UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

HUMAN-IST
HUMAN CENTERED INTERACTION SCIENCE & TECHNOLOGY

Interfacing Smart Homes to their local Smart Grid

Nina Eldridge[1]

Supervisors:

Moreno Colombo, Dr. Luis Terán, and Prof. Dr. Edy Portmann

November 10, 2021

Department of Informatics - Bachelor Project Report

Département d'Informatique - Departement für Informatik • Université de Fribourg - Universität Freiburg • Boulevard de Pérolles 90 • 1700 Fribourg • Switzerland

phone +41 (26) 300 84 65 fax +41 (26) 300 97 31 Diuf-secr-pe@unifr.ch http://diuf.unifr.ch

[1]nina.eldridge@unifr.ch, University of Fribourg

Abstract

In this project a smart grid and smart home are implemented. The focus lies on the aspects that have to be considered when programming a smart grid and on the graphical user interface of the smart home. The program simulates appliances belonging to a house, which are then scheduled for a certain time. The available energy for a certain time is calculated by considering the cloudiness and the resulting energy production of solar panels. This project can serve as a basis for more complex and more realistic smart homes and can easily be expanded.

Keywords: Bachelor project report, Human-IST Research Institute

Contents

List of Figures

1 Introduction

Smart grids and smarts homes are heard of more and more and are a fast developing technology. In the age of digitalisation all kinds of processes are tried to be optimised with the help of computer programs. Such optimised, digitalised and automated processes are called smart.
Grids are being optimised by computers to integrate the new forms of energy productions. The goal is to allocate the energy more efficiently, using programs that predict the energy production, but also know which appliances have to run. This is then the incorporation of smart homes into the smart grid. Smart homes connect appliances of a home to the local smart grid. The house owner can schedule appliances, telling the smart home until when they have to run. This way the smart grid can allocate energy to different houses when needed.
It is predicted that in the future such grids will be necessary, to manage the transition from fossil fuel to renewable energy. The problem with renewable energy is that it is unpredictable and dependant on the weather. Therefore good smart grids and smart homes will be needed, to use the energy when it is there and to allocate it to the right places.
In this thesis the goal was to create an interface for house owners, such that they could schedule appliances. Then those appliances get scheduled by the smart home according to the weather prediction. The other focus lies on the predictions for the energy supply, depending on a renewable energy source. The fastest growing renewable energy source are solar panels, so the thesis is based on their energy output. Depending on the available energy, appliances that are added by the house owner should get scheduled for a time where they use as much renewable energy as possible.

Three research questions guided the thesis:

- How much energy does a solar panel produce in relation to the weather and time of the year and day?
- To make the grid more power efficient, it is important to make a good prediction of the amount of energy in the grid during the day. How accurately can the energy be predicted considering all different actors connected to the grid?
- How should an interface for the smart grid be programmed to make it as user-friendly as possible?

To answer the questions research was done on solar panels, smart grids and smart homes. The energy production of solar panels would build the basis for all the calculations in the program, so it was important to have a mathematical basis on how solar panels produce the most energy. In what forms smart grids and smart homes already exist, what has to be considered when creating them and what could still be improved were other important aspects to the thesis.
As for every new type of technology, it will only be used if it isn't too difficult for consumers to understand. That's why the focus lies on the graphical user interface, such that people would actually use a smart home if they could. To test the user-friendliness, some prototypes were given to testers, which would give a feedback on what could be improved.
In the first chapters the theory underlying the thesis is described. Then the problem which the program should solve is stated and the methods used for that. The concrete implementation of the smart grid and smart home are described in the last chapter, where programming decisions are discussed. The code accompanying this thesis can be downloaded from github following the link in the footnote[1]. Before deploying the code, the README file should be read carefully.

[1]https://github.com/ninaeld/Smart-Home

2 Theory

This section is comprised of the theoretical parts of the project. Namely how a solar panel works, how much energy it can produce and on which factors that depends. After that smart grids and smart homes are discussed, to what extend they already exist and what they could mean for the future.

2.1 Solar Panel

A solar panel is made up of cells, which convert sunlight into electrical energy. How much of the energy a solar panel can produce, depends on different aspects. One of the most important is the location of a solar panel. Sunlight hits the earths surface in a more perpendicular angle nearer to the equator than at the poles. This results in more sunlight energy per square meter, which means there is also more energy for a solar panel to use. But also locally there can be great differences, depending on the slope of the location and whether the solar panels are located in a valley or on a hill.
The next important this is the orientation of the solar panel towards the sun. On the northern hemisphere a solar panel should always be orientated towards the south, in the southern hemisphere towards the north. Not only the orientation, but also the angle at which the solar panel is set up makes a difference on how the sunlight hits the solar panel. Since the position varies over the day and the year, the most efficient way to set up a solar panel is to have rotating axes, such that the solar panel can align itself optimally with the sun. Unfortunately that is quite expensive and solar panels are mostly installed statically on roofs.
The part that could still be improved the most is the efficiency of solar panels. At the time the efficiency for the best available solar panels lies at about 23% [15]. Considering that other energy sources can be converted into electric power with an efficiency up to 90% as with hydro power, solar panels are lacking far behind. By increasing the efficiency of solar panels the energy production could be increased without having to produce a greater amount of solar panels and without using more space to place them.
Of all of the renewable energy sources solar energy is one of the most promising. This is largely due to the fact that the sun emits enough energy in one day to meet the worlds energy supply of a whole year. The annual production of solar panels is increasing each year, which can be seen in figure 1. This is a sign of how the industry is focusing on solar energy and trying to build ever better solar panels. However the problem with solar energy is that it is mostly produced at times of low energy demand. On a daily basis energy is mostly used in the evenings, when the sun doesn't shine anymore. On a yearly basis more energy is needed in the winter months as the energy demands for heating goes up. At the same time the sun has a lower stand and a shorter trajectory over the sky, which results in less energy production. To bridge those differences, it would be important to store energy for later purposes.

These days energy is often stored in dams by pumping water up into the dam at times when there is an energy surplus. When the energy is needed, for example in the evenings, the water is let through the turbines and produces the energy that was stored. As stated before, since dams have an efficiency of up to 90%, this is a very efficient way to store the energy. Other options, such as storing the energy in batteries, has a higher loss rate and is therefore inefficient. A lot of research is going into other ways of storing the energy while trying to keep the loss to a minimum. Once there are more efficient ways of storing energy, smart grids will have to incorporate calculating whether storing the energy or directly use it is more efficient.

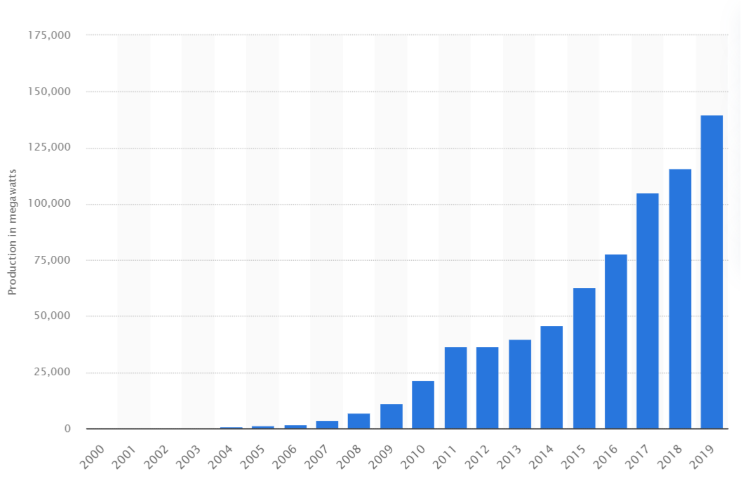


Figure 1: Global annual solar module production from 2000 to 2019

2.2 Smart Grid

As described in section 2.1, the biggest problem with solar energy is that it is unpredictable and there is mostly more of it at times where it is less used. That's where smart grids can help. Their goal is to divert produced energy in such a way that it is directly used. This helps to keep energy loss as low as possible.
A smart grid is an electrical networks that combines digitalisation with electrical power grids. It supports a two way communication system between the power suppliers and the consumers, where it uses digital automation to control, monitor and analyse the supply chain. Managing many different sources of power generation from distributed locations is the main job of a smart grid. [17]
The following features set a smart grid apart from a conventional grid. Communication between each component of the grid is fully automated. At every point in the grid the power supply is controlled automatically. The grid provides data to the service provider, which is used to detect anomalies and faults automatically. The software doing the work, the measurements and the sensors are checked regularly [17]. These features lead to many advantages of a smart grid. The power management of a smart grid is more efficient, it has a better supply and demand management and is equipped with better security. Additionally the cost of operation, maintenance and management is lowered. Because it takes renewable energy more into account, it is also CO_2 friendlier than conventional grids [17].
Most energy grids are still built in a centralised way, where there is only a one way flow of energy from the power plant to the consumer. In the last few decades more and more consumers have also become small suppliers by connecting their solar panels to the local grids and feeding spare energy into it. When there are different energy sources feeding into the same grid, the best way to manage them is with a smart grid. The smart grid diverts energy coming from different locations to the nearest place where it can be used.
A few cities world wide are starting to build smart grids to supply their city with energy. The very first city to have done so was Austin, Texas, in 2009. Austin's smart grid is also one of the only ones wholly financed by the city itself. To built it Austin had to invest $150 million for 410'000 smart meters, 86'000 smart thermostats, 2'500 sensor and 3'000 computers, servers and other components [2]. With the smart grid Austin can maintain to be one of the largest cleantech centre worldwide. Austin has ambitous environmental goals, which they try to reach in part with their smart grid.
When the planning of the smart grid started, the main motivation was to save energy and costs through more efficient transmissions, reduced operational costs and reduced peak loads. However the newer vision of their smart grid is to transform the electricity production, moving from a very centralised and inefficient system to a system where production is decentralised and green. [2]

In Switzerland the annual percentage of renewable energy in the electrical grid is growing continuously. As can be seen in figure 2, the percentage has grown from about 15% in 1990 to almost 25% in 2019 [1]. To incorporate these new forms of energy production, also Switzerland will have to make use of digitised and automated electrical grids. The Swiss Federal Office of Energy recognises that there is a need to restructure the current electrical grids. On the one hand because decentralised electricity production is growing and on the other hand because it is important to use energy more efficiently [5]. There aren't any concrete implementations of smart grids in Switzerland yet, but the SFOE has drawn up a schedule for developing such grids.

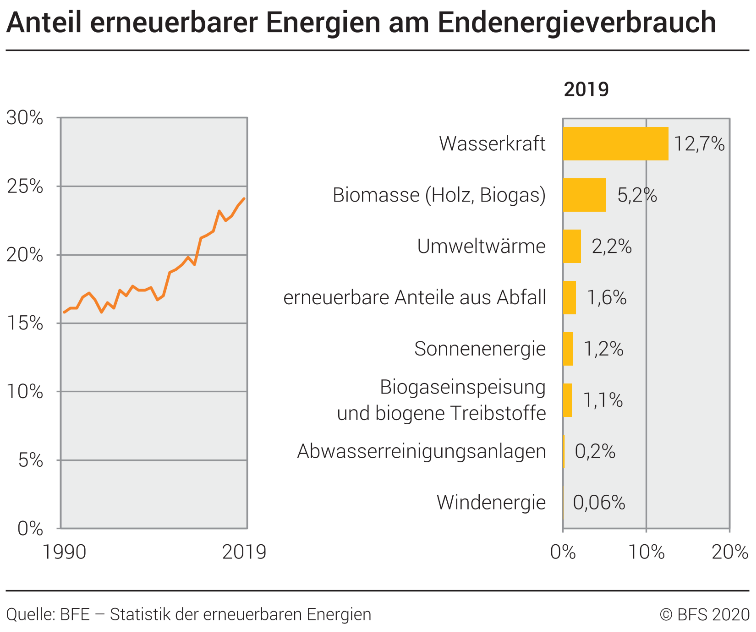


Figure 2: Share of renewable energies in final energy consumption in Switzerland

2.3 Smart Home

The word smart home can entail many different aspects. However the important thing is that it means that managing appliances and electronics of a house can be done via the electrical grid. Most research is focusing on the aspect of being able to control electronics remotely, such as turning the heating on before coming home. But for this research the focus lies on the ability to tell the smart home program until when a certain appliance should be done, but the program can chose when to let it run.
Many apps for smart homes are already on the market and different appliances can be bought which can be added to the smart home [4]. The worldwide trend is to make more appliances that can be controlled remotely and that are connected to one another in the intranet of the house. Because all appliances become connected to the internet, that can be used by a smart grid to schedule them. A house owner can tell the smart home until when an appliance should run. The smart grid calculates which time might be ideal to use as much renewable energy as possible.
Smart grids and smart homes together will make the energy usage much more efficient and less energy will be lost by storing it.

3 Problem statement

A worldwide trend is to switch from fossil fuels to renewable energy. On the one hand because of climate change, since fossil fuels emit CO_2 which is a green house gas and causes our atmosphere to heat up. On the other hand our fossil fuel reserves aren't infinite and at the current rate we are using them, they will be empty by the end of the century [11].
There is no choice than to produce more energy from renewable energy sources such as wind, solar, water, geothermal energy, bio gas etc. There are two problems they bring to the table, namely that they are to a certain degree unpredictable and that there aren't very efficient ways to store them yet [16]. Until now our grids were powered by energy sources, which can be regulated, such as nuclear power plants. When energy demand is high, the energy output is driven up, when demand is low, the output also gets lowered. Grids are also very centralised, having a few plants supplying energy for a big area over great distances. But every transport of energy also entails some loss of energy.
When switching to renewable energy sources, grids will have to change in two fundamental ways. First of all there will be more local grids, such that energy doesn't have to be transported over longer distances. Secondly those grids will be smart grids, since it will be important to directly use the energy produced, since the energy production can't be regulated.
Another aspect to use the produced energy more efficiently, is to add smart homes to the smart grid. Smart homes try to predict the energy production and then let appliances run at a time of high energy production.
In this thesis different aspects of the smart grids and smart homes were tried to be solved. On the one hand how the energy production could be fairly accurately predicted by using the weather forecast. On the other hand how the produced energy could be grouped together into local groups and then shared between one another. And as a final aspect, how a scheduling algorithm should be written, such that appliances run on renewable energy.
Having the program isn't enough, it also has to be used by consumers, in this case house owners. Therefore a graphical user interface completes the thesis. Without a graphical user interface the program can't be used. The aim was to create one that is clearly structured, displays all necessary information and is user-friendly.

4 Methods

The approach was taken to solve each problem separately and then combine them to work together as one program. Since the energy prediction constitutes the basis of the whole thesis, how to calculate an accurate prediction was tackled first. The first thought was to retrieve collected data from solar panels, on how much energy they produce depending on different factors. But finding clean data for different locations, different parameters and covering a whole year proved to be very difficult. Instead of basing the energy prediction on collected data, the decision was made to approximate the energy production with calculations. A lot of thought went into which parameters should be taken into account for the calculations. So the focus was laid on the biggest factors that influence the energy production, namely the location of the solar panel, the angle relative to the earths surface at which it is tilted, its efficiency and the time and date to calculate the angle of inclination of the sun.
What influences the energy production on a daily level is the weather. How to get the weather prediction was the next thing to think about. The main factor concerning the weather is the cloudiness, although wind and rain can alter the energy production of a solar panel, for an approximation these two factors could be omitted. So the weather predictions should be retrieved from a weather station the cloudiness would be used to calculate how much energy from the sun actually hits the earths surface.
A lot of research went into scheduling algorithms, to find a good way of scheduling the appliances. The thought was on what to prioritise, whether on the duration of the appliances or on their end time or some other factor. With electrical grids an important aspect is that an energy deficit can be compensated with non renewable energy. This is done this way until there are efficient ways to store energy and an energy deficit can be compensated with stored energy. For the scheduling this aspect is very important, because that means an appliance can also be scheduled over a timespan, where the energy supply for a certain hour is too low. But if the overall energy usage is higher than for other timespans, it is still the most efficient choice. If the energy production form the renewable energy were a strict boundary, this kind of scheduling wouldn't be possible. Because the energy supply isn't a strict boundary, it doesn't make a difference in which order appliances are scheduled. The appliance is scheduled at a time where the maximal amount of energy is left in the grid. The schedules will turn out the same no matter in which order the appliances are scheduled.
How to design the graphical user interface was done in two steps. In a first step all functionalities the GUI should have were written down. This way there was an overview on what the GUI should be able to do and the functionalities were grouped. As a second step the layout of the GUI was sketched, such that the programming could follow the sketch as a template. When a working prototype of the GUI had been created, it was given to some testers to get a feedback on what could be improved. This way the the GUI could be made more user-friendly through the input of real users.

5 Implementation

In this chapter the concrete implementation of the smart grid and smart home are described.
To make the program as organised as possible, it was important to make it modular. That way the program consists of different parts that interact with each other and if something has to be changed, it can be changed in one module without influencing the others.
For each component of a smart home a class was created. First of all a class House, that would store the information about a certain house. Then separately a class for the solar panels, where each house would have an attribute solar panel, which would be an instance of the class Solar Panel. Most importantly an Appliance class was necessary to create appliances that would belong to a certain house.

5.1 Structural Set-up

In order to create a smart grid, houses with different energy productions need to be grouped together to exchange energy. This was done by creating towns and each house belongs to one town. Energy exchange only happens within these towns, since a smart grid tries to divert energy to the nearest possible usage, which means having local grids. A town is an instance of the Town class, which stores the town name and a list with all the houses that belong to it.
Another reason that it is important to group houses into towns, is that the energy production from the solar panels depends on the location of a house. The solar irradiation and the weather are different for different locations. This influences the prediction of how much energy should be available for a certain town, which will be shown in section 5.2.
The next step was to make sure, that when an instance of a class is created the input for the attributes is correct. It is important that the types and the values match what the program expects or there will be errors when working with the instances. For example the efficiency of a solar panel has to be a decimal number between 0 and 1, since this is directly used in the calculations on how much energy the solar panel produces (see figure 10). To guarantee for certain values, getters and setters were written for each attribute of a class. This was done with the @property decorator. As can be seen in figure 3 the decorator makes sure that when a new value is assigned to the efficiency attribute, it is checked whether it has the right type and the right value. If not, it either raises a TypeError or a ValueError. These errors are then caught by the graphical user interface, which will be discussed in section 5.4.
The advantage of getter and setters with the @property decorator, is that attributes can be accessed directly via their names, e.g. solarPanel.efficiency= 0.2, but the value is still checked and will raise an error if the conditions aren't met. This omits the necessity of getter and setter methods, with which the values of attributes would have to be changed.

```python
@property
def efficiency(self):
    return self._efficiency
@efficiency.setter
def efficiency(self, value):
    self.lock.acquire()
    #make sure efficiency is a float and between 0 and 1
    if not isinstance(value, float):
        self.lock.release()
        raise TypeError("Efficiency is of wrong type")
    if (value < 0 or value > 1):
        self.lock.release()
        raise ValueError("Efficiency has to be between 0 and 1")
    self._efficiency = value
    self.lock.release()
```

Figure 3: Getter and setter of the efficiency attribute with the @property decorator

The last part that had to be done before the actual calculations could be programmed, was in which way all instances should be stored, such that each instance knows to which other instance it belongs to. Each town has a list, in which each house belonging to it is stored. This is achieved by appending the house to the list within the house constructor. The town is given to the House constructor as an argument and at the end the method appendSelfToTown() is called, which takes the town and appends the house to that town's list. The function call is the last line in figure 4. This approach is also used for the appliances, where each created appliance is appended to the appliance list of the house it belongs to.

```python
class House:
    def __init__(self, name, town, solarPanel):
        ...
        self.appendSelfToTown()
```

Figure 4: Append a house to the list of its town

To access all the lists in the most efficient way, is to store all towns in a global list called Towns. This global list can then be accessed from all modules. This way it is possible to access and change values in houses and appliances, without having to copy values and reassign new values. With the global list of all towns it is also possible to access every created instance over just one variable. To create a global list, the list is instantiated with the keyword global in a separate file called config.py. The config.py file has to be imported into every other file and then the list can be accessed via the file name from everywhere in the program.
A programming decision that was made, is that new towns cannot be created by the user. Whether a new town has a own smart grid isn't something a user can influence, that has to be done by the town authorities and therefore has to be added to the program by a maintainer. It is a sector to which a user shouldn't have any access to. In this program five towns are predefined, namely Basel, Bern, Binningen, Lausanne and Lugano. If any more towns should be added, that has to be done by the programmer in the Main.py file. It is important to remark that a new town must have the same spelling as the town's name on the website Prevision meteo [9]. Because the name of the town is used to retrieve the weather data for that town, which is shown in figure 5 and discussed in section 5.2.1. Therefore the town name can't have any mistakes or the request will throw an error.
Since this program is only a simulation of a smart grid and smart home, there is no external storage for all changes made while running the program. Everything is stored locally on the running computer and as soon as the program terminates all data is lost. For a first prototype this approach is acceptable. But as a next step in the development of the program the use of an external database would allow to store the data for a longer time. This would be necessary if it were to be used commercially.

5.2 Energy Calculation

After having created the classes to simulate towns, houses and appliances, the first part that was tackled was how to calculate the energy prediction for a certain town at a certain time of the year. For this a class Calculations was created, with one static method called get_predictions(). Static methods makes it possible to call a method via the class name, therefore it is not necessary to have an instance of the class Calculations. The get_predictions method consists of two parts, where firstly the cloud predictions is calculated and from that the resulting energy prediction while considering the irradiation of the sun and the set up of the solar panels.

5.2.1 Cloud prediction

The energy prediction should consider the cloudiness of that coming days, since cloudiness strongly influences solar irradiation, which is in turn the main factor impacting the solar panels' energy output [13]. For this reason, in this section, the estimation of cloudiness at any time during the forecast period is handled. The proposed solution to predict cloud coverage in the coming hours, is using the weather data from a weather forecasting service, as these are as accurate as it gets. The chosen service was prevision-meteo[2], as it provide a REST API interface to easily access accurate weather forecast data. The python module requests handles requests to websites to retrieve data [6]. In figure 5 the procedure of a request is shown. The link to the website has to be copied to the program. At the end of the link, the town for which the data should be gotten is specified. Since the calculations are specific for each town, the town for which the predictions should be calculated is given as an argument to the calculation method. The town's name gets converted to a string and gets concatenated to the link. Then the requests module takes the link and gets the specified data from the prevision meteo website [9]. The data is recuperated in JSON format and is stored in the response variable, which can then be used for the calculations. In the Main file, each town is passed to the calculation method once every hour.

```
link = 'https://prevision-meteo.ch/services/json/' + str(town)
#request the weather website for the prediction
response = requests.get(link)
```

Figure 5: Requesting the weather prevision

On the prevision meteo website there is a user manual on how to recuperate the data from the JSON file [9]. To predict the cloudiness over the next few days, only three parameters are of importance, namely the low, middle and high altitude cloudiness. Since the main focus of the project is on the interaction of the user and the smart home, one can simplify the problem and assume that the three cloud layers can be summed and act as a linear filter. This means that if the total cloudiness is X%, then the total solar irradiation is reduced by X% with respect to a clear sky condition by the cloud layers. Therefore the overall cloudiness for a certain hour is defined as the average of the three values. The values for cloudiness range from 0, no clouds at all, to 100, which is 100% cloudy for each layer. The three values are taken from the JSON file, have to be cast into a float and are then added up and divided by three. In figure 6 these steps are shown in line 5 to 12, where the value also gets rounded. It is good practice to round the value, because in the next step the value is transformed from percentage into a decimal value. For the decimal value it is enough to have two decimal places. To conversion of percentage into decimal is important for later calculations as shown in figure 9.

The calculations done in figure 6 result in the average cloudiness value over one hour. To calculate the cloudiness over the next few days this process has to be repeated for each hour in the next few days. This is what the for loop is for. It starts at the current hour and ends at 24. This way the values of the next hours of the current day are calculated. To get the values of the next

[2]https://prevision-meteo.ch/

```python
for x in range(next_hour ,24):
    hour = str(x) + 'H00'
    # gets the values out of the json file of the request
    # data in a range of 0 to 100
    value_high = response.json()['fcst_day_0']['hourly_data'][hour]['HCDC']
    ...
    # cast the values from a string to a float
    HCDC = float(value_high)
    ...
    # take the average value to get the average coverage of the sky
    average_value = (HCDC + MCDC + LCDC) / 3
    rounded_value = round(average_value , 0)
    #convert it to a decimal number, cause we need it later for the energy
        prediction
    decimal_value = rounded_value / 100
    # append the values into a list
    cloud_p.append(decimal_value)
```

Figure 6: Calculation of cloudiness

day a new for loop is started, where 'fcst_day_0' is incremented to 'fcst_day_1'.
Because weather predictions get less precise the further in the future they are it doesn't make much sense to try to predict the energy production too far into the future. Also when scheduling an appliance, they normally should run within the next 1 or 2 days, because most users want their appliances to be done soon. That's why the decision was made to limit the predictions to 48 hours from the current time. After having calculated the cloudiness for the rest of the day and the next day, the cloudiness for the second day is only computed until the current hour of the current day. The for loops in figure 7 are responsible for the estimation of the cloudiness for the next 48 hours.

```python
#second loop to get all 24h of the first day
for y in range(24):
...
#third loop to get the remaining hours adding up to 48h
for z in range(next_hour):
...
```

Figure 7: For loops to get the cloud prediction over the next 48h

The variable next_hour in figure 7 stores the next full hour from the current time. This is used in the first for loop in figure 6 to know at which hour to start the calculations. Then it is used again in the third for loop to know at which hour to stop the calculations.
After having calculated the cloudiness for a certain hour on a certain day, the value is appended to the cloud_p list, which is the last line in figure 6. The cloud_p list stores the cloudiness for each hour over the next 48 hours. This is important since the cloudiness will influence the energy prediction and will be used for the calculations in section 5.2.2.

5.2.2 Energy prediction

After having calculated the cloudiness, the next step is to calculate the irradiation only depending on the location, date and time of the day. The solar irradiation is power per square metre, therefore Watts/m^2. The sun emits an approximation of 1000 W/m^2 for clear sky conditions, if it hits the surface in a perpendicular angle [3]. This value gets smaller for surfaces that have a less steep angle to the sun. To calculate the real value of irradiation from the sun, depending on the date, time and the position of the earth in respect to the sun, python has the module Pysolar. With the Pysolar module it is possible to calculate the irradiation of the sun for each point on the earth at any given date and time. This module is used to calculate the irradiation for the towns.

```python
lat = response.json()['city_info']['latitude']
long = response.json()['city_info']['longitude']
```

Figure 8: Latitude and longitude of a town

From the request of the prevision meteo website it is also possible to get other information than just the weather prediction. Figure 8 shows how to extract the latitude and longitude from the JSON file. With this information it is possible to calculate the irradiation for that location. The Pysolar module has two methods that are needed, namely solar.get_altitude() and radiation.get_radiation_direct() [12].
With the solar.get_altitude() the altitude of the sun over a certain location for a certain time and date is calculated. The method takes as arguments the latitude, longitude and the date and time and returns the altitude in degrees. Then the function radiation.get_radiation_direct() takes the date and time and the calculated altitude and returns the irradiation in W/m^2 on a surface that is perpendicular to the sun rays at the given coordinates [8]. This has to be done for each of the next 48 hours and for each town separately. The radiation method returns the irradiation for the earth surface. This value is then appended to a list called houseEnergy, that stores the energy production of a house for each hour. That is a local list, to store the energy production of each house. For simplicity only the direct beam radiation of the sun is considered in the calculations. The diffuse and reflected sunlight is omitted.
To have a real prediction of the energy production the cloudiness that was calculated in the last step has to be taken into account. The function to normalise the total radiation is given by: $radiation_{total} = radiation_{cloudy} + (1 - cloudiness) * (radiation_{sunny} - radiation_{cloudy})$. The $radiation_{sunny}$ is the value calculated with Pysolar. The cloudiness was calculated before and stored in a list for each of the next 48 hours. This formula it is also the reason, why the cloudiness was converted into a decimal number (see figure 6). Because of the normalisation the cloudiness is subtracted from the value 1 and therefore it is necessary that the cloudiness is a decimal number. As an approximation the $radiation_{cloudy}$ for maximal cloudiness can be considered of going towards 0 W/m^2 since hardly any sunlight hits the earths surface. Therefore $radiation_{cloudy}$ can be omitted in the calculation for the total radiation. What is left is: $radiation_{total} = (1 - cloudiness) * (radiation_{sunny})$. Figure 9 shows how this formula is used and that the irradiation of the houseEnergy list is updated after having considered the cloudiness. After all these steps the houseEnergy list contains the prediction of the irradiation for the next 48 hours for a certain town and under consideration of the cloudiness.

```python
for i in range(48):
    real_irradiation = (1 - cloud_p[i]) * houseEnergy[i]
    rounded_irradiation = round(real_irradiation, 2)
    houseEnergy[i] = rounded_irradiation
```

Figure 9: Irradiation in consideration of the cloudiness

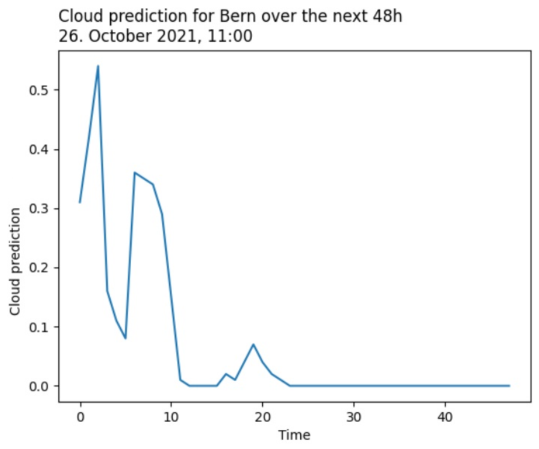
As a last step the energy output of the house is given by the area of the solar panel and its efficiency. The area of the solar panel isn't perpendicular to the sun throughout the day, so the effective area that is hit by the sunlight is actually smaller. To calculate this the angle at which the solar panel is tilted and the orientation of it are necessary.
To calculate the effective area of the solar panel that is hit by sunlight, the angle at which the solar panel is tilted and the angle of the sun is needed. The angle of the sun can be gotten with the get_altitude() function from pysolar. The goal is to calculate the angle between the normal vector of the solar panel and the vector from Earths surface to the sun. This is achieved with this formula: effectiveArea $= A\ cos(\gamma) = A\ (sin\theta\ cos\phi\ sin\theta'\ cos\phi' + sin\theta\ sin\phi\ sin\theta'\ sin\phi'\ cos + cos\theta\ cos')$ [10]. Where θ is 90° minus the angle of the sun, ϕ the sun's azimuth θ' the angle of the solar panel to the Earth's surface and ϕ' the azimuth of the solar panel.

Then for each value in the list the energy output can be calculated as follows: $effectiveArea_{solarpanel} * radiation_{total} * efficiency_{solarpanel}$. Figure 10 shows the calculation for the energy output of a house in the following 48 hours. The last step of the calculation is the last line in figure 10, where the effective area times the efficiency times the energy prediction are multiplied. The list energy_p is the list that stores the energy prediction for the whole town. For each house in the town the energy prediction is added to the overall energy prediction. At the end of the calculation the energy prediction for a town is assigned to the list attribute of the town, such that the energy prediction can be accessed via the town instance.

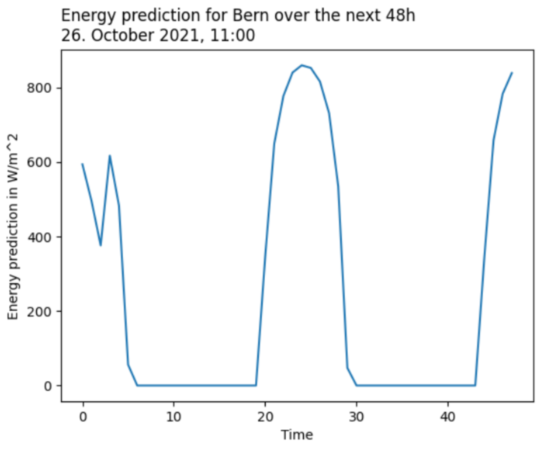
```
effectiveArea = house.solarPanel.area*(sin(radians(thetaSun))*
    cos(radians(azimuthSun))*sin(radians(inclinationSolarPanel))*
    cos(radians(azimuthSolarPanel)) +
    sin(radians(thetaSun))*sin(radians(azimuthSun))*
    sin(radians(inclinationSolarPanel))*sin(radians(azimuthSolarPanel))*
    cos(radians(inclinationSolarPanel)) +
    cos(radians(thetaSun))*cos(radians(inclinationSolarPanel)))
#the energy output is the effective area * the irradiation * the efficiency
houseEnergy[i] = round(effectiveArea * houseEnergy[i] *
                    house.solarPanel.efficiency, 1)
energy_p[i] = round(energy_p[i] + houseEnergy[i], 1)
```

Figure 10: Calculating the total energy produced

The Calculations class is very practical because of its modularity. If something in the calculations of the energy prediction should be changed it can be done fairly easily without having to alter any other modules in the code. For this project the calculated approximation of the energy production is accurately enough to simulate the scheduling of appliances on the basis of the prediction. From figure 11a and 11b it is apparent that the cloud prediction and irradiation prediction correlate. The figures show the prediction from 11 o'clock in the morning for the following 48 hours. For the first hours it is still quite cloudy, therefore the irradiation is only at around 500 W/m^2. After that the irradiation falls to 0 because of the sunset. The next morning, about 20 hours later, there are no clouds left, the irradiation grows rapidly and falls rapidly after 10 hours, which reflects the time from sunrise to sunset.



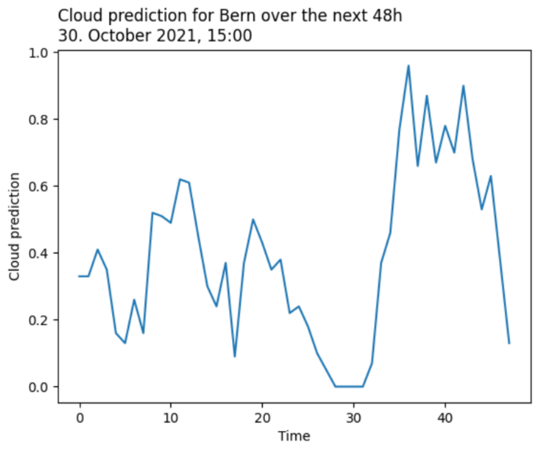
(a) Cloud prediction



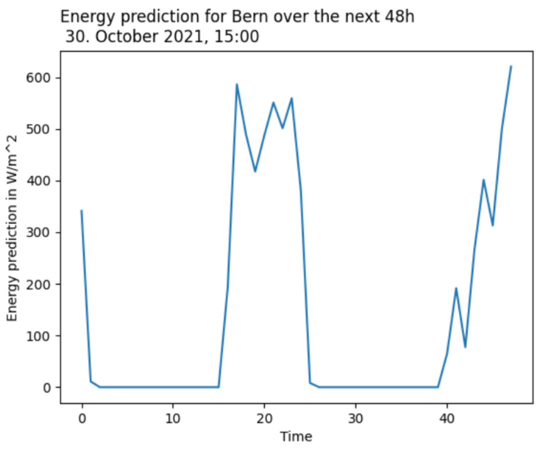
(b) Irradiation prediction

Figure 11: Predictions for Bern on the 26th of October 2021

Since the weather was fairly sunny around the 26th of October, to compare the data, measurements were also taken for a day where it was more cloudy. This is shown in figure 12, where the prediction for clouds is much higher. As above in the irradiation prediction there are two very clear dips, where the irradiation is 0. That is due to the nights, where there is no sunlight. The peak



(a) Cloud prediction



(b) Irradiation prediction

Figure 12: Predictions for Bern on the 30th of October 2021

around 20 hours after the time where the data was taken, isn't as nicely curved as in figure 11b. This reflects that for that time clouds are predicted and the solar panels will produce less energy.

5.2.3 Evaluation of the predictions

What these figures show, is that although the predictions might not be completely accurate, they do represent the real world fairly well and are definitely a good approximation for this project. However, for a bigger project or a continuation of this project, a better prediction which considers more parameters would be important.
Another approach to the prediction would be to get a database on the energy production of a solar panel over a year. This database however must be very big, preferable over several years and for different locations. Multiple years would be necessary to get different weather conditions on the same day of the year. Over the span of a year the position of the sun changes and the change is different for every location on earth. Getting such data could be quite difficult or a research team would have to collect the data themselves.
In this project the angle and the orientation of the solar panel were considered in the calculations for the energy prediction. However, that was just an approximation and can vary. The Pysolar module calculates the irradiation for a location if the surface were perpendicular to the sun rays. This means it doesn't take the angle into account at which the surface is tilted or whether it is directed into a certain direction. For a further project the exact calculations could be extended and become more accurate. Other factors such as diffuse and reflected light could be simulated and added to the energy production. Also other weather conditions could be considered, such as wind, which also has a small influence on the energy production of solar panels.

5.3 Scheduling

The scheduling of the appliances happens in a class called Scheduling. That class consists only of one static method, which takes the town index and the current datetime as arguments. As with the Calculations class, the static method allows the program to call the schedule_appliance() method on the class name. It makes the program more modular than writing the method into the Appliance class, which would have made the class very big and the method would only have been able to be invoked on a specific appliance instance.
The main goal of the scheduling class is to take all appliances within a town and allocate them a time to run, where the maximal possible amount of energy is taken from the solar panels. Therefore for each appliance it has to be checked at what time it can get the maximal amount of solar energy, also in dependence to all the other appliances that have to be run.
There must also be a way to distinguish between appliances that are running, that have already finished or simply have a scheduled time. If an appliance is running, the energy used for that appliance has to be subtracted and appliances that have already finished should be ignored by the scheduler. An easy way to implement this, is to have an attribute state in the Appliance class. This way the scheduler and the user can see what an appliance is doing at the moment. The user is able to see that on the Start page of the graphical user interface, which is depicted in figure 19. More details concerning the Start page are discussed in section 5.4.1.
When an appliance is created, the constructor automatically assigns the value "waiting" to its state. That is the state in which an appliance is in, when it hasn't yet gotten a scheduled time from the scheduler. Once it gets a schedule time from the scheduler, the state is set to "scheduled". As soon as the time has come for the appliance to run, its state is changed to "running". When the duration of the appliance has elapsed, the appliance's state is set to "finished". For each state an appliance is in, the scheduler has to consider different aspects.
In a first step the function copies the towns energy prediction list to the list availableEnergy. A copy is enough to work with, since it is not the goal to change the energy prediction list of the town. After that all appliances belonging to the town are sorted by their state. This is done with a loop, that goes over all houses and their respective appliance list. With a list comprehension, each appliance matching a certain state is filtered and stored in a local list. An example is shown in figure 13, where all appliances with the state "waiting" are filtered and stored in a waitingListj. The j stands for the current loop cycle, where each cycle iterates over the appliances of a different house. After having extracted all appliances within that house, the waitingListj is joined with the waitingList, which stores all appliances from the whole town. This is done for the states "waiting", "scheduled" and "running".

```python
waitingListj = [app for app in config.Towns[t].houseList[j].myAppliances if
    app.state == "waiting"]
waitingList = waitingList + waitingListj
```

Figure 13: List comprehension to extract all waiting appliances

5.3.1 Running appliances

The scheduler uses the runningList and the scheduledList to subtract their needed energy from the energy prediction. The amount of energy that is left can then be used to schedule the waiting appliances.
To subtract the energy that a running appliance is needing, the scheduler has to figure out for how long the appliance will still be running for. To do that the scheduler uses the current time and subtracts the scheduled time from it. The resulting time difference is converted into hours and tells the scheduler for how long the appliance has already been running for. The code snippet in figure 14 depicts this conversion. Since a minimal difference in seconds could mean that the hour_difference could be rounded down, it is made sure that the minute, second and millisecond attributes of the variables this_date and app.scheduledTime are set to 0. This way the resulting

difference of the two variables will always be the exact amount of hours.

```python
difference = this_date - app.scheduledTime
tot_sec = difference.total_seconds()
hour_difference = int(tot_sec/3600)
```

Figure 14: Converting the difference into hours

Once the difference is calculated, there are two possibilities how far along an appliance can be. Either it still has a few hours to run or it has finished running exactly at this hour. Therefore if the hour_difference is exactly the same amount of hours like the appliance's duration, the scheduler knows it has finished running. So the appliance switches its state from "running" to "finished". If not, the difference between the duration and the already passed hours is calculated. For how many hours the appliance still has to run is stored in the variable hours_left. In figure 15 the for loop then uses the hours_left variable to define for hour many iterations it has to run. Then the energy the appliance needs for the next few hours is subtracted from the availableEnergy list.

```python
hours_left = app.duration-hour_difference
for i in range(0, hours_left):
    availableEnergy[i] = availableEnergy[i] - app.power
```

Figure 15: Subtract the energy used by already running appliances

Because the list comprehension creates a copy of all the appliances, the appliances that have finished need to be updated in the appliance list of the house. A for loop iterates over all appliances in the runningList and if the appliance's state is "finished" it looks for that appliance in the houses. To make sure it is the right appliance, the __eq__ method was overridden for the Appliance class. This way two appliances can be compared with the == operator, which returns True if two appliances are equal. If the operator weren't overridden it would return True if the two objects have the same storage space. If the equal operator returns True, the found appliance in the house is updated. It then has the state "finished" which will be used in the last step of the Scheduling class discussed in section 5.3.4.

5.3.2 Scheduled appliances

After updating all the finished appliances, everything that has to be considered concerning the running appliances is done. The next step is to subtract the energy that will be needed by the already scheduled appliances. Similar to the waitingList, a for loop iterates over the scheduledList. For each appliance the amount of hours when the appliance will start running is calculated. In a second loop, shown in figure 16, it is iterated over the timespan when the appliance will run. That timespan starts at the calculated amount of hours into the future and ends at the amount of hours the appliance needs to run. For those hours the needed energy of the appliance is subtracted from the values in the availableEnergy list.

```python
for i in range((hour_difference), (hour_difference+app.duration)):
    availableEnergy[i] = availableEnergy[i] - app.power
if hour_difference == 0:
    app.state= "running"
```

Figure 16: Subtract the energy used by the scheduled appliances

If the hour_difference is equal to 0, it means that the appliance starts running at this exact hour. Therefore the state of the appliance is changed from "scheduled" to "running", depicted in line 4 of figure 25. When the next scheduling happens an hour later, this appliance will be in the list with the other running appliances. Analogue to the running appliances which changed to being finished, all appliances that changed their state from scheduled to running need to be updated in the houses. Again each appliance is looked for in the houses, once found their state is updated to "running".

5.3.3 Waiting appliances

The availableEnergy list contains the energy for each hour after having subtracted the energy needed by already running or scheduled appliances. Waiting appliances can now be scheduled according to the energy left in the grid. The highest priority when scheduling the appliances is given to their end time. Their end time is fixed and has to be adhered to. After the list comprehension, where all waiting appliances are extracted, the waitingList is then sorted by their attribute endTime. Those with the nearest end time will be at the beginning of the list. This way it is made sure, that the appliances with an end time near in the future is scheduled first.
A for loop iterates over the waitingList and for each appliance it is checked whether its end time lies further in the future than 48 hours. This is done because there is no data for more than 48 hours, so scheduling them doesn't make sense. The iteration of the loop is then cut short with the keyword continue. Nothing changes about the appliance, its state stays the same and in the next scheduling it is checked again if it can be scheduled.
If the appliance has its end time within the next 48 hours, the time when it should run is calculated. The goal is to get the time interval, where the maximum amount of energy is drawn from the solar panels. To do this a variable maxRestEnergy is declared. That variable stores the value of how much energy is left in that time interval, if the appliance would be running. It is initialised with the time interval from the current hour for the amount of hours it takes the appliance to run. This is the reference used in the for loop. The for loop shown in figure 17 goes over all possible intervals of the availableEnergy list and adds up the energy that would be left over after subtracting the energy needed by the appliance. If the energy left over is bigger than the current maximal value, a new time interval has been found, which is more energy efficient. Then the value maxRestEnergy is updated with the current value, which is line 6 in figure 17. In a second variable named index, the index at which that time interval starts is stored.

```
  for i in range(1, (hour_difference-app.duration)):
2     restEnergy = 0
      for j in range(i, i+app.duration):
4         restEnergy = restEnergy + (availableEnergy[j]-app.power)
      if maxRestEnergy < restEnergy:
6         maxRestEnergy = restEnergy
          index = i
```

Figure 17: Calculate the remaining energy after scheduling the appliance

When the loop ends, the time interval where the highest amount of energy is left in the grid is stored in the index. If the index is equal to 0, it means the time interval starting at this hour is the most efficient. Therefore the appliance has to be scheduled directly and its state has to be changed to "running", therefore skipping the state "scheduled". This is done on line 2 and 3 in figure 18. The method setScheduleTime() is invoked, which assigns the current date as the scheduled time for the appliance.
In the case that the index isn't 0, the scheduled time must be the current time plus the amount of hours stored in the index. This piece of code can be seen on line 5 of figure 18, where the time delta of the amount of hours is added to the current date. The chosenDate is then set as the scheduled time for the appliance and the appliance's state is changed to "scheduled".

```
if index == 0:
    app.setScheduleTime(dateNow)
    app.state = "running"
else:
    chosenDate = dateNow + datetime.timedelta(hours= index)
    app.setScheduleTime(chosenDate)
    app.state = "scheduled"
```

Figure 18: Set the scheduled time for the appliance

5.3.4 Finished appliances

In a final step, all finished appliances in the town are considered. The finished appliances are shown on the Start page for 24 hours after they have finished. This is done so the user can be sure his appliance has run after he might have scheduled it a while ago and didn't have a look at the interface anymore. Therefore the scheduler has to go through all appliances and checks whether their state is ”finished”. If it is, the time difference from the finished time until the current time is calculated. The appliance method removeSelfFromHouse() is invoked if the time difference is 24 hours. The method removeAppliance(), which belongs to the house class, is called and removes the applicance from the appliance list of the house it belongs to. With this approach each appliance in each state is accounted for and each appliance passes through each state and is then deleted from the house.
On the Start page of the graphical user interface (see figure 19) the energy usage of the current hour is displayed. After having subtracted the energy that each appliance is using this hour, the first item of the availableEnergy list is therefore how much energy is left in the grid. If the value is positive, it means not all of the available energy is used, if it is negative it means that some energy is drawn from other sources than solar panels. Therefore the used energy is the energy prediction of that hour minus the rest of the available energy. This value is then stored in the usedEnergy attribute of the town. That attribute is then accessed by the graphical user interface and displayed on the Start page. How much of the produced energy is used by the appliances in the town is therefore always displayed on the Start page.

5.3.5 Evaluation of the scheduling

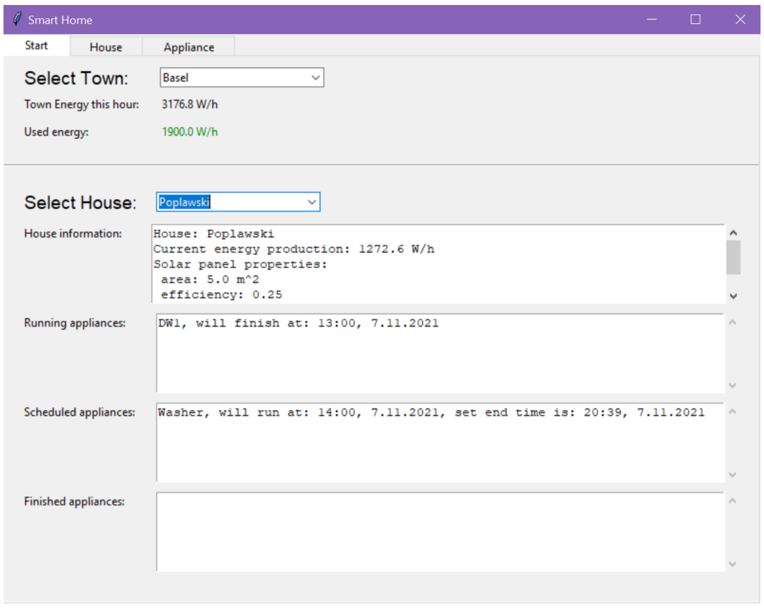
One main thing that could have been improved is that appliances that are already scheduled are checked again for each iteration, whether that is still the best scheduling time. But also in consideration of the other appliances, such that they use as much solar energy as possible. For that a bigger storage system would have been necessary, to compare previous scheduling with the new one to see which one is better. Even some machine learning might have been necessary for such an approach. Continued work on the thesis could be concentrating solely on the optimisation of the scheduling, such that all possible combinations are checked and the one with the least energy waste is used.

5.4 Graphical User Interface

For the smart home it is very important to have a well-arranged graphical user interface. That is all the user will be able to see and the only part where the user can interact with the program. Therefore the graphical user interface should be as user-friendly as possible. Python has multiple libraries to create graphical user interfaces, but Tkinter is the most widely used and was therefore was chosen for the project.
The goal was to make it as user-friendly as possible. Such that the user can navigate through it in a easy and clear way. It should serve as a prototype for a real smart home, where house owners can acquire the program to schedule their appliances. It should be easy for the user to schedule new appliances, but also have a overview over the state of the house and the appliances belonging to it. That's why it should have a page which shows all necessary information, so the user has an understanding of what is happening in the smart grid.

5.4.1 Start page

To have a modular GUI it is useful to have different pages for different parts of the program. The first step was to create a Tkinter notebook, where different pages can be added to the notebook, which are then displayed in form of a menu. The idea was to have a Start page that is responsible for displaying all information that is useful to the user. Since everything in the program can be accessed through the towns, it was clear to structure the GUI in the same way. So first the town in which the house is located has to be selected. As soon as it is selected, the energy that town produces at the current hour is displayed beneath. Additionally the amount of energy that is currently used by the grid is shown. If the amount used is within the energy production of the solar panels it is displayed in green. If the town is using more energy than the solar panels can produce at that time, the used energy is displayed in red. That way it is visible if there are many appliances running and how much of the towns energy they are using.



Select Town: Basel
Town Energy this hour: 3176.8 W/h
Used energy: 1900.0 W/h

Select House: Poplawski

House information:
```
House: Poplawski
Current energy production: 1272.6 W/h
Solar panel properties:
 area: 5.0 m^2
 efficiency: 0.25
```

Running appliances:
```
DW1, will finish at: 13:00, 7.11.2021
```

Scheduled appliances:
```
Washer, will run at: 14:00, 7.11.2021, set end time is: 20:39, 7.11.2021
```

Finished appliances:

Figure 19: Start page of the graphical user interface

When a town is selected, there are only certain houses that belong to that town. The box to select the house has to be updated each time the town is selected. That is done by binding the selection of the town to a method, where the method updates the values of the house box. To bind a method to a combobox, it has to be specified which action should deploy the method. In this case 'ComboboxSelected' means as soon as the user has selected a value from the combobox,

the function getTownValue1() is deployed. The code snippet in figure 20 depicts the binding of a method to a combobox.
The function getTownValue1() takes the town that was selected, then goes through the list of the houses that belong to the town and sets that list as the values for the combobox with which a house can be selected. Finding the right house names can be seen in figure 21. Before the names are assigned to the combobox, they are sorted alphabetically to make it easier for the user to find its house.

```
self.SelectTownCombobox1.bind('<<ComboboxSelected>>', lambda x:
    self.getTownValue1())
```

Figure 20: Binding a function to the selection of a combobox

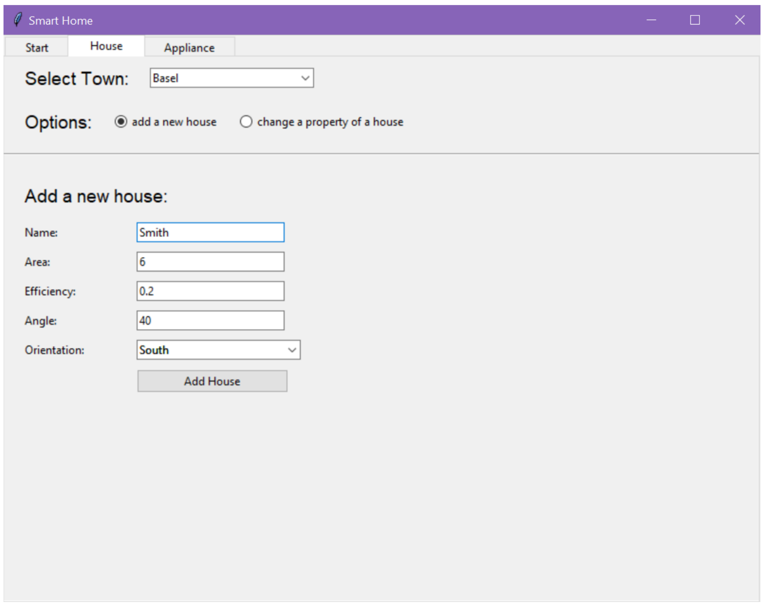
```
# gets the town selected in the town selection combobox
valueTownSelected = self.SelectTownCombobox1.get()
# looks for the right town in towns list
for i in range(0, len(config.Towns)):
    if config.Towns[i].name == valueTownSelected:
        chosen_town = config.Towns[i]
# shows all houses in that town in the combobox house
namesList = [h.name for h in chosen_town.houseList]
namesList.sort()
self.selectHouseCombobox1['values'] = namesList
```

Figure 21: Update the house values from the selected town

After having selected the town, the user can select the house that he is interested in. When selecting the house, all the information concerning that house in displayed in the text field below. The first text field shows the properties of the house. This is useful in case an owner of a house forgot some properties and wants to get the information. Below there are three further text fields, which display the appliances belonging to the house. The first one shows all running appliances, the second one the scheduled, or waiting to be scheduled, appliances and the last one the appliances that have already finished. The last text field is useful for the user so he can be sure the appliance has finished and since when it has finished. It is also useful to have a overview to all the appliances the user has scheduled, to be sure that he has scheduled them. The Start page also serves as a confirmation that a value has been updated, when the user changes something about his house or appliance. Changing a value of a house or an appliance is addressed in the sections 5.4.2 and 5.4.3.
That every user can access every house in the smart grid is obviously a privacy violation. It is done this way, because the smart home is a prototype and to test it, it needs to be possible to access all houses in all towns. If the program were to be used commercially, it would have to be altered such that each house owner could only access his own house. Then the owner would also be logged into his house automatically and there would be no need to select the town and house each time. Also the House page would become redundant and could be deleted from the graphical user interface.

5.4.2 House page

Adding a new house to the smart grid or changing a property of a house can be done with the House page. To switch to the House page, House has to be clicked in the menu at the top. Then a new page will be shown, where there are two options, either to add a new house to the smart grid or to change a property of a house.
This page consists of two frames, one that is in charge of adding a new house, the other of changing a property of a house. To switch between the two frames, the user has to select which


Smart Home

Start | House | Appliance

Select Town: Basel

Options: [✓ add a new house] change a property of a house

Add a new house:

Name: Smith
Area: 6
Efficiency: 0.2
Angle: 40
Orientation: South

Add House

Figure 22: House page of the graphical user interface

option he wants with the buttons at the top of the frame. The buttons are bound to a function, that rids the page of the frame and puts the frame that was selected on the page. This is done with the command .grid() and .grid_forget(). These functions either place a frame in the grid or get rid of a frame from the grid.
If the "add a new house" button is selected, five entries are open for input from the user. In order to create a new house, the user has to input the name of the house, the area of the solar panels, their efficiency, the angle at which they are set up and the orientation of the house. One of the biggest problems for most programs are false user inputs. Although it is specified in the user manual in which unit each input has to be, there are checks for every single input. For example if the user inputs a letter instead of a number for the area, the program will show an error message, stating that the area has to be a number and the user will be able to correct his mistake. The program also recognises if an input has a wrong value, for example if the input to the area is negative. A negative area makes no sense and will throw an error.
The code for this error handling is shown in figure 23. The errors that might occur are caught with a try except clause. That way an error doesn't terminate the program, but it stops what it is doing and returns what is specified in the except clause. In case an error occurs, an error message should pop up with a message of what went wrong. Additionally the entry field where the error occurred is emptied, such that the user is directly able to see where he has to input a new value. With a try and except clause it is possible to handle different types of errors differently. However the code snippet in line two of figure 23 throws a ValueError if the value cannot be converted into a float. The second part of the code also throws a ValueError if the value is negative. That is why two try clauses are needed, to distinguish between the two types of errors and being able to display two different error messages depending on the error that occurred. After an error occurred the function shouldn't continue to execute or the program might start to malfunction. That's why the return keyword at the end of the exception is needed, such that the the function call is ended directly.

These checks are programmed for each entry and for each kind of error there is a separate error message. It is also important to note, that to be able to distinguish between houses that belong to the same town, each house name has to be unique. Before a new house is created, it is checked whether a house with that name already exists in that town and whether the input isn't the empty string. If a house with that name already exists, an error will occur and tell the user to change

```python
try:
    a = float(self.areaEntry.get())
except:
    showerror('Error', "Area has to be a number")
    self.areaEntry.delete(0, 'end')
    return
try:
    if (a < 0):
        raise ValueError("Area can't be negative")
except:
    showerror('Error', "Area can't be negative")
    self.areaEntry.delete(0, 'end')
    return
```

Figure 23: Error handling of user input

the name of the house. To add a new house to the town, the "Add House" button has to be clicked. That button is bound to all the checks mentioned above. If all entries pass their respective checks, the house is added to the town by calling the House constructor and passing the entries as arguments. After the house has been added, a message will occur saying it was successful. The comboboxes on the first page and on the house page are automatically updated, so all houses are shown and can be selected.

A user must also have the possibility to change a property of a house. For example if the user added more solar panels to his house or if the angle at which they are tilted has changed. To do this, the button "change a property of a house" has to be selected. When selecting the button the frame will change and entry boxes to change a specific property are shown. The town to which the house belongs has to be selected, then all houses belonging to that town will be shown in the combobox "Select House". Then the combobox "change" offers all properties that the user can change about the house. After having selected the property, the new value can be entered. Here again the user has some responsibility to input the right value for the chosen property. But the program helps the user, because all checks from adding a new house are reused here. A try except clause is placed around the assignment of the new value. Because of the getter and setters described in section 5.1, if the new value doesn't match the conditions in the setter, the setter will throw an error. This error is caught by the except clause and the error message will tell the user what the problem is.

Instead of changing a property, there is also the possibility of deleting a house from a town. This might be the case when a house is taken down or when a house doesn't want to be part of the local smart grid anymore. For this the town and house have to be selected and then the "Delete house" button (see figure 24) can be clicked. This will deploy a function that looks for the house in the town list and invokes the class House method removeSelfFromTown(). That method calls the method removeHouse() which belongs to the class Town. The method is invoked on the town the house belongs to and the town then removes the house from his own list. If the house was removed successfully, an information message appears saying the deletion was successful.
Each time something about a house is changed or deleted, the comboboxes to select houses is updated automatically, such that the user always sees the current values for all towns, houses and appliances. To see the new values of a house, the user can switch back to the Start page. After selecting the house, all properties are shown in the information box. That way the user can be sure the program saved the changes.

5.4.3 Appliance page

One of the most integral parts of the program are the appliances. To simulate a smart home, the user has to be able to add appliances to his house. As with the houses, appliances should also be able to be altered in case the user made a mistake or an aspect about the appliance has changed.

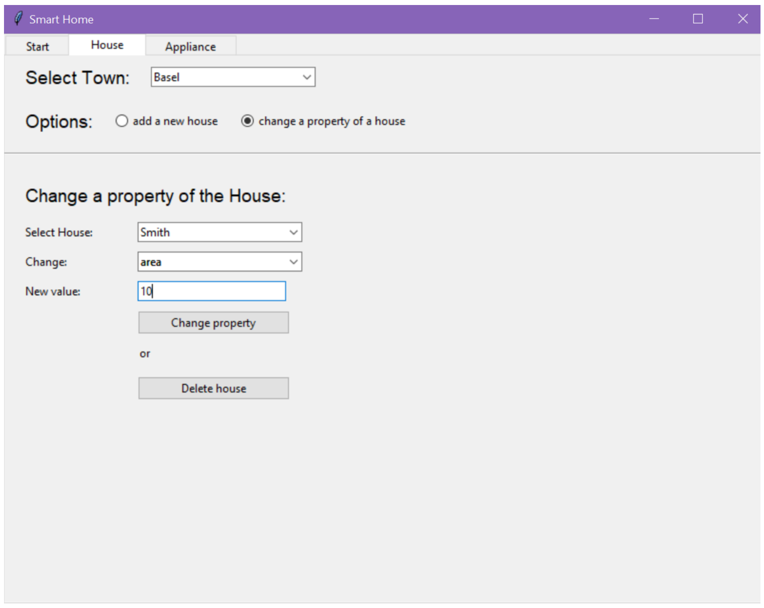
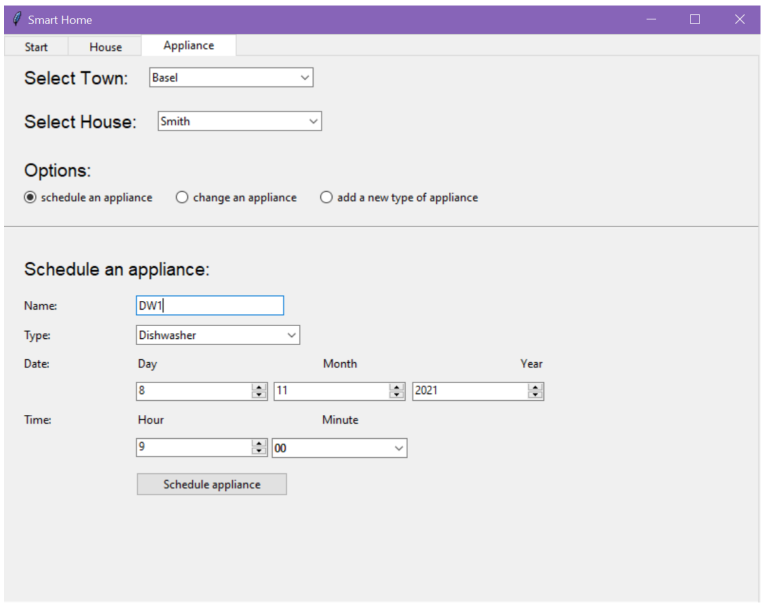


Figure 24: Change a house page of the graphical user interface

The last page in the Tkinter notebook is the Appliance page. With that page everything that has to do with appliances can be managed.
When selecting the Appliance page, the first frame shown is the "schedule an appliance" frame. That is the most important frame, since scheduling the appliances is the main task of the smart home. As with the previous two pages, the town and then the house to which the appliance belongs to have to be selected. This is due to the fact that instances are always accessed via the config.Towns list, so the program needs to know where to append the new appliance.
Once the town and house have been selected, the user needs to input the values for the new appliance which can be seen in figure 25. As with the user input for the house, the user input for the appliances is checked before it gets assigned to a new instance. So for each input there are type and value checks, which raise an error if the input isn't valid.

Firstly a name for the appliance is entered. That name has to be unique within the house and a non empty string. This decision was made to be able to distinguish appliances within a house from one another, since a house could have two washing machines or two cars running at the same time. To check the uniqueness of a name, a method checkAppNameExists() was written, which goes through all appliances in that house and checks whether there is an existing appliance with that name. If the method returns True, it means it found an appliance with that name and it will raise an error. However if the method returns False, the value for the name is accepted.
Each appliance has an attribute for the duration and the power it needs. Because an appliance always needs the same amount of time and power, it is easier for the user to specify the type of appliance and let the program assign the right values for the duration and the power. The appliance class has a static list called TypeList, which stores all types of appliances as a triple. That triple consists of the type's name, its duration and the power it needs. When the appliance constructor is called, the name of the type is passed to it as an argument. The constructor goes through the TypeList and assigns the values for duration and power to the instance. This behaviour is shown in figure 26. This doesn't only save time, it also helps against human error.
When the type is entered, it is checked whether the type exists in the list, since the user could also type anything into the combobox. If the input isn't a type that exists in the TypeList, an error is thrown, which is caught by an exception. The exception will show an error message, stating that the type doesn't exist, so the user knows he has to select an existing type. How to add new types to the TypeList is addressed later in this section.


Smart Home

Start | House | Appliance

Select Town: Basel

Select House: Smith

Options:

(●) schedule an appliance ( ) change an appliance ( ) add a new type of appliance

Schedule an appliance:

Name: DW1

Type: Dishwasher

Date: Day 8 Month 11 Year 2021

Time: Hour 9 Minute 00

Figure 25: Schedule an appliance page of the graphical user interface

```
for triple in Appliance.TypeList:
    n, watt, hour = triple
    if n == type:
        self.power = watt
        self.duration = hour
```

Figure 26: Attribute assignment of power and duration

The last part of scheduling an appliance is to define until when the appliance should be finished. To chose the date, spinboxes were created, which helps the user to not chose an invalid number, e.g. 40 for the day. Then the time has to be chosen. Each of the five entries are first checked whether they are a number. If not a TypeError is raised, which is then caught and the user gets an error message. After having passed those checks, the numbers are given to the datetime constructor as arguments. That piece of code is also checked, if it raises an error, it gets caught and the user gets informed that he inserted a non-existing date, e.g. the 31st of November. Finally all inputs are given to the Appliance constructor. That line of code is also packed into a try except clause, because there is an endTime setter. The setter makes sure that the chosen end time of the user is far enough into the future, such that the appliance is still able to run. For example if it is 8 o'clock in the morning, and the car battery needs 8 hours to charge, the end time must be at least at 4 o'clock in the afternoon or later. Therefore if the constructor throws an error, an error message is displayed to the user saying that the end time isn't far enough in the future.
To schedule the appliance and append it to it's house, the "Schedule appliance" button must be clicked. The botton is bound to the schedule_appliance_button() method, which takes all the inputs and gives them to the Appliance constructor. Within the constructor the method addAppliance() of the class House is called. The newly created appliance instance is given to that method as an argument which appends the appliance to the appliance list of that house. If the creation of the instance was successful, an information message is shown to the user to inform him of the creation. Following this, the Start page is updated, such that the user can check whether the new appliance was really appended to the house. The whole procedure and security checks when creating a new appliance are analogue to the ones described when creating a new house in section 5.4.2.

As with the houses before, a user should be able to change a property of an appliance. Whether it is it's end time, duration or any other property. To do this another frame for the Appliance page was created. When selecting the check box "change an appliance" the frames are switched from the input frame to the change frame which is shown in figure 27.

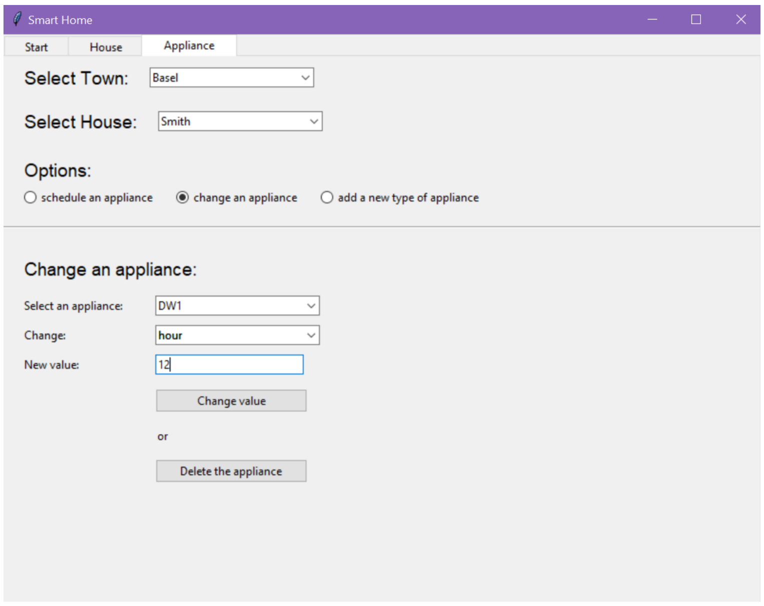


Figure 27: Change an appliance page of the graphical user interface

Analogue to the change house frame, the appliance that the user wants to change is selected, then the property and finally the new value is entered. The same issues as with changing a property of a house arise. It is important that the type and range of values entered by the user are right for the selected property. For each kind of entry checks are in place, which control whether the input meets the conditions for that attribute. If not an error is raised, which is caught for each specific occasion. The resulting error message informs the user what kind of error occurred and what has to be altered in the entry box.
It was especially important to check the changing of the duration and the end time properly. In the case that the user would prolong the duration, it would be possible that he would prolong it such that it would have to run past the set end time. In figure 28 on line 2 the calculation to check whether the new duration would exceed the set end time is shown. With the datetime module it is possible to compare dates and add a time difference to a datetime instance. First the current date and time are retrieved from the now() method. Then the value from the input is taken, which is the amount of hours the appliance should run for. That amount is incremented by 1, because the current time has to be rounded to the next full hour, since appliances are always scheduled each hour. That timedelta is then added to the current date. If the sum of the two is bigger than the set end time, it means that there aren't enough hours left from the next full hour until the end time for the appliance to run. Therefore it will raise an error and tell the user that the chosen duration is too long.

On the other hand, when the user wants to change the set end time, the program needs to check whether there is still enough time for the appliance to run. The assignment of the new value for the end time is within a try except clause. Because there is a setter for the end time, when assigning the new value, the setter automatically checks whether the new end time is still far enough in the future for the appliance to run. If the setter raises an error, the user gets an error message that the entered value isn't accepted and has to enter a new value.
The last thing that had to be thought of, is that appliances which have a new end time, duration or power needs, have to be rescheduled. As described in section 5.3 only appliances with the state

```
try:
    if (datetime.datetime.now() + datetime.timedelta(hours=(1 + int(value))) >
    app.endTime):
        raise RuntimeError
except:
    showerror('Error', "Duration is past the set end time")
    self.newAppValueEntry.delete(0, 'end')
    return
```

Figure 28: Changing the duration of an appliance

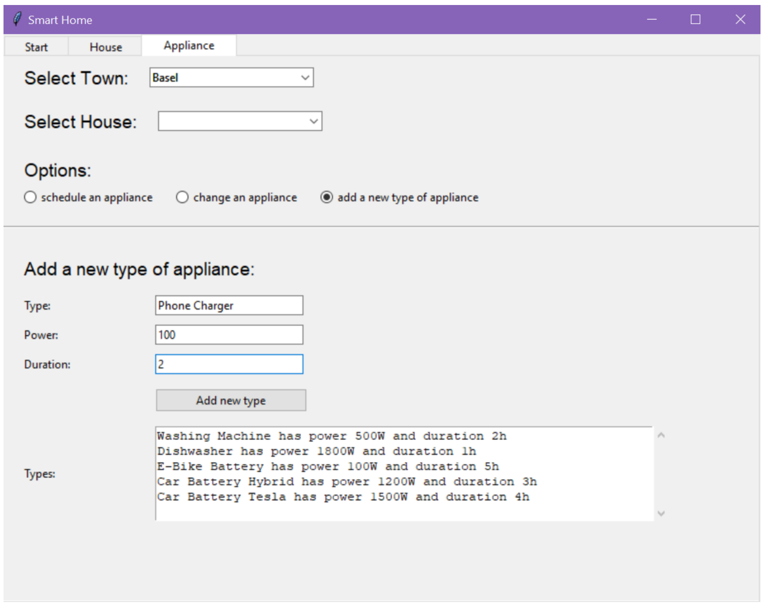
"waiting" get scheduled. Consequently when the end time, duration or power of an appliance is changed, that appliance's state is reset to "waiting". If the appliance already has a scheduled time, that time is reset with the resetScheduledTime() method.
Beneath the "Change value" button, there is also a "Delete the appliance" button. That way a user can delete an appliance he doesn't want to run after all. To do so the appliance has to be selected and then the delete button can be clicked. When clicked, the delete_application_button() method is invoked which searches for the appliances in the appliance list of the town. Once found, it applies the removeSelfFromHouse() to the appliance. In that method the appliance is given as an argument to the removeAppliance() method of the house. With that method the appliance is deleted from the list of appliances. If no errors occurred a message is displayed that the appliance was successfully deleted. When an appliance is changed or deleted, the Start page is automatically updated. That way the user is directly able to see whether the appliance has changed or isn't there anymore.

A third option on the Appliance page is to add a new type of appliance to the list of all types. To do so the check box "add a new type of appliance" has to be selected, then the frame will change. Adding a new type is then done via the frame shown in figure 29. Similar problems have to be considered as with the other frames. First of all a new type cannot have the same name as an already existing type, otherwise the program wouldn't be able to distinguish between two types. For the user to know which types already exist, there is a box at the bottom of the page, which displays the names of the types and their values for power and duration. That should help the user not to try and create an identical type. Nevertheless there are checks when creating the new type to make sure the name isn't the empty string and doesn't already exist.

For the entries of the power and duration there are each two checks, to make sure they are both a number and that the value is in a certain range, e.g. both numbers can't be negative. If all entries pass their respective checks, the triple is added to the TypeList of the class Appliance. This is why the list is static, because there are two advantages. For one the list can always be accessed via the class name, even if no instances of appliances exist, the list can still be altered. The other is that if the list is altered, the changes can be seen from all instances, which makes it similar to a global variable.
What has to be considered is that types cannot be altered or deleted after their creation. Therefore the user has to be sure to have entered the right values. If the type has a mistake it simply won't be used to create appliances.

5.4.4 Evaluation of the graphical user interface

In total the graphical user interface is easy to use, clearly structured and has all necessary functionalities. From a user perspective there are a few things that could be improved. For one the types of appliances are shown on the last frame of the interface, but cannot be altered. It would be more user-friendly to be able to select a type of appliance and change or even delete it. The other thing that could be improved, is that for every page the town has to be selected separately. This is due to the fact that the comboboxes on the respective pages are three different boxes. What the user doesn't realise, is that when he selects the town on the Start page, it isn't the same combobox as the one on the house page. Because the interface is made up of a notebook, which contains



Select Town: Basel

Select House:

Options:

schedule an appliance change an appliance [✓ add a new type of appliance]

Add a new type of appliance:

Type: Phone Charger

Power: 100

Duration: 2

Types:

```
Washing Machine has power 500W and duration 2h
Dishwasher has power 1800W and duration 1h
E-Bike Battery has power 100W and duration 5h
Car Battery Hybrid has power 1200W and duration 3h
Car Battery Tesla has power 1500W and duration 4h
```

Figure 29: Add a new type of appliance page of the graphical user interface

three different pages, it wasn't possible to create one combobox for all three pages.
An additional feature of the interface could have been to show the energy prediction in a graph. To visualise the scheduled appliances, they could be placed on top of the graph in form of boxes. If the graph line were above the box, it would mean that the appliances are running solely on solar energy, if the height of the boxes exceeded the graph, parts of the appliances would run on other electricity.
From a programmers point of view, the interface should have been programmed in a much more modular way. All the different pages and buttons are declared in the same class. It would have been possible to create separate classes for each frame and then create a specific instance of each class for the interface. Also all the methods that are called because they are bound to a button, are unique for each button. Within the methods there are parts of code that are reused for different parts of the interface. To have less redundant code, the recurring parts could have been declared in separate methods, which are then called from the method that is bound to the button. If the sole focus would have been on the graphical user interface, it could have been improved by using programming languages that are more versatile for graphical user interfaces. Especially its look could be improved, since many users prefer interfaces that are more appealing.

5.5 Threading

For the graphical user interface to stay open until the user explicitly closes it, the GUI instance has to be added to the mainloop(). That loop keeps the window open and it makes the GUI wait for events, such as a button click. At the same time the aim was to let the scheduler run once an hour. That can be achieved with the schedule module provided by python [14]. This module allows to schedule a certain function every few minutes/hours or at a certain time each minute/hour. The get_prediction() method from the Calculations class and the schedule_appliances() method from the Scheduling class are invoked inside a function called calculation(). This function is called once an hour by the schedule module. For this to work, an infinite loop is created which calls the method schedule.run_pending(), which runs all scheduled methods.
Since both the GUI and the schedule module need an infinite loop to work, they need to run simultaneously. This is achieved by creating two threads, where one thread creates the GUI and the other calls the schedule function. However this leads to a new problem, namely that the schedule method doesn't see the instance of the GUI, because the GUI is created within the other thread. They do share the same variables, namely the Towns list which stores all towns, houses and appliances, but they can't communicate with another.
Only after scheduling the appliances is it necessary for the threads to communicate. The Start page should get updated each time the scheduling of the appliances was done, such that the information displayed on the page is always up to date. To bridge the problem, that the schedule thread can't see the window created in the GUI thread, a static list within the GUI class was created. That list contains all create instances of the class. This was the GUI instances can be accessed via the class name. This is what the scheduler method takes advantage of. It updates the Start page by applying the update method to each instance of the class GUI. In this case it is just one instance.
When creating threads that share variables the problem of race conditions occur. If both threads try to access the Towns list at the same time that could lead to severe errors. It is therefore important that only one thread at a time can read/write to the Towns list. The threading module has an inbuilt object for this, namely locks. As the name suggest, locks lock the variable for a certain thread, such that no other thread has access to that variable when a lock is active. Locks are implemented around all variables which belong to the Towns list. Also in the functions where the Towns list is manipulated, such as the get_predictions() and schedule_appliances(), locks are set around them, such that another thread cannot access the Towns list while it is updated.

6 Conclusion

Energy consumption is growing world wide. To meet the demands fossil energy is used, which accelerates climate change and isn't a long term solution. Therefore more and more research and investments are made into renewable energy sources. Those forms of energy must however be incorporated into the electrical grids. To accommodate the new needs, smart grids are developped all over the world, which should help with distributing energy and use it more efficiently.
Advantages of such grids are energy efficiency, having local grids, being able to incorporate many small suppliers and many consumers and using more clean energy. However a drawback is that a lot of money and effort will have to be put into transforming our current electrical grids into smart ones. Also since a smart grid is heavily dependant on the internet and data, this could lead to security and privacy problems and a target for cyber attacks. The same argumentation applies to smart homes. If appliances are connected to the internet, they can also be hacked.
This thesis scratched the surface of a real smart grid and smart home and set the focus on three things: the energy prediction, the scheduling and the graphical user interface. Each of the three parts are an approximation of a real smart grid, which try to simulate the behaviour, but don't consider all necessary factors.
A lot can be improved in the energy calculations. For one many more factors influence the energy production of solar panels, so this could be improved in a continuation of the thesis. Another aspect is that smart grids collect a huge amount of data to calculate and simulate the energy prediction by using machine learning. Instead of calculating the energy prediction, it could be done by collecting data over an amount of time and evaluate it. This in itself could comprise a whole thesis.
This approach would also mean that the data has to be stored somewhere. An external storage that also stores all the changes to the smart grid was omitted in this thesis. When closing the program all changes that were made are lost. That would be a necessity for a real smart grid and smart home, but as a first prototype it is acceptable not to have a storage.
The scheduling part of the code is based on a very basic scheduling algorithm. Some research and maybe even machine learning would have to go into the scheduling algorithm to optimise it. Also the aspect that appliances that are already scheduled aren't rescheduled every time to check whether it is still the optimal time to run them isn't ideal. But for that much more storage would be needed and all possible scheduling possiblities would have to be compared to find the optimal one.
Smart homes will only become commercially viable, if the user interface is easy to handle. Some aspects would have to be improved, such as having to select the town on each page, instead of selecting it once. Also there should be two different programs, where one is for the service provider and the other for the users. The users would only be able to see their own house and the information about the town. But to maintain the smart grid, the service provider would be able to have access to all towns and houses.
As a further project, the program could be extended to have security and privacy measures. Since everything in a smart grid and smart home is on the internet, it has to be secured against cyber attacks. This was completely omitted in this project and wouldn't be realistic.
What wasn't done was to evaluate the effectiveness of the smart grid. It could be done by running simulations of the energy consumption on a global grid and on a local smart grid. The aim would be to compare the ratio of renewable energy in the local grid to that of the global grid. The higher the ratio would be, would mean that more renewable energy is used in the smart grid, which means it is more efficient in comparison to the global list. That way there would be a measure on how effective the smart grid is.
Overall the thesis achieved the main goal of trying to predict the energy production of a solar panel, of being able to schedule appliances according to the energy prediction and having a working user interface. However each part of the program has its limitations and could be improved.

References

[1] Bundesamt für Statistik Anteil erneuerbarer Energien am Endenergieverbrauch. `https://www.bfs.admin.ch/bfs/de/home/statistiken/energie.html`. Sept. 2021.

[2] WWF Austin smart grid. `https://wwf.panda.org/wwf_news/?204660/Austin-smart-grid`. Sept. 2021.

[3] Wikipedia: Daylight. `https://en.wikipedia.org/wiki/Solar_irradiance`. Aug. 2021.

[4] axisbits Future Development of Smart Homes. `https://axisbits.com/blog/Future-Development-of-Smart-Homes`. Oct. 2021.

[5] Smart grids and Swiss Federal Office of Energy smart metering. `https://www.bfe.admin.ch/bfe/en/home/supply/electricity-supply/electricity-networks/smart-grids.html`. Sept. 2021.

[6] Requests: HTTP for Humans. `https://docs.python-requests.org/en/master/`. Aug. 2021.

[7] Wikipedia Hydropower. `https://en.wikipedia.org/wiki/Hydropower`. Sept. 2021.

[8] G. Masters. "Renewable and Efficient Electric Power Systems". In: *Wiley-IEEE Press* (2004).

[9] Prevision meteo. `https://prevision-meteo.ch/services`. Aug. 2021.

[10] Arbitrary Orientation and PV education Tilt. `https://www.pveducation.org/pvcdrom/properties-of-sunlight/arbitrary-orientation-and-tilt`. Oct. 2021.

[11] How long before we run out of fossil fuels? Our World in Data. `https://ourworldindata.org/how-long-before-we-run-out-of-fossil-fuels`. Sept. 2021.

[12] Read the docs Pysolar. `https://pysolar.readthedocs.io/en/latest/`. Aug. 2021.

[13] Attila Fodor Roland Bálint and Attila Magyar. "Model-based Power Generation Estimation of Solar Panels using Weather Forecast for Microgrid Application". In: *Acta Polytechnica Hungarica* 16.7 (2019).

[14] Read the docs Schedule. `https://pypi.org/project/schedule/`. Sept. 2021.

[15] solar.com Solar panel efficiency. `https://www.solar.com/learn/solar-panel-efficiency/`. Sept. 2021.

[16] Yale Environment 360 The Challenge for Green Energy: How to Store Excess Electricity. `https://e360.yale.edu/features/the_challenge_for_green_energy_how_to_store_excess_electricity`. Oct. 2021.

[17] Electrical Technology What is a Smart grid technology? `https://www.electrical-technology.com/2019/05/what-is-smart-grid-its-overview-and-advantages.html`. Sept. 2021.

Appendix

A User Manual

User Manual to the Smart Home Program

Author:	Nina Eldridge
Address:	Holeeholzweg 75
	4102 Binningen
E-Mail:	nina.eldridge@unifr.ch
Place and Date:	Freiburg, 28. October 2021

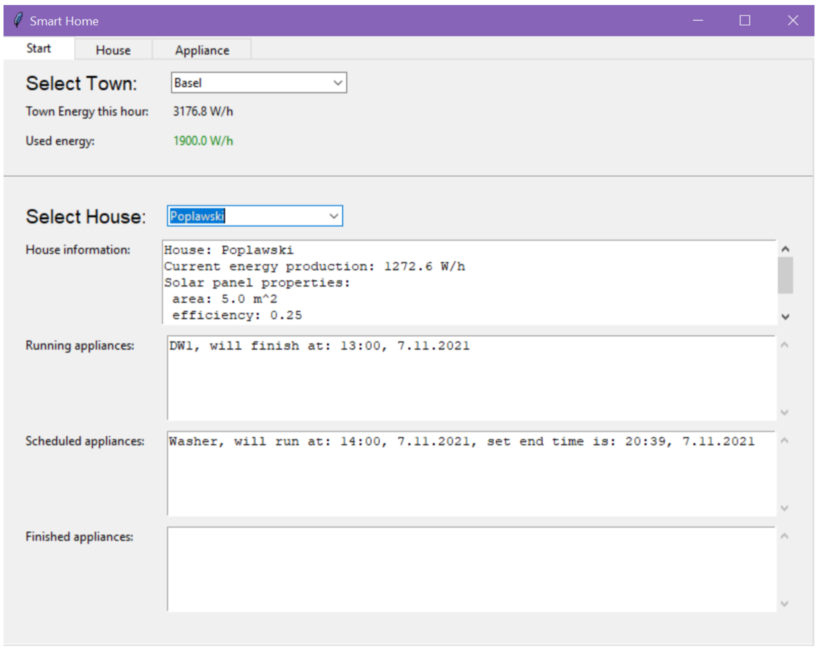
Contents

1 Introduction

In this user manual all functionalities of the Smart home program are explained. It should be used as a guide and a help of how to use the program. Carefully read which input types are accepted for which kinds of input, although error messages will help with false inputs. When working with the program, always select boxes from top to bottom, such that all information is displayed correctly.

2 Start page

When opening the program, the first page that is shown will be the Start page as seen below in figure 1.



Select Town: Basel
Town Energy this hour: 3176.8 W/h
Used energy: 1900.0 W/h

Select House: Poplawski

House information:
```
House: Poplawski
Current energy production: 1272.6 W/h
Solar panel properties:
 area: 5.0 m^2
 efficiency: 0.25
```

Running appliances:
```
DW1, will finish at: 13:00, 7.11.2021
```

Scheduled appliances:
```
Washer, will run at: 14:00, 7.11.2021, set end time is: 20:39, 7.11.2021
```

Finished appliances:

Figure 1: Start Page

The Start page serves to display information about towns and the houses. The first step is to select the town from which one wants to get the information from. As soon as the town is selected, e.g. Basel in figure 1, the information about the current energy production and current energy usage in that town is shown below. If the energy usage of the town in displayed in green, it means all appliances are

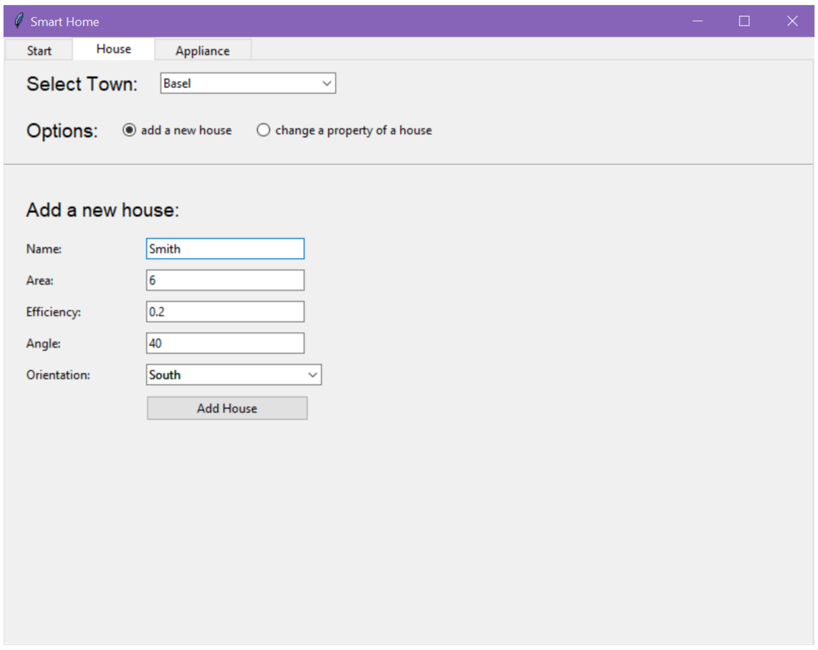
2

currently running on solar energy. If it is displayed in red, there wasn't enough energy produced by the solar panels to run all necessary appliances.
To get information about a certain house in that town, the next step is to select the house. As soon as the house is selected, all information about the house such as its name, its energy production and all properties of its solar panels are displayed in the House information box. This is especially useful to check what values a house has and which values should be changed/updated (see chapter 4).
The three boxes below serve to display the information about all appliances belonging to the selected house. In the first box all running appliances are shown, in figure 1 there is a dishwasher currently running. Below that all scheduled appliances are shown with the time for which they are scheduled. If an appliance hasn't been scheduled yet, it will say "waiting to be scheduled". The last box contains all appliances that have finished, to show that they have been done. The finished appliances will delete themselves from the list 24 hours after having finished.
Since the start page is to display information, nothing can be selected and changed on the start page. To add or change a house, select the House page (see chapters 3 and 4). To schedule or change an Appliance, select the appliance page (see chapters 5 and 6).

3 Add a new house

When opening the House page, the first page that is shown is to add a new house to the town as shown in figure 2.

To add a new house to a town, the town to which it should be added has to be selected, in figure 2 that is Basel. Then all necessary information concerning the new house has to be entered. Firstly it is the name of the house, which has to be unique for each house in the town.
Then all the information concerning the solar panels of the house has to be entered. The area in square meters, the efficiency between 0 and 1, the angle relative to the ground at which the solar panels are tilted and the orientation of the solar panels. If there is an invalid input the program will show an error message saying which entry is invalid and why.
For the rare case that a house has no solar panels, but still wants to be part of the smart grid, the input for the area has to be 0. The efficiency and angle can also be set to 0 and for the orientation any value can be chosen, since it will be disregarded by the program during the calculations.
When all entries have a value, the "Add house" button can be clicked. An information message will show that the house has been added to the town. To find the house, switch back to the start page, select the town and then the new house can be selected under "Select House" (see chapter 2).
If some properties change or if a mistake was made when creating the house, it is possible to change one property at a time. For that check the box "change a property of a house" under the "Select Town" box. When selected, the page will change to "Changing a property of a house". How to use that page is shown in chapter 4.



Select Town: Basel

Options: [✓ add a new house] change a property of a house

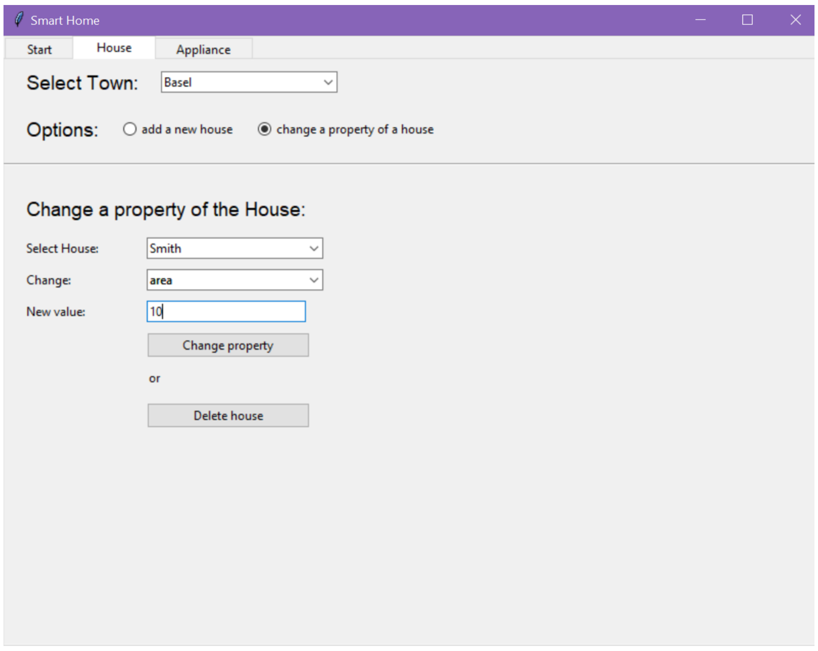
Add a new house:

Name:	Smith
Area:	6
Efficiency:	0.2
Angle:	40
Orientation:	South

Figure 2: Add a new house

4 Change a property of a house

After having added a new house, there is always the possibility to change certain properties of the house. It is important that it is possible to change the values of the solar panels, since a house could get more solar panels, or new ones with different efficiency or have them set up at a new angle.
To do so select the house page and then checkt the box "change a property of a house". Then select the house and beneath the property that should be changed as shown in figure 3. The new value for that property can then be entered below. Depending on which property is chosen different values are accepted. The user should know which kind of values are suitable for which property, but if there is a mistake, an error message will show what kind of values are accepted. After having entered the value, press the "Change property" button, such that the value is updated. A message will be shown if it was successful.
On this page there is also the possibility to delete a house from a town. Either if the house is taken down or if it won't be a part of the smart grid anymore. To do this make sure the right town is selected and the house that should be deleted. Then click the "Delete house" button. A message will appear saying the house was deleted successfully.


Smart Home

Start | House | Appliance

Select Town: Basel

Options: ○ add a new house [✓ change a property of a house]

Change a property of the House:

Select House: Smith

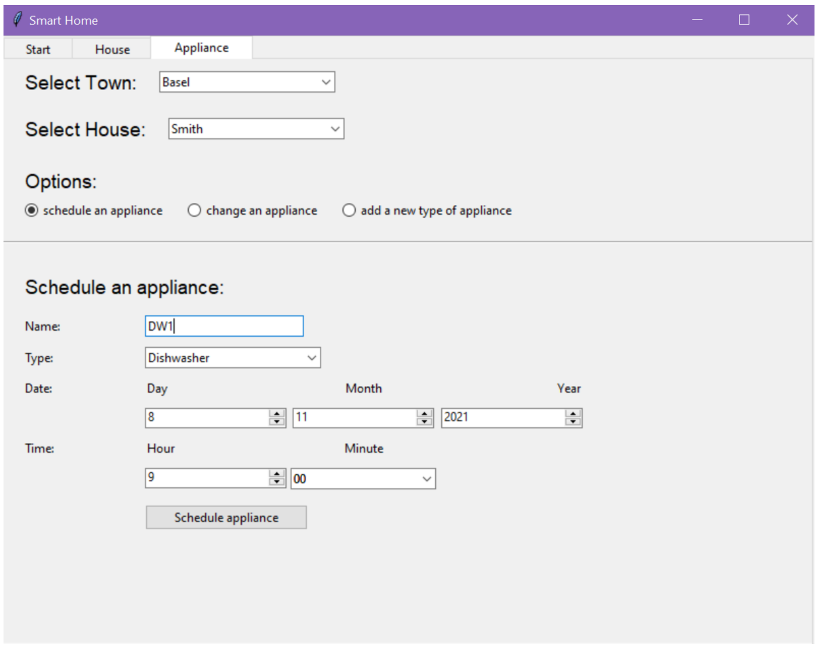
Change: area

New value: 10

Figure 3: Change a property of a house

5 Schedule a new appliance

To make use of the smart home, appliances should be scheduled within houses. The Appliance page handles everything that has to do with the appliances. An appliance is always of a certain type, e.g. a dishwasher or washing machine. Since a dishwasher will always use the same amount of energy and time, that data is stored in the type, so only the type has to be specified but not specifically the energy and time needed. However if for once an appliance will take longer or uses a different amount of energy, that can be changed in the "change an appliance" page (see chapter 6).
As on the previous page, first the town and then the house to which the appliance belongs have to be selected. For each appliance a name has to be chosen to be able to distinguish between two washing machines for example. That name has to be unique within each house. If an appliance with that name already exists, it will show an error message.
Then the type of appliance is selected. However if the appliance is of a new type that isn't yet in the database, new types can be added to the smart home. This is done on the last page (see chapter 7).
Finally the date and time until when the appliance shall be done have to be set.



Select Town: Basel

Select House: Smith

Options:
[✓ schedule an appliance] change an appliance add a new type of appliance

Schedule an appliance:

Name: DW1

Type: Dishwasher

Date: Day 8 Month 11 Year 2021

Time: Hour 9 Minute 00

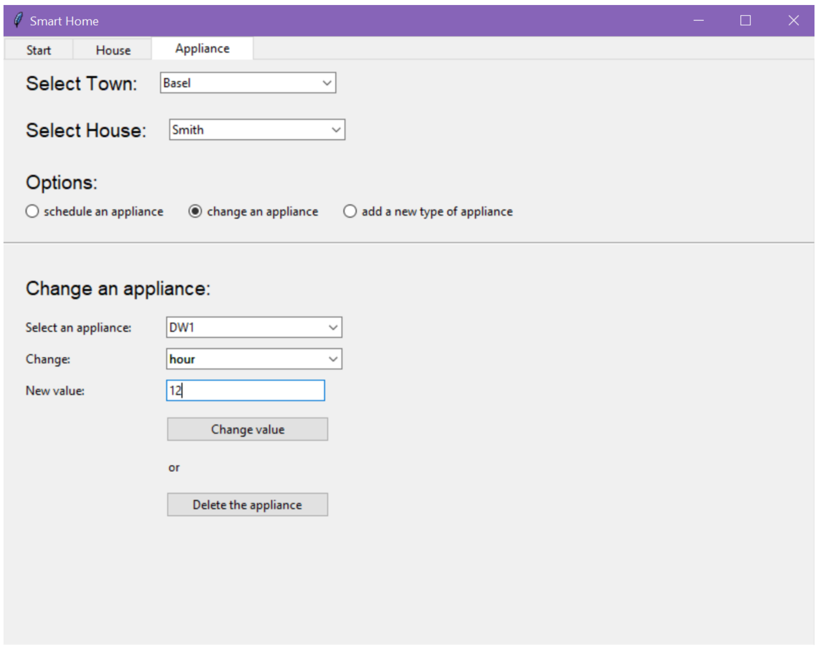
Figure 4: Schedule an appliance

The spinboxes (figure 4) are a help such that invalid numbers aren't selected, e.g. 14 for the month. The end time of an appliance is only every half hour, that is why the "Minute" box only has two options. To schedule the appliance, the "Schedule appliance" button has to be clicked. A message will appear to say it was successful. If it happens that a wrong date is entered, e.g. 31. November, an error message will display the mistake and a new date can be entered. Also, an error occurs when the selected date and time are too near in the future, such that the appliance doesn't have enough time to run.

6 Change a property of an appliance

Appliances that are already running or which have already finished can't be altered anymore. But for appliances that are still waiting to be scheduled or have a schedule time, it is possible to change a property. For example if there was a mistake when adding the appliance or if something about the appliance has changed. To do that select the town and house to which the appliance belongs and check the button "change an appliance" under "Options".

Chose which appliance has to be altered, then chose which property has to be changed. Then insert the new value below. Press the button "Change value" and the value for that appliance will be updated.
Here the user has to be careful to insert valid values for the chosen property. However the program will return an error message, if a wrong value was entered. Especially when it comes to changing the end time or the duration of the appliance. The duration can't go past the set end time and the end time can't be moved forward such that there isn't enough time left to run the appliance.



Select Town: Basel

Select House: Smith

Options:
( ) schedule an appliance [✓ change an appliance] ( ) add a new type of appliance

Change an appliance:

Select an appliance: DW1

Change: hour

New value: 12

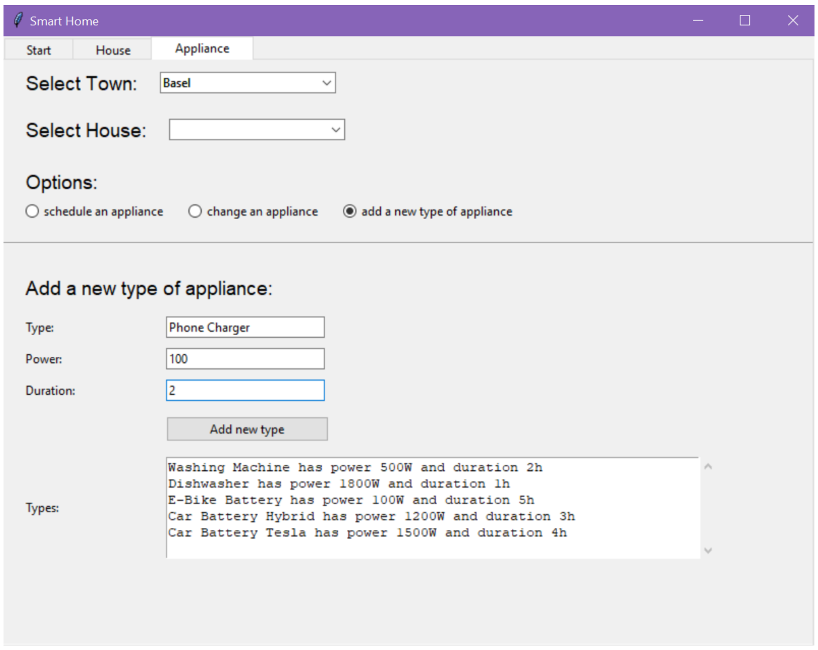
Figure 5: Change an appliance

On this page there is also a "delete" button as can be seen in figure 5. If an appliance was created that shouldn't be scheduled after all, the appliance can be selected and by pressing the delete button it will be erased from the house.

7 Add a new type of appliance

The last feature of the smart home are the types of appliances that are stored in the program. To make it more user friendly and to save time, some types of appliances are defined, such that they can be selected when scheduling a new appliance.

If a new type of appliance should be added to the database, the name of the new type, the power it has and the duration it needs have to be entered. The power is in Watts and the duration in hours. By clicking the "Add new type" button, the new type will be added to the list. This is also shown in the box below, which lists all types of appliances and their properties (figure 6). The new type can then be used to schedule an appliance of that type.



Select Town: Basel

Select House:

Options: ( ) schedule an appliance ( ) change an appliance [✓ add a new type of appliance]

Add a new type of appliance:

Type:	Phone Charger
Power:	100
Duration:	2

Types:

```
Washing Machine has power 500W and duration 2h
Dishwasher has power 1800W and duration 1h
E-Bike Battery has power 100W and duration 5h
Car Battery Hybrid has power 1200W and duration 3h
Car Battery Tesla has power 1500W and duration 4h
```

Figure 6: Add a new type of appliance