompts the user with the error describing why it is invalid.
Postcondition	No chart are added to the simulation window	

## [12] See Dynamic Energy Prices for a Period

Name	See dynamic energy prices for a period	
Purpose	<b>The user</b> want to inspect the power prices for a period	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	-	
Related GUI Mock-up	Data Input ribbon	
Related Functional Requirement	Import and edit energy prices	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the “inspect power prices” button in the top menu.	
2.		The data about the power prices are shown in a new table based window.
3.	<b>The User</b> selects two dates defining the period	
4.	<b>The User</b> selects what version of the data set they want to use. (Default or one of the user defined)	
5.		The selected data within the set period is displayed in the table.
Postcondition	The power price data is displayed to the user	
<b>Exception flow 1 (invalid information filled in)</b>		
Step	The Actor	The System
4a.		<b>The user</b> is informed that the information is invalid and there are no data available within the set period
Postcondition	No data displayed	

### [13] See Weather Forecast for a Period

Name	See weather forecast for a period	
Purpose	<b>The user</b> want to inspect the weather forecast for a period	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	-	
Related GUI Mock-up	Data Input ribbon	
Related Functional Requirement	Import and edit weather forecasts	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the “inspect weather forecast” button in the top menu.	
2.		The data about the weather information are shown in a new table based window.
3.	<b>The User</b> selects two dates defining the period	
4.	<b>The User</b> selects what version of the data set they want to use. (Default or one of the user defined)	
5.		The selected data within the set period is displayed in the table.
Postcondition	The power price data is displayed to the user	
<b>Exception flow 1 (invalid information filled in)</b>		
Step	The Actor	The System
4a.		<b>The user</b> is informed that the information is invalid and there are no data available within the set period
Postcondition	No data displayed	

## [14] Create New Component

Name	Create new component	
Purpose	<b>The user</b> can create a new specific component in the toolbox, e.g. a washing machine with the desired properties	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	-	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create custom components to use in the modelling	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> clicks on the <i>New Component</i> button on the top menu	
2.		A <i>Create Component</i> wizard is displayed
3.	<b>The user</b> fills in the information about the component and clicks the <i>Create</i> button	
4.		The component is created and added to the toolbox
Postcondition	The component is created and added to the toolbox	
<b>Alternative flow (right-click)</b>		
Step	The Actor	The System
1a.	<b>The user</b> right clicks the toolbox and clicks <i>New Component</i> in the menu that appears (continue at <u>step 2</u> )	
Postcondition	The component is created and added to the toolbox	
<b>Exception flow (invalid information)</b>		
Step	The Actor	The System
4b.		The information is invalid. The user is asked to type in new information ( <u>step 3</u> )
Postcondition	The component is not created	



## [15] Edit Existing Component

Name	Edit existing component	
Purpose	<b>The user</b> can edit a component in the toolbox, such as changing its name or the properties	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	A component exists in the toolbox	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create custom components to use in the modelling	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the component in the toolbox and then clicks on the <i>Edit Component</i> button in the top menu	
2.		A window is displayed where the user can change the information about the component
3.	The user changes the information about the component and clicks <i>OK</i>	
4.		The component in the toolbox and the instances on the canvas are updated
Postcondition	The component is changed and all added instances of the component on the canvas are updated as well	
<b>Alternative flow (right click)</b>		
Step	The Actor	The System
1a.	The user right-clicks on the component in the toolbox and clicks on <i>Edit</i> in the menu that appears (continue at <a href="#">step 2</a> )	
Postcondition	The component is changed	
<b>Exception flow (invalid information)</b>		
Step	The Actor	The System
4b.		The information entered by the user is invalid. The user is asked to retype correct information (continue at <a href="#">step 3</a> )
Postcondition	The component is not changed	

## [16] Delete Existing Component

Name	Delete existing component	
Purpose	The user can delete a component from the toolbox. It can then no longer be added to the scenarios	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	A component already exists in the toolbox	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the component in the toolbox and clicks on the <i>Delete Component</i> button in the top menu	
2.		A message dialog is displayed asking the user if he is sure he wants to delete the component
3.	The user clicks <i>Yes</i>	
4.		The component is deleted from the toolbox
Postcondition	The component is deleted	
<b>Alternative flow 1 (right click)</b>		
Step	The Actor	The System
1a.	The user right clicks on the component in the toolbox and clicks on <i>Delete</i> in the menu that appears (continue at <a href="#">step 2</a> )	
Postcondition	The component is deleted	
<b>Alternative flow 2 (keyboard)</b>		
Step	The Actor	The System
1b.	The user clicks on the component in the toolbox and clicks on <i>Delete</i> on the keyboard (continue at <a href="#">step 2</a> )	
Postcondition	The component is deleted	
<b>Exception flow</b>		
Step	The Actor	The System
3c.	The user clicks <i>No</i> in the message dialog	
Postcondition	The component is not deleted	

## [17] Delete Chart from Simulation Window

Name	Delete chart from simulation window	
Purpose	<b>The user</b> needs to delete chart from simulation window	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	Custom charts created in the simulation window	
Related GUI Mock-up	Simulation Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> finds the chart he wants to delete.	
2.	<b>The User</b> selects “delete” button in the top right corner of the chart	
3.		The system prompts user with a confirmation dialog to confirm deleting of chart
4.	<b>The User</b> selects “Yes”	
5.		The system deletes the chart from the simulation window.
Postcondition	The desired chart are removed from the simulation window.	
<b>Alternative flow 1 (No changes)</b>		
Step	The Actor	The System
2a.	<b>The User</b> right clicks on the chart and selects “delete chart” in the menu.	
Postcondition	Chart has been deleted	
<b>Alternative flow 2 (User do not want to delete)</b>		
Step	The Actor	The System
4b.	<b>The User</b> selects “No”	
Postcondition	Chart has not be deleted	

## [18] Save a Project

Name	Save a project	
Purpose	The user can save a project (collection of scenarios, simulations and settings) for later use	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	A project is currently open	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Save/load files	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Save Project</i> button on the top menu	
2.		A file dialog is displayed to the user
3.	The user selects a location and name for the file and clicks <i>OK</i>	
4.		The project is saved
Postcondition	The project is saved	
<b>Alternative flow 1 (previously saved)</b>		
Step	The Actor	The System
2a.		The project has previously been saved: Skip to <u>step 4</u>
Postcondition	The project is saved	
<b>Alternative flow 2</b>		
Step	The Actor	The System
1b.		The user clicks on Ctrl-S on the keyboard (continue at <u>step 2</u> )
Postcondition	The project is saved	
<b>Exception flow (Not enough space)</b>		
Step	The Actor	The System
4c.		There is not enough space on the harddrive: The saving is cancelled
Postcondition	The project is not saved	

## [19] Load a Project

Name	Load a project	
Purpose	The user can load a previously saved project to continue working on it	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	A project has previously been saved	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Save/load files	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Load Project</i> button on the top menu	
2.		A file dialog is displayed
3.	The user chooses the project he wishes to open and clicks <i>Open</i>	
4.		The project is opened. If the project contains components in the toolbox that do not exist in the current version of the application these are created.
Postcondition	The project is opened	
<b>Alternative flow (corrupt file)</b>		
Step	The Actor	The System
4a.		The file is corrupt: An error message is displayed to the user
Postcondition	The project is not opened	

## [20] Undo Action

Name	Undo action	
Purpose	The user can undo the previous action performed in the application	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	At least one action has been performed since the application was launched	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Undo/Redo mechanisms	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Undo</i> button on the top menu	
2.		The state of the application is returned to the state before the previous action was performed
Postcondition	The action is undone	
<b>Alternative flow</b>		
Step	The Actor	The System
1a.	The user presses Ctrl-Z on the keyboard (continue at <u>step 2</u> )	
Postcondition	The action is undone	

## [21] Redo Action

Name	Redo action	
Purpose	The user can redo an action he has undone	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	At least one action has been undone	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Undo/redo mechanisms	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Redo</i> button on the top menu	
2.		The previously undone action is redone
Postcondition	The action is redone	
<b>Alternative flow</b>		
Step	The Actor	The System
1a.	The user presses Ctrl-Y on the keyboard (continue at <u>step 2</u> )	
Postcondition	The action is redone	

## [22] Print Report of Simulation

Name	Print report of simulation	
Purpose	<b>The user</b> can print a report of the best findings from their comparisons	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	Simulations are created and the simulation window is open.	
Related GUI Mock-up	Simulation Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> creates the graphs needed.	
2.		The specific data from the simulations are plotted and shown to the user in the window.
3.	<b>The User</b> selects “print report”	
4.		System invokes a print method and a PDF are shown or it invokes it as a print job.
Postcondition	A report are printed as either PDF or on Paper	
<b>Alternative flow 1 (No custom reports)</b>		
Step	The Actor	The System
1a.	<b>The User</b> adds no custom graphs and just selects print	
2a.		The system prints only the overview of the selected simulation as a PDF or on Paper.
Postcondition	A report are printed as either PDF or on Paper	



### [23] See Estimated Yearly Usage in Watts for a Scenario

Name	See estimated yearly usage in watts for a scenario	
Purpose	The user can see an estimation of the amount of watts being used by the created scenario. This is based on approximations and not simulations.	
Actors	<b>The user</b> of the application	
Frequency	Very often	
Preconditions	A scenario is created	
Related GUI Mock-up	Main Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user adds/removes or edits a component in a scenario	
2.		The total usage of the scenario is updated and displayed on the canvas on the scenario
Postcondition	The total usage of watts is displayed	

## [24] Browse Components from Simulation

Name	Browse components from simulation	
Purpose	<b>The User</b> wants to inspect the components that were used in a given simulation.	
Actors	<b>The user</b> of the application	
Frequency	Rare	
Preconditions	The user is seeing the result of a simulation and have the simulation window open.	
Related GUI Mock-up	Simulation Window	
Related Functional Requirement	Create graphical reports	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The User</b> selects “inspect components” in the top toolbar	
2.		The system prompts the user with a selecting dialog with a list of the components within the scenarios.
3.	<b>The User</b> selects the component they want to inspect	
4.	<b>The User</b> selects “show” button.	
5.		The system opens a settings window in read only mode with the information about the component.
Postcondition	A window with information about desired component is displayed.	
<b>Exception flow 1 (No selection)</b>		
Step	The Actor	The System
5a.		The system shows a message dialog telling the user that they have not selected any component.
Postcondition	No information about a component are shown.	

## [25] Move a Scenario

Name	Move a scenario	
Purpose	The user can move a scenario on the canvas to another position	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	A scenario already exists on the canvas	
Related GUI Mock-up	Main Window	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks and holds the left mouse button and drags the scenario to a new position	
2.		The scenario and the attached components are moved with the mouse
3.	The user releases the mouse button	
4.		The scenario is placed in the position of the mouse
Postcondition	The scenario has got a new position where the mouse button was released	
<b>Exception flow 1 (outside canvas)</b>		
Step	The Actor	The System
3a.	The user drops the scenario outside of the canvas	
4a.		The scenario is returned to its previous position
Postcondition	The scenario is not moved	
<b>Exception flow 2 (click button)</b>		
Step	The Actor	The System
3b.	The user clicks on the right mouse button or Esc on the keyboard while dragging the scenario	
4b.		The scenario is returned to its previous position
Postcondition	The scenario is not moved	

## [26] Duplicate Scenario

Name	Duplicate scenario on the canvas	
Purpose	The user can duplicate a scenario on the canvas	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	A scenario already exists on the canvas	
Related GUI Mock-up	Main Window, Model ribbon	
Related Functional Requirement	Create One or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the scenario and clicks on the <i>Duplicate Scenario</i> button on the top menu	
2.		A duplicated version of the scenario appears next to the scenario
Postcondition	Two identical scenarios exist	
<b>Alternative flow 1 (keyboard)</b>		
Step	The Actor	The System
1a.	The user clicks on the scenario and presses Ctrl-C on the keyboard followed by Ctrl-V (continue at <u>step 2</u> )	
Postcondition	Two identical scenarios exist	
<b>Alternative flow 2 (right-click)</b>		
Step	The Actor	The System
1b.	The user right-clicks on the scenario and clicks on <i>Duplicate</i> on the menu that appears (continue at <u>step 2</u> )	
Postcondition	Two identical scenarios exist	

## [27] Edit Dynamic Energy Prices

Name	Edit dynamic energy prices	
Purpose	<b>The user</b> can modify the power prices to simulate changes	
Actors	<b>The user</b> of the application	
Frequency	Rare	
Preconditions	-	
Related GUI Mock-up	Data Input ribbon	
Related Functional Requirement	Import and edit energy prices	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the “inspect power prices” button in the top menu.	
2.		The data about the power prices are shown in a new table based window.
3.	<b>The User</b> selects the power prices and edits the value within the field	
4.	<b>The User</b> selects “save”	
5.		The system asks for a name to save the customized as.
6.	<b>The User</b> writes a name for the new user dataset	
7.		The system saves a new version of the dataset within the system containing the changed data.
Postcondition	A new custom dataset with power prices are created	
<b>Alternative flow 1 (No changes)</b>		
Step	The Actor	The System
5a.		The system will not save the data as there are no changes. It informs the user about it.
Postcondition	Nothing has been saved	
<b>Exception flow 1 (Invalid input)</b>		
Step	The Actor	The System
5b.		The price information in the field are below zero and not allow. The system will not save and informs the user about it.
Postcondition	Focus on field with invalid price.	

## [28] Edit Weather Forecast for a Period

Name	Edit weather forecast for a period	
Purpose	<b>The user</b> can modify the weather forecast to simulate specific changes	
Actors	<b>The user</b> of the application	
Frequency	Rare	
Preconditions	-	
Related GUI Mock-up	Data Input Ribbon	
Related Functional Requirement	Import and edit weather forecasts	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the “inspect weather forecast” button in the top menu.	
2.		The data about the weather information are shown in a new table based window.
3.	<b>The User</b> selects the weather information and edits the value within the field	
4.	<b>The User</b> selects “save”	
5.		The system asks for a name to save the customized as.
6.	<b>The User</b> writes a name for the new user dataset	
7.		The system saves a new version of the dataset within the system containing the changed data.
Postcondition	A new custom dataset with power prices are created	
<b>Alternative flow 1 (No changes)</b>		
Step	The Actor	The System
5a.		The system will not save the data as there are no changes. It informs the user about it.
Postcondition	Nothing has been saved	

## [29] See Results of Earlier Simulations

Name	See results of earlier simulations	
Purpose	<b>The user</b> want to see the results of earlier created simulations of scenarios	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	There has been created simulations in the past	
Related GUI Mock-up	Simulation Window, Simulation ribbon	
Related Functional Requirement	Simulate power consumption in a scenario	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The User</b> goes to the simulation top menu.	
2.	<b>The User</b> selects “inspect earlier simulations”	
3.		The system prompts the user with a message dialog with a list of earlier performed simulations.
4.	<b>The User</b> selects the desired old simulation in the list and press’s “show”.	
5.		The system opens the simulation window with the data from the chosen old simulation.
Postcondition	An old simulation is displayed to the user.	
<b>Alternative flow 1 (No old simulations)</b>		
Step	The Actor	The System
3a.		The system shows a message dialog telling the user that there are no old simulations.
Postcondition	No simulation is displayed to the user.	

### [30] Export a Scenario from a Project

Name	Export a scenario from a project	
Purpose	The user can export a scenario from a project to a file structure. This can be used to move a scenario from one machine to another or from one project to another.	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	At least one scenario has been created in the project currently open	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Save/load files	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the scenario and clicks on the <i>Export Scenario</i> button on the top menu	
2.		A save file dialog is presented to the user
3.	The user selects a name and location for the file and clicks <i>Save</i>	
4.		The scenario is exported to a file
Postcondition	The scenario is exported	
<b>Alternative flow (right-click)</b>		
Step	The Actor	The System
1a.	The user right-clicks on the scenario and clicks <i>Export</i> on the menu that appears (continue at <u>step 2</u> )	
Postcondition	The scenario is exported	
<b>Alternative flow (keyboard)</b>		
Step	The Actor	The System
1b.	The user clicks on the scenario and presses Ctrl-E on the keyboard (continue at <u>step 2</u> )	
Postcondition	The scenario is exported	



### [31] Import a Scenario Into a Project

Name	Import a scenario into a project	
Purpose	The user can import a scenario that has been exported into the project that is currently open.	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	An exported scenario already exists and a project is currently open	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Save/load files	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Import Scenario</i> button on the top menu	
2.		A file dialog is displayed
3.	The user chooses the scenario to import and clicks <i>Import</i>	
4.		The scenario is imported into the project. If the scenario contains components that do not exist in the currently open project's toolbox these are created.
Postcondition	The scenario is imported into the project	

## [32] Print Model

Name	Print model	
Purpose	The user can print the model on the canvas to a PDF file or to a printer	
Actors	<b>The user</b> of the application	
Frequency	Rarely	
Preconditions	-	
Related GUI Mock-up	Main Window, General ribbon	
Related Functional Requirement	Create graphical report	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the <i>Print Model</i> button on the top menu	
2.		The system opens a print dialog
3.	The user chooses the destination (PDF or printer) and clicks <i>Print</i>	
4.		The model is printed to the destination
Postcondition	The model is printed	
<b>Alternative flow (keyboard)</b>		
Step	The Actor	The System
1a.	The user presses Ctrl-P on the keyboard (continue at <u>step 2</u> )	
Postcondition	The model is printed	
<b>Exception flow (printer is not reachable)</b>		
Step	The Actor	The System
4b.		The printer is not reachable: The system displays an error message
Postcondition	The model is not printed	

### [33] Recreate Model from Simulation

Name	Recreate model from simulation	
Purpose	<b>The user</b> wants to recreate the model used for a simulation	
Actors	<b>The user</b> of the application	
Frequency	Rare	
Preconditions	The user has the simulation window open with a simulation.	
Related GUI Mock-up	Simulation Window, Main Window	
Related Functional Requirement	Create one or more models of a house (a scenario)	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> select “recreate model from this simulation” in the top menu of the simulation window.	
2.		The system prompts the user with a confirmation box asking if they are certain that they want to overwrite the current model on the canvas.
3.	<b>The user</b> selects “yes”	
4.		The system recreates the model on the canvas based on the model information in the simulation. It then closes the simulation window and focus on the main window with the new model.
Postcondition	New scenarios are displayed on the canvas, old contents get deleted.	
<b>Alternative flow 1 (at least one scenario already exists)</b>		
Step	The Actor	The System
3a.	The user selects “no”	
Postcondition	No new scenarios are created on the canvas, old content stays.	

### [34] Display Popup Hint

Name	Display popup hint	
Purpose	When the user performs a task that has a logical following task such as creating add components to a scenario after creating the scenario the system will display popup hints describing the next logical step	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	-	
Related GUI Mock-up		
Related Functional Requirement	Guide the user	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user performs an action with a logical following task	
2.		The system waits a designated amount of time
3.		The system shows the popup hint near the area from which the following task is reached
Postcondition	A popup hint is displayed	
<b>Exception flow</b>		
Step	The Actor	The System
2a.	The user starts a task before the designated amount of time has passed	
		The popup hint is not displayed
Postcondition	The popup hint is not displayed	

### [35] Hide Popup Hint

Name	Hide popup hint	
Purpose	The user can hide the popup hints when they are displayed, either permanently or just the one time	
Actors	<b>The user</b> of the application	
Frequency	Often	
Preconditions	A popup hint is displayed	
Related GUI Mock-up		
Related Functional Requirement	Guide the user	
<b>Normal flow</b>		
Step	The Actor	The System
1.	The user clicks on the x in the top right corner of the hint	
2.		The system hides the hint
Postcondition	The hint is hidden	
<b>Alternative flow</b>		
Step	The Actor	The System
1a.	The user checks the <i>Don't show this hint again</i> box and clicks on the x in the top right corner	
2a.		The system hides the hint permanently (it will not be displayed again)
Postcondition	The hint is hidden permanently	

### [36] Select Specific Historic Weather Forecast

Name	Select specific historic weather forecast	
Purpose	<b>The user</b> want to inspect the weather forecast for a specific event	
Actors	<b>The user</b> of the application	
Frequency	Sometimes	
Preconditions	-	
Related GUI Mock-up	Data Input ribbon	
Related Functional Requirement	Import and edit weather forecasts	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the “inspect weather forecast” button in the top menu.	
2.		The data about the weather information are shown in a new table based window.
3.	<b>The User</b> selects from a list a already defined weather event (for example storm December 2006)	
5.		The selected data within related to the selected event is displayed in the table.
Postcondition	The power price data is displayed to the user	

### [37] Save chart as Image

Name	Save chart as image	
Purpose	<b>The user</b> want to save a specific chart as image file	
Actors	<b>The user</b> of the application	
Frequency	Rare	
Preconditions	Charts created in the simulation window	
Related GUI Mock-up	Simulation window	
Related Functional Requirement	Create graphical report	
<b>Normal flow</b>		
Step	The Actor	The System
1.	<b>The user</b> selects the chart he want to save as image	
2.	<b>The user</b> selects the save icon/button next to the delete button next to the chart	
3.		The system converts the visual chart to a image and prompts a save dialog to the user
5.	<b>The user</b> selects the destination where he want to save the image file	
6.		The system saves the image in the desired destination
Postcondition	A image displaying the selected chart is saved in the defined destination.	

# Appendix E: ISO 9241-171 Conformity

On the following 9 pages a filled out checklist for the ISO 9241-171 standard can be seen.



Table C.1 — Checklist for assessing applicability and conformity with this part of ISO 9241

Clause/subclause of this part of ISO 9241	General guidelines and requirements	Applicability			Conformance			Comments
		Yes/No	Reason not applicable	Yes	Partially	No		
<b>8</b>	<b>General guidelines and requirements</b>							
<b>8.1</b>	<b>Names and labels for user-interface elements</b>							
8.1.1	Provide a name for each user-interface element	Y		X				
8.1.2	Provide meaningful names			X				
8.1.3	Provide unique names within context			X				
8.1.4	Make names available to assistive technology	Y			X			
8.1.5	Display names			X				
8.1.6	Provide names and labels that are short			X				
8.1.7	Provide text label display option for icons				X			
8.1.8	Properly position the labels of user-interface elements on the screen			X				
<b>8.2</b>	<b>User preference settings</b>							
8.2.1	Enable individualization of user-preference settings				X			
8.2.2	Enable adjustment of attributes of common user-interface elements				X			
8.2.3	Enable individualization of the user interface look and feel					X		
8.2.4	Enable individualization of the cursor and pointer	C				X		
8.2.5	Provide user-preference profiles					X		
8.2.6	Provide capability to use preference settings across locations					X		
8.2.7	Enable user control of timed responses	C				X		

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Applicability		Conformance				Comments
	Yes/No	Reason not applicable	Yes	Partially	No		
<b>8.3</b>	<b>Special considerations for accessibility adjustments</b>						
8.3.1	Y	/			X		
8.3.2		X					
8.3.3	Y	/			X		
8.3.4		X					
8.3.5		X					
8.3.6		X					
<b>8.4</b>	<b>General control and operation guidelines</b>						
8.4.1			X				
8.4.2			X				
8.4.3			X				
8.4.4	C			X			
8.4.5	C				X		
8.4.6			X			<i>in Release</i>	
8.4.7			X				
8.4.8				X			
8.4.9	Y	/	X				
8.4.10			X				
8.4.11			X				
8.4.12					X		

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Yes/No	Applicability		Conformance				Comments
		Reason not applicable	Yes	Partially	No			
<b>8.5</b>	<b>Compatibility with assistive technology</b>							
8.5.1	General						X	
8.5.2	Enable communication between software and assistive technology	Y	/				X	
8.5.3	Use standard accessibility services	Y	/				X	
8.5.4	Make user-interface element information available to assistive technologies	Y	/				X	
8.5.5	Allow assistive technology to change keyboard focus and selection	Y	/				X	
8.5.6	Provide user-interface element descriptions	C	/	X				
8.5.7	Make event notification available to assistive technologies	Y	/				X	
8.5.8	Allow assistive technology to access resources						X	
8.5.9	Use system-standard input/output	C	/		X			
8.5.10	Enable appropriate presentation of tables	Y	/	X				
8.5.11	Accept the installation of keyboard and/or pointing device emulators	Y	/	X				
8.6.12	Allow assistive technologies to monitor output operations	Y	/		X			
8.5.13	Support combinations of assistive technologies						X	
<b>8.6</b>	<b>Closed systems</b>							
8.6.1	Read content on closed systems	Y	/				X	
8.6.2	Announce changes on closed systems	Y	/				X	
8.6.3	Operable through tactilely discernable controls	Y	/				X	
8.6.4	Pass through of system functions	Y	/				X	

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Applicability		Conformance				Comments
	Yes/No	Reason not applicable	Yes	Partially	No		
<b>9</b>	<b>Inputs</b>						
<b>9.1</b>	<b>Alternative Input options</b>						
9.1.1	Provide keyboard input from all standard input mechanisms		X				
9.1.2	Provide parallel keyboard control of pointer functions	Y			X		Not tested
9.1.3	Provide pointer control of keyboard functions				X		
9.1.4	Provide speech recognition services				X		
9.1.5	Provide system-wide spell-checking tools				X		
<b>9.2</b>	<b>Keyboard focus</b>						
9.2.1	Provide keyboard focus and text cursor	Y		X			
9.2.2	Provide high visibility keyboard focus and text cursors	Y		X			
9.2.3	Restore state when regaining keyboard focus				X		Not tested
<b>9.3</b>	<b>Keyboard input</b>						
9.3.1	General		X				
9.3.2	Enable full use via keyboard	C			X		
9.3.3	Enable sequential entry of multiple (chorded) keystrokes	Y		X			
9.3.4	Provide adjustment of delay before key acceptance	Y			X		
9.3.5	Provide adjustment of same-key double-strike acceptance	Y			X		
9.3.6	Provide adjustment of key repeat rate				X		
9.3.7	Provide adjustment of key repeat onset				X		
9.3.8	Allow users to turn key repeat off	Y			X		
9.3.9	Provide notification about toggle-key status				X		
9.3.10	Provide accelerator keys		X				

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Applicability				Conformance		Comments
	Yes/No	Reason not applicable	Yes	Partially	No		
9.3.11	Provide implicit or explicit designators						
9.3.12	Reserve accessibility accelerator key assignments	Y	X	X			
9.3.13	Enable remapping of keyboard functions						
9.3.14	Separate keyboard navigation and activation	Y				X	
9.3.15	Follow platform keyboard conventions			X		X	
9.3.16	Facilitate list and menu navigation				X		
9.3.17	Facilitate navigation of controls by grouping						
9.3.18	Arrange controls in task-appropriate navigation order			X			
9.3.19	Allow users to customize accelerator keys					X	
<b>9.4</b>	<b>Pointing devices</b>						
9.4.1	General			X			
9.4.2	Provide direct control of pointer position from external devices	Y				X	
9.4.3	Provide easily-selectable pointing-device targets			X			
9.4.4	Enable the reassignment of pointing-device button functions	Y				X	
9.4.5	Provide alternative input methods for complex pointing-device operations					X	
9.4.6	Enable pointing-device button-hold functionality	Y		X			Delay-and-dropt
9.4.7	Provide adjustment of delay of pointing-device button-press acceptance			X			
9.4.8	Provide adjustment of minimum drag distance			X			
9.4.9	Provide adjustment of multiple-click parameters	Y				X	
9.4.10	Provide adjustment of pointer speed	Y				X	

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Applicability				Conformance		Comments
	Yes/No	Reason not applicable	Yes	Partially	No		
9.4.11	Provide adjustment of pointer acceleration	C				X	
9.4.12	Provide adjustment of pointer movement direction					X	
9.4.13	Provide a means of finding the pointer	C				X	
9.4.14	Provide alternatives to simultaneous pointer operations	Y				X	
<b>10</b>	<b>Outputs</b>						
<b>10.1</b>	<b>General output guidelines</b>						
10.1.1	Avoid seizure-inducing flash rates	Y		X			
10.1.2	Enable user control of time-sensitive presentation of information	Y		X			
10.1.3	Provide accessible alternatives to task-relevant audio and video	C				X	NO sound or video
<b>10.2</b>	<b>Visual output (displays)</b>						
10.2.1	Enable users to adjust graphic attributes				X		
10.2.2	Provide a visual information mode usable by users with low visual acuity				X		
10.2.3	Use text characters as text, not as drawing elements			X			
10.2.4	Provide keyboard access to information displayed outside the physical screen	C			X		"shortcuts"
<b>10.3</b>	<b>Text/fonts</b>						
10.3.1	Do not convey information by visual font attribute alone					X	
10.3.2	Enable users to set minimum font size					X	
10.3.3	Adjust the scale and layout of user-interface elements as font-size changes			X			

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Yes/No	Applicability		Conformance				Comments
		Reason not applicable	Yes	Partially	No			
<b>10.4</b>								
<b>Colour</b>								
10.4.1	Y		X					
10.4.2					X			
10.4.3					X			
10.4.4						X		
10.4.5			X					
<b>10.5</b>								
<b>Window appearance and behaviour</b>								
10.5.1			X					
10.5.2			X					
10.5.3	Y		X					
10.5.4	Y			X				
10.5.5	Y				X			
10.5.6			X					
10.5.7	Y		X					
10.5.8			X					
10.5.9			X					
10.5.10	Y		X					

Table C.1 (continued)

Clause/subclause of this part of ISO 9241	Applicability				Conformance		Comments
	Yes/No	Reason not applicable	Yes	Partially	No		
<b>10.6</b>	<b>Audio output</b>						
10.6.1	Use tone pattern rather than tone value to convey information		X				
10.6.2	Enable control of audio volume	Y				X	
10.6.3	Use an appropriate frequency range for non-speech audio		X				
10.6.4	Enable adjustment of audio output		X				
10.6.5	Control of background and other sound tracks		X				
10.6.6	Use specified frequency components for audio warnings and alerts		X				
10.6.7	Allow users to choose visual alternative for audio output	Y				X	
10.6.8	Synchronize audio equivalents for visual events	Y				X	
10.6.9	Provide speech output services	Y				X	
<b>10.7</b>	<b>Text equivalents of audio (captions)</b>						
10.7.1	Display any captions provided	Y				X	
10.7.2	Enable system-wide control of captioning		X				
10.7.3	Support system settings for captioning	Y				X	
10.7.4	Position captions to not obscure content		X				
<b>10.8</b>	<b>Media</b>						
10.8.1	Enable users to stop, start and pause	Y				X	
10.8.2	Enable users to replay, rewind, pause and fast- or jump-forward		X				
10.8.3	Allow user to control presentation of multiple media streams		X				
10.8.4	Update equivalent alternatives for media when the media changes		X				










Table C.1 (continued)


Clause/subclause of this part of ISO 9241	Yes/No	Applicability		Conformance			Comments
		Reason not applicable	Yes	Partially	No		
<b>10.9</b>	<b>Tactile output</b>						
10.9.1	Do not convey information by tactile output alone						
10.9.2	Use familiar tactile patterns		X				
10.9.3	Enable tactile output to be adjusted		X				
<b>11</b>	<b>On-line documentation, "Help" and support services</b>						
<b>11.1</b>	<b>Documentation and "Help"</b>						
11.1.1	Provide understandable documentation and "Help"			X			
11.1.2	Provide user documentation in accessible electronic form	Y		X			
11.1.3	Provide text alternatives in electronic documentation and "Help"	Y		X			
11.1.4	Write instructions and "Help" without unnecessary device references			X			
11.1.5	Provide documentation and "Help" on accessibility features	Y				X	
<b>11.2</b>	<b>Support services</b>						
11.2.1	Provide accessible support services	Y				X	
11.2.2	Provide accessible training material					X	

## Appendix F: Unit Test Results


Below a detailed overview of the unit test can be seen. Here all the specific test for each test category can be examined.


✓  <Common.Tests> (30 tests)	Success
✓  Common.Tests (30 tests)	Success
✓  DragDropDataObjectTest (5 tests)	Success
✓ GetDataPresentTest	Success
✓ GetDataTestComponent	Success
✓ GetDataTestScenario	Success
✓ SetDataTest	Success
✓ SetDataTestNewFormat	Success
✓  IconProviderTest (6 tests)	Success
✓ GetIconUriTest	Success
✓ GetIconUriTestNotExist	Success
✓ GetIconUriTestNull	Success
✓ InitializeIconsTest	Success
✓ SetIconTest	Success
✓ SetIconTestAlreadyExist	Success
✓  StaticResourcesTest (8 tests)	Success
✓ SortElementsStressTest100x100	Success
✓ SortElementsStressTest10x50	Success
✓ SortElementsStressTest1x100	Success
✓ SortElementsStressTest1x1000	Success
✓ SortElementsStressTest3x100	Success
✓ SortElementsTestEightElements2x3	Success
✓ SortElementsTestEmpty	Success
✓ SortElementsTestThreeElements1x1	Success
✓  UndoRedoManagerTest (11 tests)	Success
✓ CanRedoTest	Success
✓ CanRedoTestEmpty	Success
✓ CanUndoTest	Success
✓ CanUndoTestEmpty	Success
✓ CombinedTest	Success
✓ DoActionTest	Success
✓ DoActionTestNull	Success
✓ RedoTest	Success
✓ RedoTestEmpty	Success
✓ UndoTest	Success
✓ UndoTestEmpty	Success

✓  <Common.Modelling.Tests> (30 tests)	Success
✓ {} Common.Modelling.Tests (30 tests)	Success
✓ AddElementActionTest (17 tests)	Success
✓ AddElementActionCtrTest	Success
✓ AddElementActionCtrTestNull	Success
✓ DoImplTestComponent	Success
✓ DoImplTestComponentNoScenario	Success
✓ DoImplTestScenario	Success
✓ FindClosestScenarioTest	Success
✓ FindClosestScenarioTestNoScenario	Success
✓ FindClosestScenarioTestOnTop	Success
✓ FindClosestScenarioTestTwoScenariosDiffDistance	Success
✓ FindClosestScenarioTestTwoScenariosDiffDistance2	Success
✓ FindClosestScenarioTestTwoScenariosSameDistance	Success
✓ FindClosestScenarioTwoScenariosNegativePosition	Success
✓ FindClosestScenarioTwoScenariosNegativePosition2	Success
✓ UndoImplTest	Success
✓ UndoImplTestComponent	Success
✓ UndoImplTestNotInList	Success
✓ UndoImplTestWrongComponent	Success
✓ DeleteElementActionTest (5 tests)	Success
✓ DoImplElementsEmptyTest	Success
✓ DoImplNullElementsTest	Success
✓ DoImplObjectNullTest	Success
✓ DoImplTest	Success
✓ UndoImplTest	Success
✓ DraggableCanvasObjectViewModelTest (3 tests)	Success
✓ GetDragEffectsTest	Success
✓ SetPositionFromCenterNoSizeTest	Success
✓ SetPositionFromCenterSuccessTest	Success
✓ MoveElementActionTest (5 tests)	Success
✓ DoImplElementsEmptyTest	Success
✓ DoImplNoneExistingTest	Success
✓ DoImplObjNullTest	Success
✓ DoImplTest	Success
✓ UndoImplTest	Success

✓  <CanvasModule.Tests> (12 tests)	Success
✓ {} CanvasModule.Tests (12 tests)	Success
✓ CanvasModuleTest (1 test)	Success
✓ CanvasViewModelTest (11 tests)	Success
✓ AddCanvasObjectTest	Success
✓ AddCanvasObjectTestObjNull	Success
✓ DeleteComponentObjNullTest	Success
✓ DeleteComponentTest	Success
✓ DeregisterCommandsTest	Success
✓ DropObjNullTest	Success
✓ DropTest	Success
✓ ElementsTest	Success
✓ GetDropEffectsObjNullTest	Success
✓ GetDropEffectsTest	Success
✓ RegisterCommandsTest	Success



✓  <PowerModule.Tests> (18 tests)	Success
✓ {} PowerModule.Tests (18 tests)	Success
✓ NordPoolPowerProviderTest (5 tests)	Success
✓ GetDataTest	Success
✓ GetDataTestFutureNonExistant	Success
✓ GetDataTestNonExistant	Success
✓ GetDataTestTomorrow	Success
✓ GetDataTestTwoYears	Success
✓ PowerModuleTest (6 tests)	Success
✓ GetPowerDataAsyncTest	Success
✓ GetRemainingDataTestNoStorage	Success
✓ GetRemainingDataTestStorageAll	Success
✓ GetRemainingDataTestStorageBeginning	Success
✓ GetRemainingDataTestStorageEnd	Success
✓ GetRemainingDataTestStorageMiddle	Success
✓ PowerStorageManagerTest (7 tests)	Success
✓ GetDataTest	Success
✓ GetDataTestNoSource	Success
✓ GetDataTestStartEqualEnd	Success
✓ GetDataTestStartLessThanEnd	Success
✓ SaveDataTestEmptyList	Success
✓ SaveDataTestNoList	Success
✓ SaveDataTestNoSource	Success

✓  <ModellingModule.Tests> (22 tests)	Success
✓ {} ModellingModule.Tests (22 tests)	Success
✓ ModellingModuleTest (20 tests)	Success
✓ ConvertComponentToViewModelTestCooking	Success
✓ ConvertComponentToViewModelTestCoolingFreezing	Success
✓ ConvertComponentToViewModelTestEntertainment	Success
✓ ConvertComponentToViewModelTestHeating	Success
✓ ConvertComponentToViewModelTestLights	Success
✓ ConvertComponentToViewModelTestNull	Success
✓ ConvertComponentToViewModelTestSolarPanel	Success
✓ ConvertComponentToViewModelTestWashing	Success
✓ ConvertComponentToViewModelTestWindTurbine	Success
✓ ConvertScenarioToViewModelTest	Success
✓ ConvertScenarioToViewModelTestNull	Success
✓ CreateConnectionTest	Success
✓ CreateConnectionTestNullComponent	Success
✓ CreateConnectionTestNullScenario	Success
✓ CreateObjectInstanceTestComponent	Success
✓ CreateObjectInstanceTestComponentNull	Success
✓ CreateObjectInstanceTestScenario	Success
✓ CreateObjectInstanceTestScenarioNull	Success
✓ GetAllObjectsTest	Success
✓ RefreshElementsTest	Success
✓ XmlParserTest (2 tests)	Success
✓ GetComponentsTest	Success
✓ GetScenariosTest	Success

## **Appendix G: Case Studies – House**

On the next page a recording of the appliances within the test house for the Case Study House.

Item	Place	Type	Total Wattage (W)	On Time
Refridgerator	Kitchen	CoolingFreezing	110	always
Micro wave oven	Kitchen	Cooking	1000	15 m/w
Dish washer	Kitchen	Washing	2200	4 h/d
Oven	Kitchen	Cooking	2000	30 m/w
Lights x2	Kitchen	Lights	120	2 h/w
Radio	Kitchen	Entertainment	30	1 h/w
Coffee machine	Kitchen	Cooking	1400	40 m/w
Cooking Plate x 2	Kitchen	Cooking	4000	1 h/w
Lights x2	Living Room	Lights	22	6 h/d
TV	Living Room	Entertainment	120	2,5 h/d
DVD reader	Living Room	Entertainment	53	2 h/w
Sattellite receiver	Living Room	Entertainment	30	2,5 h/d
Digital picture frame	Living Room	Entertainment	7,7	2 h/m
Speakers x3	Living Room	Entertainment	260	1,25 h/d
Sound system	Living Room	Entertainment	45	1,25 h/d
Lights x2	Living Room	Lights	80	6 h/d
Lights x2	Living Room	Lights	80	1 h/d
Lights	Living Room	Lights	60	3 h/d
Computer w. screen x2	Office	Entertainment	200	2 h/d
Lights	Office	Lights	11	2 h/d
Lights x2	Office	Lights	40	1 h/d
Lights x6	Hallway	Lights	120	4 h/d
Lights x6	Hallway	Lights	100	5 h/d
Lights	Hallway	Lights	60	4 h/d
TV	Bedroom	Entertainment	152	1 h/m
Lights x3	Bedroom	Lights	60	5 m/d
Alarm clock	Bedroom	Entertainment	12	always
Lights x2	Bedroom	Lights	80	30 m/d
Sowing machine	Misc room	Entertainment	90	4 h/w
iPod Dock	Misc room	Entertainment	12	4 h/w
Flattening iron	Misc room	Bathroom	2400	30 m/w
Radio	Misc room	Entertainment	30	30 m/w
Hair dryer	Dressing room	Bathroom	1600	10 m/w
Lights x5	Dressing room	Lights	120	30 m/d
Lights	Bathroom	Lights	20	1 h/d
Electrical tooth brush charger	Bathroom	Bathroom	1	always
Dehumidifier	Basement	Washing	260	4 h/d
Washing machine	Basement	Washing	2300	4 h/w
Dryer	Basement	Washing	2900	30 m/w
Chest freezer	Basement	Washing	100	always
Lights x2	Basement	Lights	120	1 h/d

## **Appendix H: Case Studies – Apartment**

On the next page a recording of the appliances within the test apartment for the Case Study Apartment.

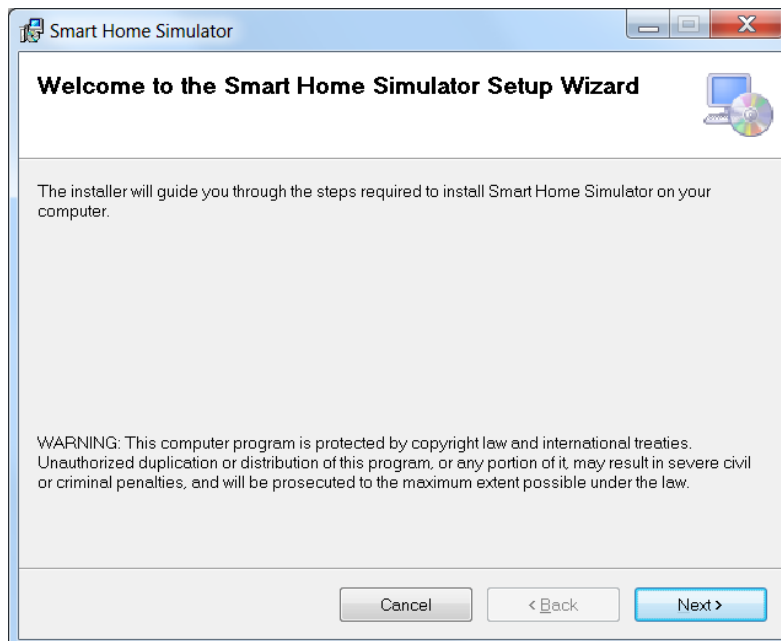


Item	Place	Type	Total Wattage (W)	On Time
Lamp (loft)	Hallway	Light	65	1 h/day
Lamp (bed)	Bedroom	Light	5	0,12 h/day
Lamp (bed)	Bedroom	Light	5	0,08 h/day
Clock	Bedroom	Entertainment	2	always
TV	Bedroom	Entertainment	130	0,5 h/day
Lamp (loft)	Bedroom	Light	65	0,1 h/day
TV	Livingroom	Entertainment	122	3,5 h/day
Video	Livingroom	Entertainment	30	0,001 h/day
Playstation 3	Livingroom	Entertainment	122	0,01 h/day
Stereo	Livingroom	Entertainment	50	0,005 h/day
Lamp (loft)	Livingroom	Light	40	0,05 h/day
Lamp (TV)	Livingroom	Light	15	1 h/day
Lamp (Table)	Livingroom	Light	11	1 h/day
Laptop 1	Office	Entertainment	90	1,5 h/day
Laptop 2	Office	Entertainment	90	1 h/day
Desktop - Box	Office	Entertainment	450	3 h/day
Desktop - Monitor 1	Office	Entertainment	24	3 h/day
Desktop - Monitor 2	Office	Entertainment	55	2 h/day
Lamp (loft)	Office	Light	11	0,5 h/day
Oven	Kitchen	Cooking	1020	0,5 h/day
Microwave	Kitchen	Cooking	800	0,01 h/day
Kettle	Kitchen	Cooking	1850	0,03 h/day
Toaster	Kitchen	Cooking	700	0,001 h/day
Coffeemaker	Kitchen	Cooking	1500	0,001 h/day
Blender	Kitchen	Cooking	650	0,001 h/day
Loftlampe	Kitchen	Light	60	0,01 h/day
Flatting iron	Bathroom	Bathroom	150	0,001 h/day
Vacum	Kitchen	Entertainment	1300	0,002 h/day
Fridge	Kitchen	Heating/Cooling	KWh/year: 235	always

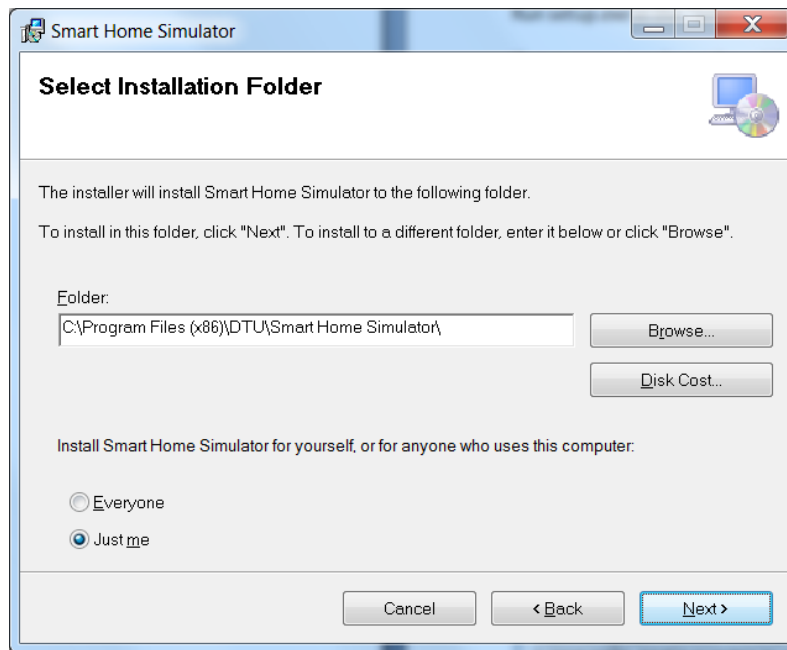
## Appendix I: Installation Guide

To install the program on your own computer, you will need to perform the following steps.

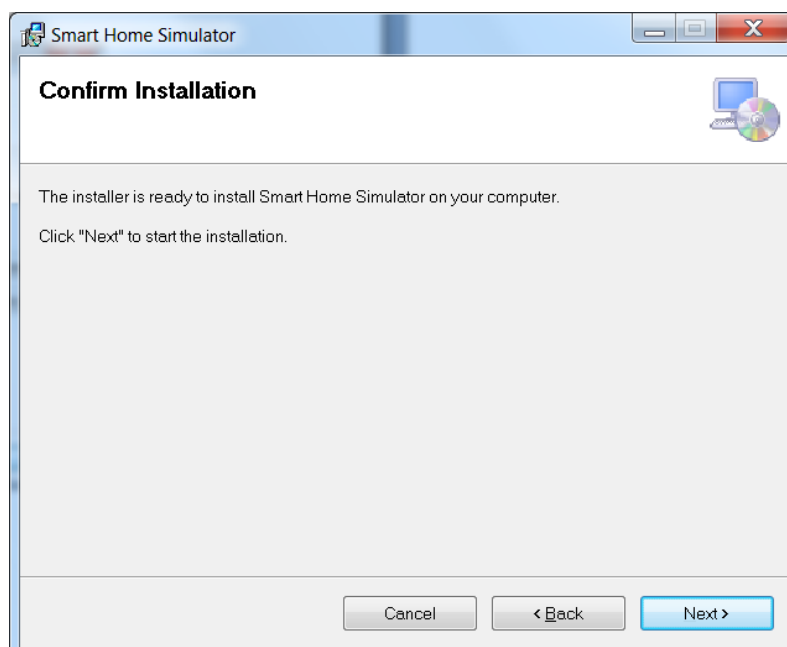
- 1) Run setup.exe in the “Smart Home Simulator” folder on the provided CD/Media.
- 2) If your computer do not have Microsoft .NET 4 Framework installed, the setup wizard will ask you to install this before going further with the installation. The setup will automatically start downloading and guide you through the installation of the framework.
- 3) If your computer do not have Microsoft SQL Server Compact Edition installed, the setup will ask you to install this before going further with the installation. The setup will automatically start downloading and guide you through the installation of the need software.
- 4) The setup wizard will now welcome you to Smart Home Simulator Setup Wizard, which you have to press “Next” to continue with the installation.



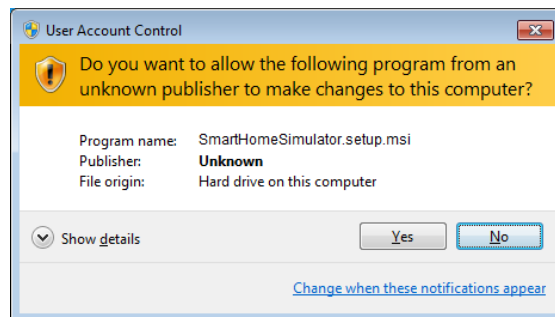
- 5) The setup will now require you to define the folder where the program can be installed in and in which scope you want to install the application (For yourself or for all the users on the computer).



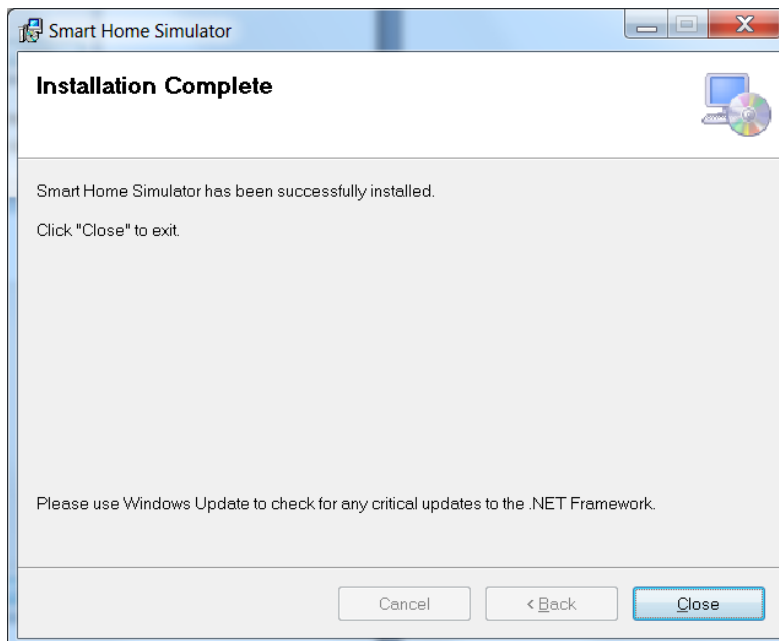
- 6) You now have to confirm that you want to install the program by pressing "Next" to continue, "Back" to change settings for the installation or "Cancel" to exit the installation.



- 7) The application will, while installing, ask you for administration rights, such that it can create and place files in the program folder. So allow this and proceed with the installation press “Yes”.



- 8) The installation will now complete and confirm it with the following window. To close the setup wizard press “Close”.



- 9) The program can now be opened for the first time by either using the new “Smart Home Simulator”-shortcut on your desktop or by going to the “Start menu” -> “All programs” -> “Smart Home Simulator”-folder -> “Smart Home Simulator”-shortcut.

## Appendix J: Paper of the Thesis

On the next 9 pages the paper which will be submitted to the conference "*The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies - ENERGY 2013*".

# Simulation Tool For Energy Consumption

## *The Development of a Simulation Tool for Measuring the Impact of a Smart Grid on a building.*

Michael Nysteen<sup>1</sup>, Henrik Mynderup<sup>1</sup>, Bjarne Poulsen<sup>1</sup> and Chresten Træholt<sup>2</sup>.

<sup>1</sup>Department of Informatics and Mathematical Modelling, Technical University of Denmark, Kongens Lyngby, Denmark

<sup>2</sup>Department of Electrical Engineering, Technical University of Denmark, Kongens Lyngby, Denmark  
{ s062413, s062373}@student.dtu.dk, bjp@imm.dtu.dk, ctr@elektro.dtu.dk.

Keywords: Smart grid, Simulation, Controller, Smart Home, Software, Graphical User Interface.

Abstract: In order to ensure the adoption of the smart grid with the general public it is necessary to be able to visualize the benefits of a smart home. Software tools that can simulate the effects can help significantly in this area. However, little work has been done in the area of visualizing energy consumption in smart homes. This paper presents a prototype of a simulation tool that allows graphical modeling of a home. Based on the modeled homes the user is able to simulate the energy consumptions and compare scenarios. The simulations are based on dynamic weather and energy price data as well as a controller unit of the user's choice. The results of the simulations can be compared using a dynamic reporting window that allows the user to create custom charts of the data. The application has been designed such that it can easily be extended with additional controller units, price and weather data as well as appliances and other electrical components used in the modeled homes.

## 1 PROBLEM & BACKGROUND

According to the Danish workgroup [1], the government needs to increase the awareness of the smart grid technology with the consumers to smooth the transition to smart grid. If the suggestion is carried out a tool that may help people realizing the effects of integrating a private home with the technologies needed.

The power grid today is highly affected by variations in the use of power during a day. The energy use during the night is very low, but the use of energy increases significantly during a couple of hours in the morning, afternoon and evening (called peak hours) when people come home from work. Today the demand in these peak hours are met by powering up so called *peaker plants* that can be started rapidly when more power is required [2]. These typically run on oil or natural gas and are very expensive to run. Furthermore, they produce a significant amount of pollution [3]. In some countries the consumption in the peak hours is reaching, or even exceeding, the full capacity of the grid and results in frequent power outages [4]. A way of shaving these peaks in energy consumption would be to give the consumer a motivation for

moving her energy consumption to other periods of the day. Such a motivation would likely be of a financial character, e.g. by making energy cheaper outside of peak hours.

In order to maximize the financial gain of dynamic pricing, energy should be consumed at the cheapest possible time. Several systems [5, 6], so called *controller units*, have been developed that control the energy consumption in the home by automatically increasing consumption when the prices are low and decreasing consumption during peak hours. This includes the Control4 EMS 100 controller system [5] and Tendril Connect [6] among others. Both of these systems can control any energy consumption in the home such as heating, lights and appliances as well as charging electric vehicles.

### 1.1 Network Architecture of the Smart Grid

The smart grid is based on a tiered architecture of a number of networks that connects all the parts of the smart grid, from power plants to the private home. The communication goes both ways so that each partner in the smart grid can communicate with each other in theory. The smart grid can be divided

into three main networks; **Wide Area Network (WAN)**, **Neighbourhood Area Network (NAN)** and **Home Area Network (HAN)**.

The solution is only focusing on the HAN, as this network is responsible for the communication between the appliances within the house. There are currently two overall approaches for constructing a Home Area Network without looking at which technologies to use. Common for both of the approaches is that the smart meter is connected to the HAN along with all the appliances.

- A Decentralized HAN solution is where each appliance on the network will be responsible of their own scheduling. Each appliance is communicating with the smart meter directly using the chosen standard and they contain embedded controller software that based on the current power price decides how the appliance needs to act. This price is the current market price for a KWh which the smart meter receives from the distribution network.
- A Centralized HAN network is a network where all the appliances are connected to a router which may be connected to the internet. The smart meter and a control unit are also connected to this network. In this scenario each of the appliances, which are connected to the network, only provides a software interface that can be used to control the appliance. This interface can then be accessed by the controller unit on the HAN. The controller unit will control the appliance based on the power price information that the controller unit gets from the smart meter.



Figure 1: An example of a centralized HAN.

A Home Area Network in itself does not provide any intelligence to the home. This intelligence comes from the controller units connected to it. By

making the user aware of the dynamic prices the consumer can save a significant amount of energy if he is using this information. A report from California Energy Commission [7] states that the average household can save 34.5% through technology used to inform others of the varying prices and each units automatically adjusting their consumption according to the price. In order to maximize the benefit from this information, however, you need something that can control energy consumption 24 hours a day. This is the so-called *controller unit*. A controller unit is placed in either a centralized or decentralized HAN (here it is actually not a physical unit, but instead a unit placed in each device). The centralized approach can be divided into two types of controller units: A physical unit installed in the home that controls each unit based on the customer's preferences or an external service communicating with the household over e.g. the Internet.

In order to make people aware of the potential benefits of intelligent energy consumption it is very useful to be able to simulate their home setup before investing in the required components. Therefore there is a need for simulation tools for such scenarios. So far only little work has been done in this area.

## 1.2 State-of-art simulation solutions

A number of simulation tools that examine the energy consumption in homes have been developed. Many tools such as GridLab-D [8] and EnergyPlus [9] examine the energy consumption with focus on heat generation and thermal load of a building. Very few tools examine the home in the context of a smart grid where the price of energy varies throughout the day. One of the few tools that were found is PowerMatcher Simulation Tool [10]. It is, however, only intended to demonstrate the PowerMatcher technology which is a home controller system and the tool is therefore scarce in features. It does e.g. not provide any means of modeling accurate energy consumption patterns based on many independent appliances, but simply allows the user to create a static *energy demand* [11].

None of the existing simulation tools consider the effects of using different controller units for controlling the home's energy consumption and none of them provide a modeling environment that allows the user to create an accurate representation of her home to be used in the simulations. This means that the existing tools cannot properly show a consumer how turning her home into a smart home will be of value to her. These shortcomings make up the foundation of this paper.

The remainder of this paper is organized as follows. Section 2 describes the identified requirements for the simulation tool and the targeted audience. Section 3 presents the core features of the prototype simulation tool and shows the Graphical User Interface (GUI). Finally, conclusion and future work are presented in sections 4 and 5.

## 2 SOLUTION REQUIREMENTS

As mentioned above a simulation tool was developed, which can provide reasonable simulations of how a private household is (financially) affected by a smart grid implementation. The simulation tool focuses on allowing the user to create a virtual household and add electrical appliances (washing machines, dish washers, refrigerators etc.) as well as local energy sources (such as wind turbines, solar cells etc.) to the household using a user friendly drag-and-drop approach. The focus of the simulations is to see how different scenarios affect the power bill when the energy consumption is controlled using various controller units.

In order to simulate this, the application will use environmental variables which includes the weather (affecting power generated by wind turbines, solar cells etc.) and the prices of buying energy from external energy sources at any given time. Most of these variables will be importable from real sources or manually configurable. The application supports the creation of multiple home scenarios and lets the user compare these, giving the user the possibility of evaluating the value of potential appliances that might be added to the house in the future. The scenarios can also be executed without smart-scheduling, thereby allowing the user to compare the scenarios with and without smart-scheduling enabled to see the effect of using controller units to schedule the energy consumption.

### 2.1 Target Audience

There have been identified three groups of people as potential users of the application. These groups are *the consumer*, *energy advisers/salesmen* and *educators/educates*. Each of these groups has been analyzed and their preferences in four categories have been compared: *Focus*, *Interaction with Application*, *Desired Output* and *Generic vs. Real Data*. *Focus* refers what the user wishes to achieve when using the application. *Interaction with Application* indicates what is most important to the user in the way he interacts with the application. *Desired Output* is the kind of output that supports

the focus of the user the best. Finally, *Generic vs. Real Data* refers to whether the user prefers realistic data about weather, energy prices, and more importantly manufacturer specific appliances or generic versions of the appliances and customizable weather and prices.

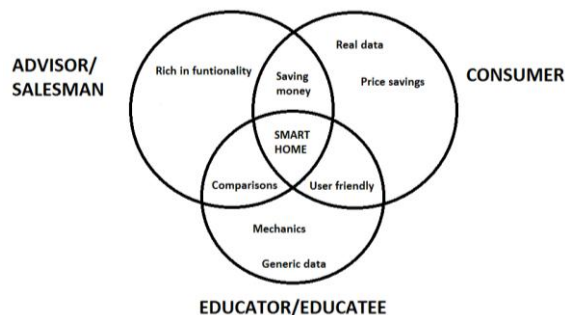


Figure 2: Shared interest of the target audiences.

The three user groups have different shared interests that they find important; Money savings, User friendliness and scenario comparisons. From the three main focus areas of the program it seems realistic to develop an application that can be used by all three user groups. Therefore the target audience is a combination of the three user groups.

### 2.2 Electric Components in the Home

In order to give the user the best possible results, the program needs to be as realistic as possible when modeling scenarios that they want simulated. Therefore the program has to contain realistic and user-customized content together with a realism check on the modeled setup.

In the program a home can consist of a lot of components such as lights, refrigerators, freezers, computers, TVs, washing machines etc. Some of the components, such as washing machines, can be scheduled based on dynamic prices while others, such as TVs, computers and lights, cannot. While you can still implement energy saving initiatives for non-schedulable components such as automatically turning off the components when you're not home, this thesis focuses on how components can be scheduled based on dynamic prices during a day. Therefore components will be placed in two groups; schedulable and non-schedulable components. Schedulable components will, as the name indicates, be scheduled by the simulation based on the dynamic prices, whereas the non-schedulable components will simply be run during a user-defined period each day regardless of the price.



### 3 SOLUTION PRESENTATION

When the user opens the program they are prompted with a startup dialog. Depending on whether the user chooses *new-*, *load-* or *latest* project the main window is shown either as empty or filled with content. The main window is shown in Figure 3 and the four main regions of the program are marked.

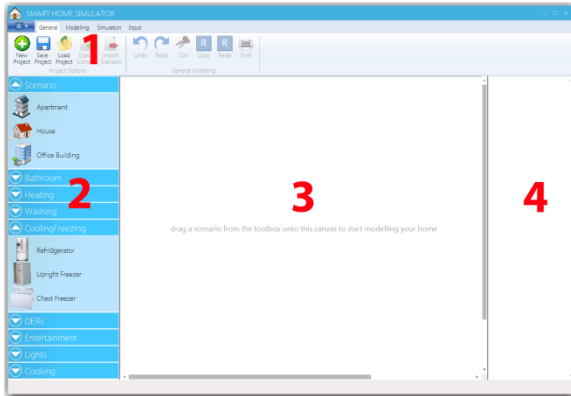


Figure 3: The main window of the application with each of the primary regions marked.

1. **Ribbon navigation:** This region provides the user with various options to manage the project including created models and simulations.
2. **Toolbox:** This region contains all the elements that can be used when modeling a home.
3. **Canvas:** The elements from the toolbox can be dragged onto this region to create the desired models of houses.
4. **Summary:** This region provides a brief overview of the yearly power consumption distribution in the modeled homes.
  - A. **Input:** The application provides data about power prices and weather to be used during the simulations.
  - B. **Simulation:** The simulation component of the program simulates the consumption of the modeled homes and provides the results to the user. This is not shown in Figure 3.
  - C. **Reporting Window:** The reporting window allows the user to analyze the results of the simulations provided by the simulation component. This can be seen in Figure 10.

The user can now start adding content to the Canvas (3) by dragging elements from the Toolbox (2) onto the Canvas or he can go through the options within the ribbon navigation (1). Initially the text on the Canvas informs the user that he can drag

scenarios onto the canvas to start modeling. If the user chooses to add content to the Canvas (3) the text on the canvas will change to inform the user of the next step, i.e. adding components. Once the user adds components to the scenario the Summary (4) will update. The summary region will contain a sum up of the content currently on the canvas, with information about their estimated yearly consumption and their overall part of the consumption of the house (Scenario).

#### 3.1. Modeling – Toolbox

In the left side of the program's main window is a toolbox containing all the elements that can be dragged onto the centered canvas. The toolbox is divided into logical categories where each category contains the elements related to it. Clicking the expander arrow of a category unfolds it and clicking it again collapses the category.

It is possible for the user to create new, delete or edit existing toolbox elements. This allows them to create exactly the elements that they use within their home and reuse them in another project. When the user wants to edit or delete elements they can do this from two places; from the modeling tab in the Ribbon and from a context menu that appear when right clicking on an element. Creating new elements can only be done from the modeling tab in the Ribbon.

When choosing to delete or edit an element through the Ribbon the users will be prompted with a simple window where they can choose the element to edit or delete. When the user chooses to delete the element from either the window above or from the context menu, the element is deleted and will be removed from the toolbox.

When choosing to edit an element the user is prompted with a new window which contains all the possible properties which can be edited on the selected element. Before saving the changes the input values are verified in order to ensure that they are of the correct format. This could be that numbers do not contain letter and so on. In Figure 4 a screenshot of the edit window can be seen, containing fields and values related to a Light Bulb element. As seen in the figure two buttons are placed in the bottom of the window. These buttons allow the user to cancel changes and close the window or save changes and close the window.

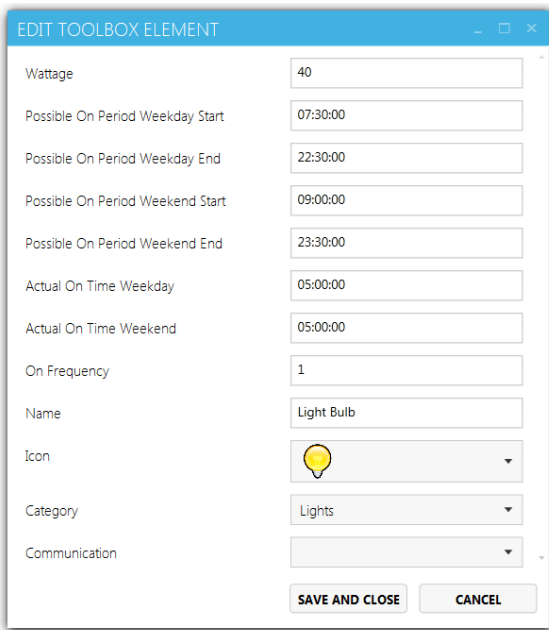


Figure 4: The Edit Toolbox Element window that allows the user to edit properties of an element in the toolbox.

The window for creating a new element is very similar to the edit window. The main difference is that when creating a new element you need to choose which type of element you want to create before the related fields will occur. When new elements are added to the toolbox it is automatically refreshed to include the new element.

### 3.2. Modeling – Canvas

The canvas is a crucial part of the modeling in the program. It is responsible for containing and presenting the result of the users modeling actions. Graphically the canvas is only a white background, but the elements which are shown upon it are represented by an icon and some small buttons used for control and information.



Figure 5: An example of how a component looks on the canvas.

In Figure 5 an example of a component (Washing machine) can be seen. The estimated yearly consumption of the component is shown in

the top right corner. The color indicates how high the consumption is; the darker red, the higher consumption. The information button in the bottom right corner shows the name of the icon upon mouse-over. The button in the bottom left corner deletes the component from the canvas and the button on the top left opens up the edit component view.

When modeling a home the components added to the home are connected via colored lines. The colors of these lines represent which category the component belongs to as seen in the toolbox.

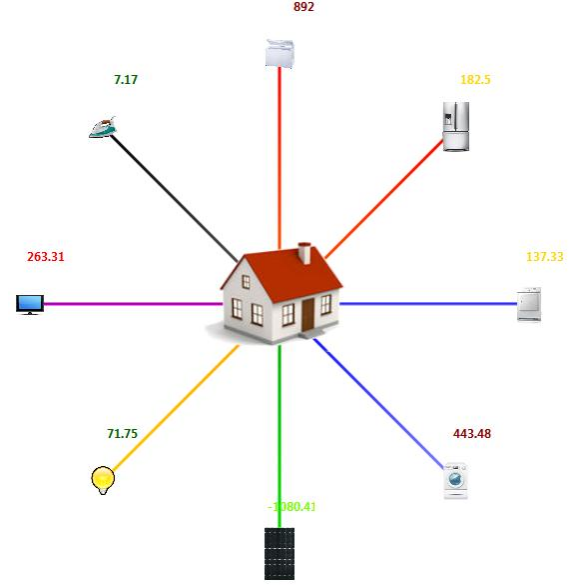


Figure 6: Example of a modeled house with eight connected components.

### 3.3. Modeling - Summary

As mentioned above a graphical summary of the estimated consumption of the scenarios on the canvas is shown in a view to the right of the canvas (see Figure 7). The summary contains a section for each scenario on the canvas in which the consumptions of the components connected to the scenario is summed up and grouped by their category. The estimated consumptions are also displayed as a pie chart along with an estimated total yearly consumption for the scenario. The color of the categories is the same as the colors used by the lines connecting the components with the home. This makes it a lot easier for the user to identify which components are members of which category such that the reasons for a high consumption in a particular category can be identified quickly.

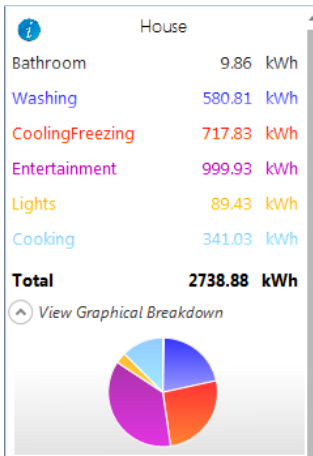


Figure 7: The summary displaying a breakdown of the consumption from two scenarios.

### 3.4. Input Data

There are currently two types of input windows in the solution, one for weather data and one for power data. It is two very similar windows in which it is possible to retrieve data from a selected data provider. The window can be accessed through the buttons from the input tab in the ribbon navigation. Below a screenshot of the “Edit Energy Prices” window can be seen.

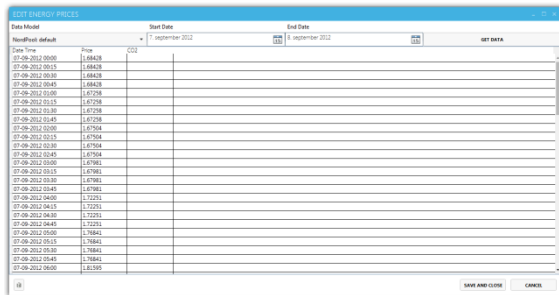


Figure 8: The window in which the user can edit the energy prices and create specific power models to be used by the simulation.

In the screenshot above you can see that in the top of the window is a drop down menu containing the providers you want to retrieve data from. Next to the drop down menu are two calendars where the user can select the start and end date for the period that they want to retrieve data for. Clicking the “Get Data” button retrieves the specified data and displays it in the window. There will in the current implementation be no data in the CO<sub>2</sub> field as there is currently no accessible source for that information.

To create custom data models the user can add or edit values in the fields by selecting the field with the mouse cursor and then typing the new value into the field. This enables the user to create their own examples of data for special events like storms or increasing oil prices.

After having edited the data the user can choose to close the window by pressing the “Cancel” button or to press “Save and Close” button to save the changed data as a custom data model. If the user chooses to save a new model, they get prompted with a dialog in which the user can choose to either override an existing model or save a new model with a chosen name.

### 3.5. Simulation

To perform a simulation the user has to navigate to the Simulation tab, which can be seen below in Figure 9. In this tab the user can either inspect previously performed simulations or perform a new simulation, as described in this section. Before starting a new simulation the user has to consider the values in the five settings controls to the left of the “Run Simulation”-button.

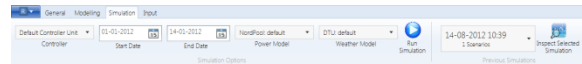


Figure 9: The simulation tab of the ribbon navigation menu.

As seen in the figure above the users starts off by choosing which type of controller they want to use for their scheduling. The application supports having multiple different controllers using different simulation strategies to choose from. Then the users need to set the time span that they want to run the simulation over, by setting a start and end date. Finally it is possible for the users to select which power and weather data they want to use for the simulation. Here it is possible to select the earlier created input models (Weather and Power), such that they can simulate a user constructed setup. To run the simulation the user presses the “Run Simulation”-button and then waits for the program to notify when the simulation is finished. While waiting, the user can see the progress of the simulation in a window that opens when the simulation starts.

### 3.6. Reporting

Based on every simulation performed, one or more reports can be made. A report consists of an overview of the simulation results along with a number of charts. The reporting window lets the user see and edit the reports of a simulation. The reporting window can be found either by clicking on the “Show Report” button when a simulation is finished or by choosing an older simulation in a drop down menu in the ribbon and clicking on the “Inspect Selected Simulation” button. The reporting window can be seen in Figure 10.

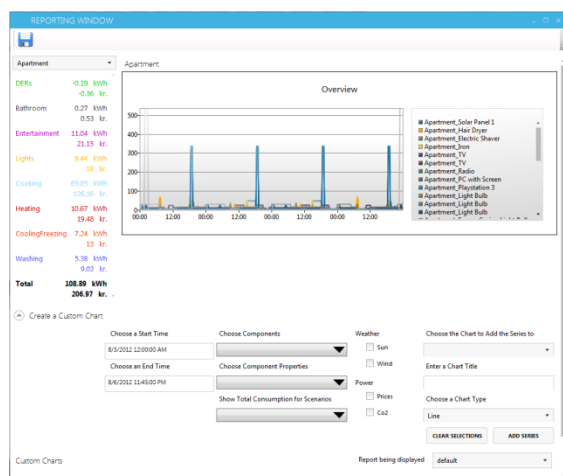


Figure 10: The reporting window used to analyze the results of a simulation.

At the top left is a drop down menu saying “Apartment” in Figure 10. This menu lets the user choose which home in the simulation result to show the overview for (in this screenshot the name of the chosen home is “Apartment”). The chart called “Overview” shows the consumption pattern of each of the components of that home in the simulated interval. The x-axis shows the time and the y-axis shows the consumption at the given time in Watt-hours. To the left of the overview chart is a view very similar to the summary. This area works similarly, but instead of showing the estimated yearly consumption it shows the consumption in the simulated interval for each category as well as the total price for the power used by each category. These values are from the actual simulation rather than estimations and should therefore be more accurate than the ones found in the summary in the main window. These categories are also shown in the specific color of the category as in the summary. Upon mouse-over of each category an estimation of the yearly consumption, price and CO<sub>2</sub> emission can

be seen, calculated by scaling up the simulated results to match a year. Therefore the longer the simulation period is, the more accurate these results become (see Figure 11).

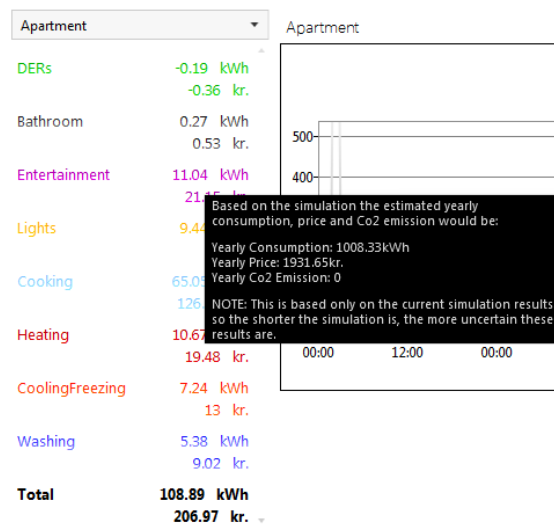


Figure 11: A screenshot showing the tooltip displayed when hovering the mouse of a category in the overview.

Below the overview is a region called “Create a Custom Chart”. It provides the user with controls to create charts based on the simulation results. The charts can be generated from a wide range of parameters. The first column of controls lets the user choose the start and end time of the data for the new chart. By default this is the first and last data entry of the simulation result. The second column lets the user select which data series to add from the model. In the first drop down the user can select which components to include in the chart. The second drop down holds the properties to be displayed for the selected components such as price, CO<sub>2</sub> emission, consumption or on-time intervals. The last drop down lets the user add a data series to the chart that shows the total consumption of the homes selected in this drop down. All three drop down menus allow for multiple elements to be chosen at the same time.

In the third column the user can choose which input data to display in the chart such as wind speeds, power prices etc. By adding these it becomes easy to see how the components have been scheduled compared to the fluctuating prices and weather. The final column specifies information about the chart that should hold the series specified in the other columns. The first drop down lets the user add the series to an existing chart or to choose to add them to a new chart. If “New Chart” is chosen (i.e. leaving the value empty) the user can

specify a title and a chart type (line, bar, pie etc.) for the chart. Clicking the “Add Series” button creates the series and/or chart specified and adds them to the “Custom Charts” region.

Figure 13 shows two examples of charts that have been created. The first one contains series showing the prices over time for the refrigerator and washing machine in the home. The second chart shows the fluctuating power prices in the same period. It is important to note that all types of series (consumption, prices, wind speeds etc.) can be combined in the same charts. This creates a very strong flexibility when creating charts, but also means that the y-axis of the custom charts is without unit as it can at the same time be currency, Watt-hours, m/s and more. Additionally, some series, such as consumption of washing machines, will show very large values whereas some series, such as power prices, will show very low values, making these series impractical to have in the same chart. Therefore it can be a good idea to separate them into two charts and compare these as done in Figure 13.

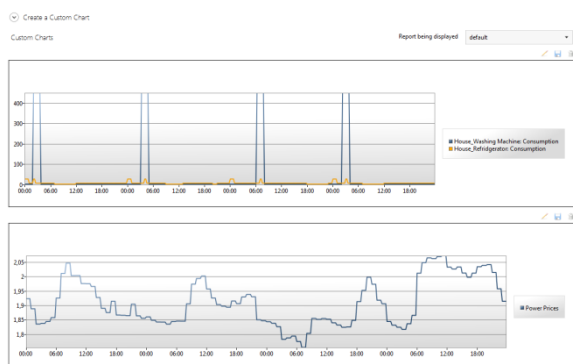


Figure 13: A selection of the reporting window showing the Custom Charts section in which the charts created by the user are displayed.

In the upper right corner of each chart are three buttons: Edit, save and delete. The delete button removes the chart from the report. The edit button opens an *Edit Chart* window (see Figure 12) in which the user can change the name and type of the chart as well as remove series from the chart by clicking on the trash can next to the series.

When a report has been created with the desired charts it can be saved using the save icon in the upper left corner of the reporting window. This lets the user choose a name for the report and when it has been saved the user can toggle between which report to show, using the drop down menu just above the region with the custom charts. Since reports are connected to a single simulation result only reports connected to the same simulation result

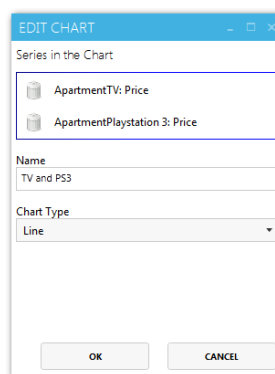


Figure 12: The view in which the user can edit an existing chart.

can be displayed this way. If reports connected to another simulation result should be displayed the reporting window must be closed and the desired simulation result should be found in the drop down menu in the ribbon using “Inspect Selected

Simulation” as explained earlier in this section. It is, however, possible to have two reporting windows open at the same time showing different simulation results if comparison across simulations is required.

## 4 CONCLUSION

As mentioned in the beginning of the paper the Danish workgroup, Smart Grid Netværket, suggested that the knowledge of the smart grid should be increased at consumer level in order to increase the acceptance of the smart grid. This solution helps reach this goal by allowing consumers to model their own home and see how their energy consumption would change if the appliances were smart grid enabled and the house used fluctuating energy prices rather than flat rate prices.

An analysis of the state-of-the-art simulation tools showed that very little work has been done in the area and that the existing tools are mostly non-graphical and therefore not very easy to use. Additionally they do not provide much opportunity to model specific homes and the incorporation of real energy and weather data is non-existent. Lastly the graphical analysis and comparison of different simulations were not present in any significant extent in any of the tools. These missing features provided the foundation for the developed simulation tool.

Additionally, a number of other areas have been identified in which the application would prove valuable. This includes salesmen that wish to illustrate the financial benefits of their product (see Figure 2) and utility companies that wish to measure the flexibility of the homes connected to the grid, as identified during a meeting with a leading manufacturer of smart home controllers in Denmark.

The application has multiple uses. One of them is to answer the question: “How will the smart



grid affect my energy bill?”. This question can be answered by creating a model of the home and running a simulation using no smart grid enabled components and a flat rate power model and running a simulation using smart grid enabled components and fluctuating energy prices. This has been done in the case studies and the results show that in one of the scenarios the consumer can save €78 per year by changing from flat rate to fluctuating energy prices and a total of €148 per year if the components are also smart grid enabled. This should be enough motivation for most users to participate in the smart grid [12].

All in all the application fulfill multiple target groups’ requirements in the area of smart grids that are all currently unmet.

## 5 FUTURE DEVELOPMENT

The goal was to create the application such that it would be simple and easy to develop further later on. Therefore one of the main focus areas of the application was extensibility. The application was therefore designed such that the functionality was split into logical groups of components which were then loosely coupled with as simple communication between them as possible. A list of the most significant possibilities for extensions is shown below.

- Integration with other simulation tools, such as GridLab-D, to include other aspects of consumptions for heating of homes.
- Including floor plans of the modeled household could give more precise heating calculations.
- Provide a list of components based on the simulations that would provide lower energy consumptions than the components in the home.
- Dependencies between components, such that the scheduling between them can be synchronized when simulating.
- Historic consumption data from the user could be integrated to provide knowledge of usage patterns and thereby provide even more accurate results.
- Web based deployment such that the application can use more computational power and the users can get access to the application through a web browser.

## REFERENCES

1. Klima- Energi- og Bygningsministeriet, *Smart Grid Netværkets Arbejde - Sammenfatning og anbefalinger*, 2011.
2. Haas, K. *Peaking Power Plants*. 2009 4/11 [cited 2012 13/2]; Available from: <http://www.energyandcapital.com/articles/peaker-plants-bridge-the-gap/991>.
3. Illinois Environmental Protection Agency. *Peaker Power Plant Fact Sheet*. 2011 [cited 2012 13/2]; Available from: <http://www.epa.state.il.us/air/fact-sheets/peaker-power-plant.html>.
4. Nozik, K. *Eastern Power Outage Unfortunate but Entirely Predictable*. [www.frugalmarketing.com](http://www.frugalmarketing.com) n.d. [cited 2012 25/8]; Available from: <http://www.frugalmarketing.com/dtb/electricity.shtml>.
5. Control4. *Introducing the Control4 EMS 100*. 2012 [cited 2012 24/2]; Available from: <http://control4.com/energy/products/>.
6. Tendril. *Tendril Connect*. 2012 [cited 2012 24/2]; Available from: <http://www.tendrilinc.com/platform/connect/>.
7. California Energy Commission and Charles River Associates, *Statewide Pricing Pilot: Summer 2003 Impact Analysis*, 2004, Charles River Associates.
8. GridLAB-D. *GridLab-D Simulation Software*. 2011 [cited 2012 22/2]; Available from: <http://www.gridlabd.org/>.
9. EnergyPlus, *Getting Started with EnergyPlus*, in *Basic Concepts Manual - Essential Information You Need about Running EnergyPlus* 2011, US Department of Energy.
10. PowerMatcher Smartgrid Technology. *Basic Structure and Agent Roles*. n.d. [cited 2012 6/6]; Available from: <http://www.powermatcher.net/get-technical/powermatcher-technology/basic-structure-and-agent-roles/>.
11. PowerMatcher Smartgrid Technology, *PowerMatcher Simulation Tool* 2010, Energy Research Centre of the Netherlands.
12. Konkurrencestyrelsen, *Konkurrencen på detailmarkedet for el*. 2009.