AN AGENT-BASED SIMULATION OF ENERGY GENERATION AND EXCHANGE IN A SMART GRID

Ruben Kip (s2756781) & Thijs van der Knaap (s2752077)

Faculty of Science and Engineering
Rijksuniversiteit Groningen

Daily supervisor: Ang Sha
Supervisor: Marco Aiello

July 2017

Abstract

Using a multi-agent system to simulate an energy grid has advantages in speed performance since it can run in parallel, due to the concurrency of agents. Additionally, it supports isolated decisions by houses, which are the core players of a real world smart grid. In this project, we work with JADE which is an agent development framework for Java. We try to create a simulation representative of a smart energy grid of a residential area. The simulation needs to be fast, modular and of course "smart" by which we mean that houses should be able to sell and buy energy from neighbours according to a given priority. With this simulation we investigate how much renewable energy sources one should add to be (largely) independent of energy providers.

The implementation can be found on Github.

keywords

Smart grid; MAS; multi-agent-system; energy exchange; distributed energy generation; simulation; topology;

Contents

1

Introduction

Currently, the energy grid is based on having a few big energy producers. They produce a controlled outlet of power, that is consumed by a lot of large and small consumers, the latter are mostly homes and offices. There are a few houses which have solar panels installed and even less own a wind turbine. The current prospects of the future indicate that the application of these green local energy sources will increase. Every house in our system will be able to produce and consume energy and will be called a prosumer. An example of a prosumer can be seen in figure 1.1. A prosumer with a lot of solar panels will on a sunny day produce more energy than it consumes, and therefore will want to sell its surplus of energy. The next day it may be rainy and therefore the production is lower than the consumption, so this house has to buy any missing energy. Although this is a positive trend it also has some problems.

When a house produces a surplus of energy, this energy is sold to an energy company for a reduced price. If the house is able to sell his surplus of energy to his neighbour, for a price that is between the buyback and sell price of the energy company, both houses would be better off. Price is used as the relevant variable in this example, but rerouting excess power to another house can be done based on many different values (like least energy loss, highest need etc).

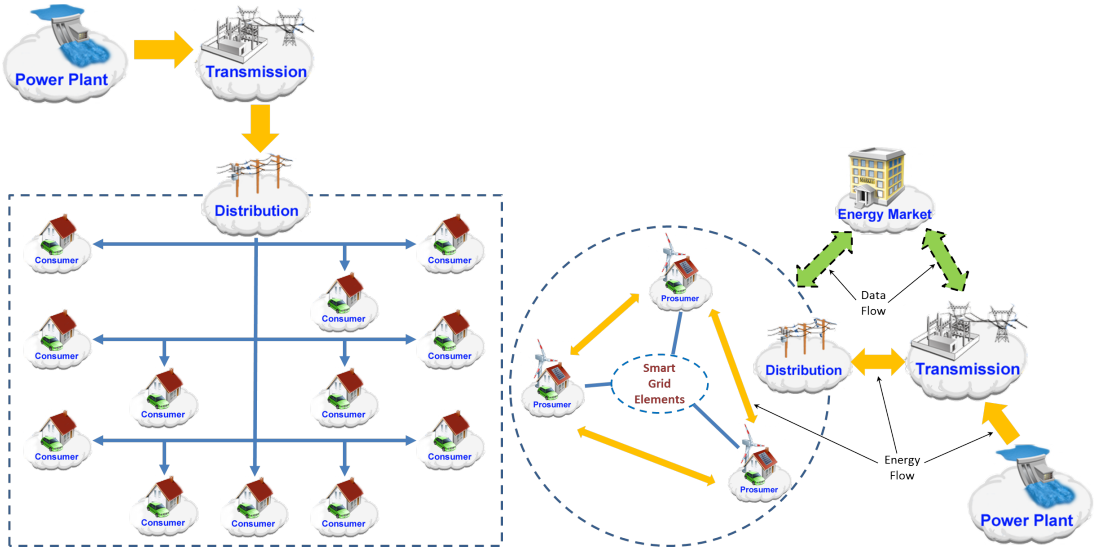
A solution to this problem could be to create a "smart" grid. A definition of a smart grid would be: "an electricity supply network that uses digital communication technology to detect and react to local changes in usage". This is a broad definition where for example reacting can be interpreted as routing, buffering, storing etc. Our definition of smart refers to the idea that every house can make its own choices and can communicate to other houses. Allowing houses to sell energy between them. This will not only give an economical incentive to invest in green energy, but would also make households less dependant on the large energy producers. Some important differences between a "smart" grid and the traditional grid are: the energy flow is bidirectional instead of only being pushed down, nodes are able to communicate with each other allowing local deals and energy need prediction and neighbourhoods become more self reliant. The comparison can also be seen in figure 1.2.

To build on the idea of becoming as independent as possible and to see the smart grid in action, one of our goals will be to see how much solar panels and wind mills need to be placed in the community to be as independent of big energy providers as possible. We will also assume the community will use a smart grid system. Independence is defined as the least amount of energy transactions with the provider, this includes buying surplus energy back.

The easiest and cheapest way of initial testing is running a digital simulation. This simulation must resemble the real world as closely as possible to get use full data. A lot of research has been done on

Figure 1.1: Example of an prosumer

Figure 1.2: Traditional(left) and Smart(right) grid [1]



this subject, with each focusing on certain aspects of the necessary simulation. More information about related work can be found at section 2

Our goal is to create a program that simulates an energy grid. This grid contains houses that can consume and produce power. The surplus of power can then either be sold to their neighbours or to the big energy producer. The focus of our project is on the interaction between houses and the appliances. To attain the mentioned independence for every house we are simulating the grid as a Multi-Agent System (MAS). An agent in a MAS is a separate entity with its own behaviour that can communicate to other agents. Every house is represented by an agent. This ensures that each house has a state and means of communication. A MAS automatically models the race condition between prosumers. For example buying energy before it is sold out. These race condition are also applicable in the real world. A second advantage that a MAS introduces is parallelism. The agents can be ran in parallel, since they only are dependant on messages, that are asynchronous. This increases the simulation power significantly.

To increase the quality of our grid simulation, real weather data is used. This is essential for simulating realistic productions of green energy producers, such as solar panels or wind turbines. The aim is to implement it in a modular manner, such that different energy producers that depend on the weather can have different sources for their information. For example, if you have a source with very accurate weather about the wind, but it doesn't contain the solar irradiance.

2

Related Work

To get an overview of the research field the present work contributes to, we review related work that has been performed on smart grids. For every work, we look for similarities and differences in the approach of achieving a realistic grid simulation. After this comparison, we summarize the knowledge that is relevant for our research, from the articles.

In addition to articles about smartgrids, we also looked at two articles concerning micro-grids. A micro-grid is a discrete energy system consisting of distributed energy sources (including demand management, storage, and generation) and loads capable of operating in parallel with, or independently from, the main power grid [2]. Smart grids are a subset of micro-grids, since they also assume independent decision-making of components. It is interesting to analyze the methods that are used for micro-grids, since they might also be applicable to smart grids.

Real-Time Energy Management in Microgrids

Wenbo Shi, Na Li, Chi-Cheng Chu and Rajit Gadh [3] research online approach of an energy management strategy with as goal to optimize long term cost. They only want to use the current system state, so no a priori knowledge. The system is formulated as a Stochastic optimal power flow problem. They conclude that their online algorithm preforms almost as good as an optimal offline algorithm and that it exceeds a greedy algorithm.

Similarities

- They also simulate the grid and its energy loss in transfer, their grid is even more advanced since it also deals with maximum capacity per line.
- They also use a big energy producer which can buy and sell energy but do have the option to run the system without it.
- Their production of solar panels and windmills are also using real weather data.

Differences

- Their approach is very different since they try to find the best distributions and actions controlled from a central point where we giving this responsibility to every individual house to make sure that

it gets the amount of energy it needs.

- They are simulating real time instead of simulating a certain time span. This changes the aim of the program and allows a more detailed simulation since you don't have a speedup factor.
- They implemented local energy storage. which allows more advanced simulation, since you now have to decide when you want to keep your energy stored or if you want to sell it for the current price.

Relevance

The approach of this article is very different due to being real-time and trying to find a distribution form a central point. If you don't use a distribution from a central point but let every agent fight for its own you automatically get a greedy approach. In their article they show that a greedy approach is almost always performing worse. In the current energy grid the energy is handled from central points, so sticking to this method but making energy transfer lines bidirectional might be a better option. One thing is certain, if we move to a grid where every house is responsible for its own energy it will get very hard to move back to a more centralized system.

Trends in Microgrid Control

The article [4] is an overview of popular micro grid technologies. The goal of the authors is to summarize and identify requirements and technologies for use in a micro energy grid. It discusses technologies like MAS, droop control, model predictive control. It concludes about the important aspects and of a micro grid and the techniques for implementing it.

Simularities

- Use also an agent based approach

Differences

- They use three phases of control, The primary layer is responsible for the real power and the input and output of this. The secondary layer is responsible for the determination of the selling price and to make deals, it links the houses together. The Third Layer allows multiple grids to be connected to each other. Our program only is focused on the second layer.
- They allow energy to be stored
- They have a central controller that holds information giving prosumers more information to determine their buying and selling prices.

Relevance

It would be really nice to have an central controller that stores data from previous deals made and thereby inform prosumers what a proper price for their energy might be. Also our implementation could be extended by the mentioned third layer allowing multiple grids to be coupled together. It also mentions

the importance of electronic storage systems and we agree that it would give a much more big guy independence

Simulation of a Smart Grid City with Software Agents

Stamatis Karnouskos and Thiago Nass de Holanda [5] aim at creating a real time simulation using JADE. The simulation simulates multiple connected cities. These cities contain randomly placed houses that contain advanced formulas to model the power consumption. In the simulation electric cars are modeled to travel between houses and cities. They conclude that their simulation can function as a backbone for future more advanced simulations.

Similarities

- Both are simulating using a MAS implemented by JADE
- Both use real weather data to calculate the production of solar panels and windmills.

Differences

- The simulation is in real time, whereas our simulation can be speedup/slowed down.
- They mention that energy must be bought and sold but don't implement it.
- They have more advanced energy consumers. Although these consumers still can only be on or off, so you for example can't put the washmachine on eco settings.
- They have simulated multiple cities and electric cars that commute between them.
- Each time a simulation is ran all the houses and other 'buildings' are randomly created and assigned.

Relevance

This article has implemented a very similar simulation program as we have, but their addition of movable cars and having more advanced energy consumers will bring theirs more to reality. A big difference is that they are simulating in real time instead of with a certain speedup. This allows per time instance for much more computing time and therefore a broader aim of features. It is really interesting to see that they assign the houses in a random location in a city. on the one hand this allows to find a lot of different cases and circumvents getting stuck into one neighbourhood and performing all your research on this. It also brings problems, since each simulation will be inherently different you can't be sure which difference comes from this random placement of houses and which come from wrongly implemented behaviour.

An adaptive agent-based system for deregulated smart grids

In article [6], the authors Nicola Capodieci, Giuliano Andrea Pagani, Giacomo Cabri and Marco Aiello, describe an agent modeling to support a service oriented system of an energy grid of the future. Their contribution is an agent-based system that will run as a simulation, and that follows game theory strategies

when shifting energy. The article contains dynamic weather, advanced dynamic prices etc. to help represent the system.

Similarities

- Also uses the same agent framework (JADE), Agent based system were agents are the prosumers.
- Same concept with prosumers and big guy.
- Energy generated very similar, only looks at windmills and solar panels, based on real weather, uses the same mathematical functions.

Differences

- Auction/Market system for selling energy and calculating pricing, for us this is static per agent.
- They use weather services instead of static weather data.

Relevance

This article shows a very similar way of handling prosumer. However it also has some differences like the auction/selling system. The auction system would make an interesting addition to the simulation because it would add more realism and adds more possibilities to research and improve a smart grid for real world purposes. Besides the auction system, the weather in this article is also very relevant to us because it uses mathematical functions for solar panels and windmills, but also because the use of real time weather is an interesting addition to the simulation.

3

Design

3.1 JADE

Java Agent DEvelopment Framework (JADE), or in short JADE, is a framework that implements the Multi Agent System following the specifications of FIPA [7]. FIPA is an IEEE Computer science standards organization which goal is to standardize Multi-Agent Systems and its cooperation with other technologies.

To create your own agent you have to extend from the Agent class [8,9]. An agent in JADE isn't built using its constructor but is instead initialized by calling the method setup. When an agent is terminated the takeDown method is called.

Actions that an agent has to perform are in JADE implemented as Behaviours [10]. To create your own Behaviour you have to extend the JADE Behaviour and at least implement the "action" and "done" method. You can create a behaviour using its constructor. The behavriour can be added to an agent using the addBehaviour method. The actions of behaviours are ran in serial, so you can be certain that there are no race conditions between two behaviours. After an action of a behaviour is ran the "done" method is inspected. If this returns true the behaviour is finished and removed from the agent. Each agent gets a thread assigned on which every action of its behaviours are scheduled.

To communicate with other agents JADE implements asynchronous message passing. Message can be retrieved from the message queue using templates, this template allows each behaviour to pop messages that are relevant for it. Since the message passing is asynchronous, behaviours don't automatically wait when they try to receive a message. To avoid a busy wait Behaviours have a block method, this method will remove the behaviour from the execution queue until a new message has been received. This happens when a message is received, so this message does not have to be relevant for the behaviour.

JADE also supplies an DFService [11], this is a kind of digital Yellow pages where agents can register that they provide a service. For example, all the prosumers in our simulation are registered as agents that can buy energy. Sellers can retrieve this information and send offers to all these agents.

3.2 Prosumers

A prosumer represents a single house, it is the heart of the simulation and contains a lot of different aspects of our program. Because of its isolated state they are responsible to handle their own energy level, by

consuming and generating energy, (see 3.6), and by sending and accepting deals (see 3.7). Knowledge about the (cost) distance to other prosumers is stored in the routing table (see 3.9). A prosumer extends "TimedAgent", this adds awareness of the simulation time, so it can properly calculate how much energy is generated (see 3.5).

3.3 Simulation settings

The simulation initial settings are stored in two files. The first is a JSON file called "grid.json" containing the structure/cable details (length, connected houses, resistance) of the grid together with the prosumers details (consumer/producer type, size, efficiency etc.) which have extra consumer/producer elements. The second is a properties file called "json-agent-container.properties" which contains all prosumers and other agents, this file is used by JADE to create agents and specify the simulation start and end time. We have chosen to use these file types as the settings because both files are easily human readable and should be changed or extended to run the simulation with different values or a different structure.

3.4 Initializer

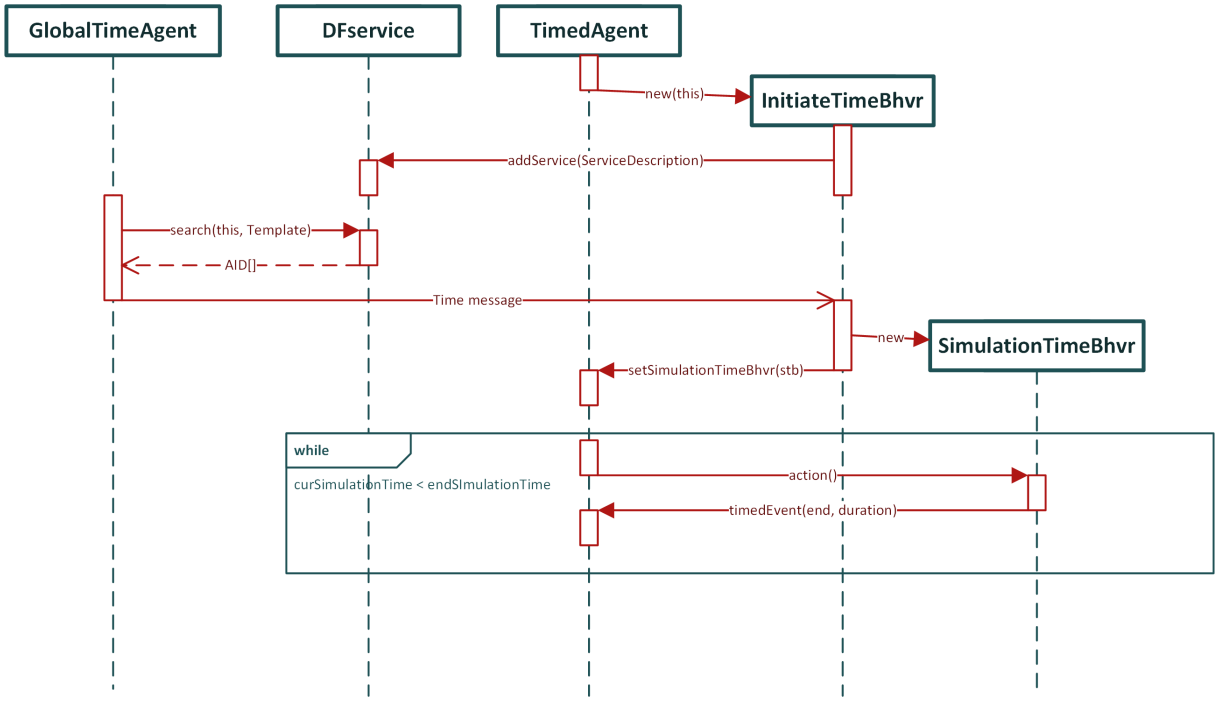
The initializer program is run by running the main inside the path: src \main \java \com \rug \energygrid \parser. It is a small UI created for quick and easy initialization of grid A 5. It shows the current grid inside "grid.json" and can initialize(with grid A) and create(when not already present) the required ".json" and ".properties" file whenever run. It also offers the possibility to change the start and end time of the simulation and give the speedup, by changing the provided fields. When you run the initializer the program will change the settings and close. now the simulation can be ran as any other JADE program. It doesn't offer the option to add your own grid and settings, this should be done manually by replacing the files, the idea behind this choice is that when you want to add extra features for your simulation we do not want to limit the possibilities by using an static UI(for example by adding an extra energy producer with different fields).

3.5 Timing the simulation

For running a simulation it is necessary to have a means of keeping track of the time. Every agent in Jade runs its own thread and can only communicate to other agents using messages. You cannot assume that a broadcast of messages will reach each agent at exactly the same time, therefore this cannot be used to simulate the passing of time. We can assume that all the agents will run on the same system, so we know that the system clock will be equal for every agent.

Every agent that needs to have awareness of the current simulation time needs to extend TimedAgent. TimedAgent will when it is initiated add itself to the DF(Yellow pages) as an "timedAgent" service. GlobalTimeAgent will give all other agents YELLOW_PAGES_REGISTER_WAIT_TIME amount of time to add themselves with the mentioned service. When this period is passed it will show the progressbar with the startbutton. After the button is pressed GlobalTimeAgent will retrieve all TimedAgents from the yellow pages and sends them a message. This message contains when the simulation will start in the current system time. The simulation will start START_SIMULATION_WAIT_TIME milliseconds later then the broadcast of the message. This allows all the TimedAgents to receive this information before

Figure 3.1: Sequence diagram initialization of time



the simulation starts. In addition to the real start time the message also contains the start and end time which will be simulated and the speedup at which the simulation will be ran. This message is handled by InitiateTimeBhvr, it creates with the received data a SimulationTimeBhvr. SimulationTimeBhvr is responsible for calling timedEvent with the appropriate parameters.

It is important to set the speedup to a value adequate for your goal, a high speedup reduces the amount of time the simulation will take, but will also make you lose precision. To give an example, if you would set the speedup 604800, this means one second is equal to a week. One cycle would take a second the amount of energy would be averaged for a week. Meaning that if you would sell any excess energy at the start of the week but buy a lot at the end these values would cancel each other out.

Behaviors are scheduled using a round Robin scheduling [8]. Currently a prosumer agent has three continues running behaviours. So without limitations our current implementation will try to run as many simulation iterations as possible. This feature is nice, but eventually resulted in an overflow of messages. This overflow is mainly caused by an energyOffer, send by every energy producing prosumer to every other prosumer. This offer will be sent each simulation step. To circumvent this problem we set a minimal time difference between every simulation step. This gives the agents time to process all the messages.

When TimedAgent is extended the abstract method timedEvent has to be implemented. This method has two parameters, "duration" and "end". "duration" tells the method how much simulation time has passed since the last time this method was called. "end" contains the current simulation time moment the simulation is at. This method is called when the SimulationTimeBhvr has performed a simulation step.

To start the all the agents simulation clocks the GlobalTimeAgent should be added to the container.

The sequence diagram of the time initialization can be found at figure 3.1

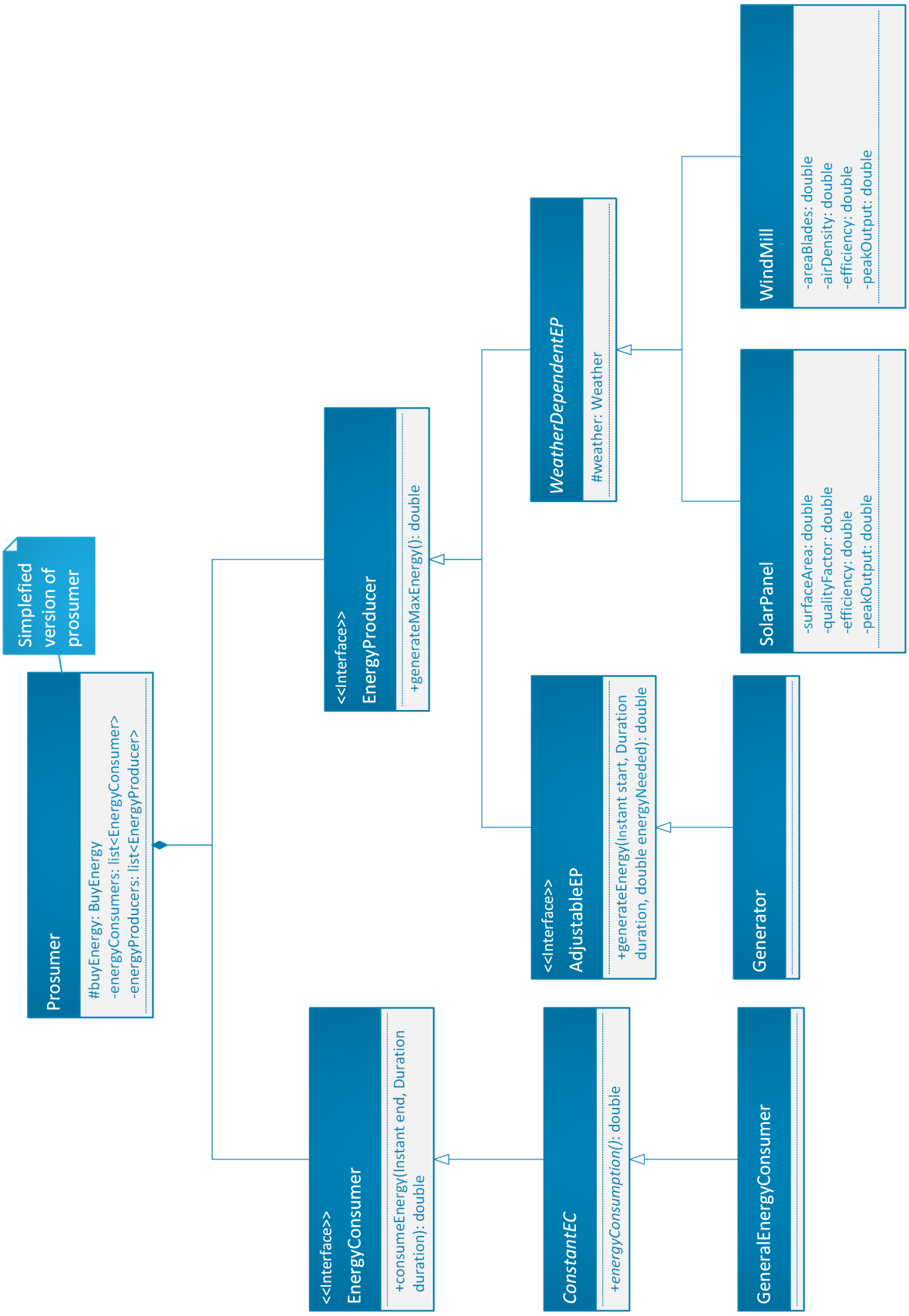
3.6 Generating and consuming energy

Every prosumer has a list of EnergyConsumers and EnergyProducers. They are both interfaces which contain either the method consume or produce energy. Every timedEvent of an prosumer both lists are iterated and the consume and produce methods are called with the appropriate "end" and "duration" value. We focused on the energyProducers. EnergyConsumer only has one used implementation, GeneralEnergyConsumer. GeneralEnergyConsumer consumes the average amount of energy for a 4 persons semidetached house in the Netherlands [12]. At EnergyProducer we extended the interface to also allow adjustable energy producers. This is for example a hydro powerplant, or a generator which is not always ran at full capacity. Currently, we are not using these adjustable energy producers, but the framework is present. We also have the abstract class WeatherDependantEP this implements EnergyProducer and is as the name suggest weather dependant. This producer will receive a Weather object in its constructor which it will use to calculate its production. This allows every producer to have its own source of weather data, increasing the modularity. For the implementation of the Weather object see 4.1

This implementation can easily be extended also make a weather dependent energy consumer, you can then think of heating and air-conditioner.

All the results of the production are subtracted by the consumption and this result is added to the current energy of the prosumer.

Figure 3.2: Class diagram of an EP's and EC's



3.7 Buying and Selling interaction

After the energy is generated or consumed the surplus energy has to be sold, or the lack of energy has to be compensated. The buy and sell interaction can be divided in two different stages, the sending of the offers and the request of buying energy. We will first discuss the sending of the offers.

If there is a surplus of energy available after the generation step, SellEnergy will send an EnergyOffer containing the amount of energy and the price of the energy per unit. The receiving prosumer will get the EnergyOffer and creates a RemoteEnergyOffer. RemoteEnergyOffer also holds the sellers address and cable Resistance. The cable resistance is retrieved from the routing table. This new RemoteEnergyOffer is now placed in the priority queue present at BuyEnergy. This queue can be ordered on different aspects, for example on price or distance, allowing agents to use different strategies when buying energy. More information concerning this queue can be found in section "Cost algorithms" 4.5.

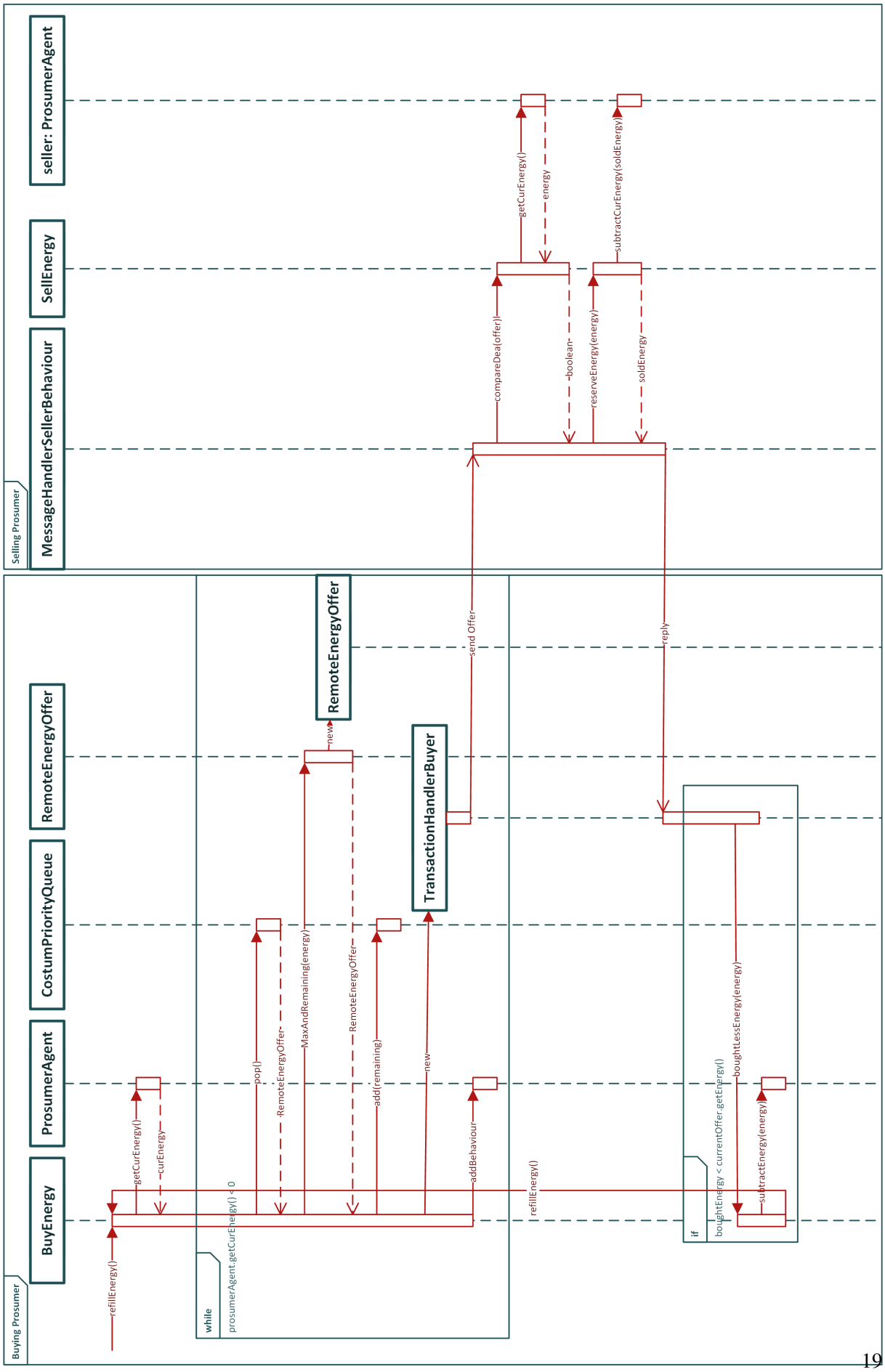
The second stage is initiated when a prosumer is short of energy after the generation. It will pop an entry from the priority queue and contact the corresponding seller with the request to buy energy. The buyer also has to pay for the energy lost in the exchange from the seller to him. The buyer assumes that it can buy the energy, therefore the supposed to be bought energy is added to the curEnergy. If the deal is not accepted by the seller the amount of energy will be reduced again. The seller receives the message that a buyer wants to buy a certain amount of energy for a given price. The seller will compare the received price with its current price, if this is unequal or the seller does not have any energy left to sell, it will send a rejection to the buyer. The Buyer will now reduce its energy with the amount that it was supposed to buy and after that will look again in the queue to find a new seller. If the prices do match and the seller still has an energy surplus it can either have enough to fill all the requested needs of the buyer, or only partially. In both cases the amount that is to be sold is subtracted from the sellers energy and is send back to the buyer. The buyer will now compare the received energy with the energy that he was supposed to buy. If this is less than supposed, the buyer will decrease its energy by the amount that it is less than. If the curEnergy is still negative it will try again to buy this energy from another seller. The buy interaction can be seen in figure 3.3.

The Big guy will also send its offer to the buyer so it is en-queued in the priority queue. The choice of buying from another prosumer or big guy if implicit. Since the big guy has infinite amount the queue will never be empty. Selling to the Big guy is a different matter. The seller is in our buy/sell interaction very passive. Therefore we added a expiration date to produced energy which starts when it is produced. We keep track of these amounts using a queue in Prosumer. If you sell or you generate a negative amount this will be subtracted from the oldest entries of the queue. If the expiration date has passed you automatically sell that amount of energy to a big guy at a reduced price.

3.8 Outputting data

The reason to run a simulation is to retrieve data. This data needs to be gathered somewhere so you can output it all in a consistent manner. For this we have the class GatherData, it uses a singleton pattern so every agent can retrieve it. It currently stores the productions and current energy status of every prosumer and all the energy deals between them. The output style of the data is easy to change. This can be done by extending the OutputData class and implementing the output method which will be called when the simulation is finished.

Figure 3.3: Sequence diagram buy interaction



Currently there are two output types implemented. The first OutputCSV simply writes all the deals that where made to an CSV file. This file can be inspected using a Excel or another spreadsheet program. Secondly OutputR, this generates a directory named output where all its output will be stored. In this directory a directory is added for every prosumer present in the system. At every prosumer directory there will be four CSV files, buyerDeals, energyStatus, production and sellerDeals. These CSV files will be read by an R script called importData. This script will parse all the data and then generate a plot for every agent containing all the corresponding information. All these plots have the same x-axis scale so they are easy to compare. The y-axis contains the time.

3.9 Routing table and Length table

Our prosumers have a routing table containing all the shortest paths to all other agents. This is useful when we want to know the costs to another prosumer from an "EnergyOffer", assuming that we always use the optimal route. The optimal route is not the shortest path in length, but the shortest path in resistance over length, $cablecost = resistance * length$.

To initialize the routing table we use Dijkstra's algorithm with a minimal priority queue. [13] Our program calculates the routing table in $O(|E| + |V| \log |V|)$ (where $|E|$ is the number of edges).

Because knowing the shortest path in length can also have benefits, like recalculating the average resistance(see 4.3) or any other future uses, we added also a length table. This table contains the length of the current (resistance) shortest path. We do not need to calculate this table separately because it is calculated while making the routing table.

4

Simulation

4.1 Weather

To improve the simulation we use real weather data to calculate the production of solar panels and windmills. The weather data set is represented in the code by a child of the abstract "Weather" class, an instance of this child class is contained in each prosumer. This approach was chosen for modularity, it offers the option to easily add new kinds of data sets to the simulation by creating a child class of "Weather". (Examples are "ExampleDataSet_KNI" and "AmericanDataSet") The new "Weather" child should contain the name of the data set file, the indexes of the columns of the required weather characteristics and the time formatting of the data set, which can be to the precision of hours at maximum.(see examples in path: src \ main \ java \ com \ rug \ energygrid \ weather)

Figure 4.1: 4 entries from the KNI weather dataset

```
240,19510113,  213,   72,   82,  108,    3,   36,   20,    ,    ,   38,   27,   22,   46,   14,    ,    ,    ,    ,
240,19510114,  225,   82,   87,  154,   13,   41,    1,    ,    ,   46,   22,    1,   68,   13,    ,    ,    ,    ,
240,19510115,  278,   77,   82,  139,   24,   41,    7,    ,    ,   49,   41,    9,   57,    1,    ,    ,    ,    ,
240,19510116,  278,   46,   62,  123,    1,   31,   17,    ,    ,   44,   29,    9,   67,   13,    ,    ,    ,    ,
```

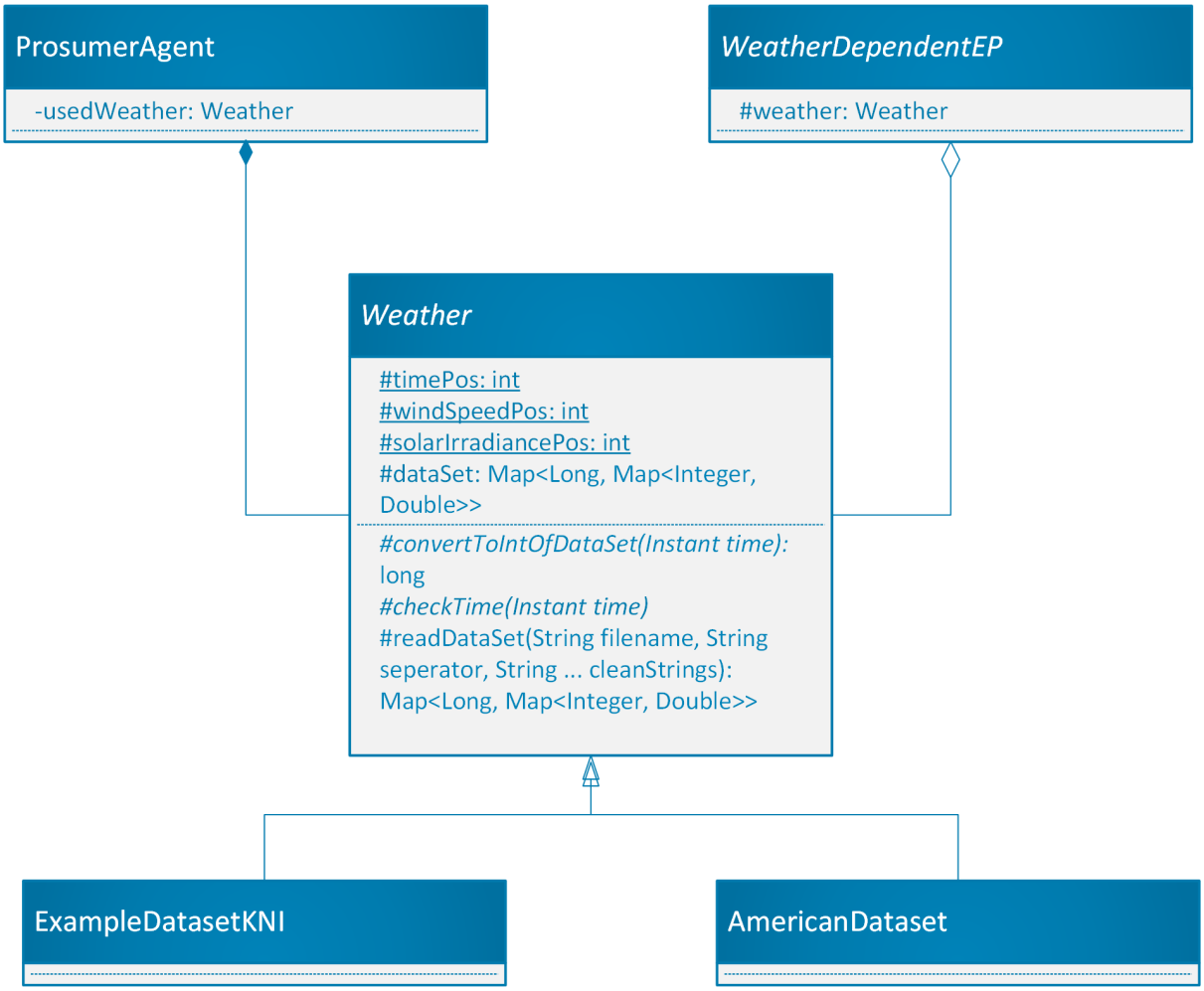
Here index 2 is the time, index 4 windspeed (in 0.1 m/s) and the last index(20) represents solar irradiance (even though empty)

A "Weather" object that uses a data set will parse the given data set on initialization. An prosumer will create his "Weather" instance on initialization. This has an advantage and an disadvantage, we have chosen this approach because it offers more freedom, now you can initialize different kinds of weather for separate agents easily. However this also means that the data set is parsed for every prosumer instead of parsing it once and sharing it over all the prosumers. Because the data set is represented by a file, although we are using an multi-agent-system(MAS) which is multi-threaded, only a single prosumer can access the file. This result in a linear increase in start up time. On an average computer this takes roughly 0.32 seconds per prosumer. By enhancing and extending the weather initialization, while keep it working with a MAS, there is some room for improvement here.

At every cycle of the simulation, the current simulation time is taken by the prosumer. Now when the prosumer wants to calculate his total energy production for that cycle using it's functions (see 4.2), this time will be be passed as an argument to the "Weather" object methods which will return the appropriate requested characteristic weather value for that time.

For now we mostly focused on a static data set, however we did also take into account the possible option of using a dynamic weather set (weather API). The structure of "Weather" is so that parsing a data

Figure 4.2: Class diagram of the weather implementation



set is optional, in this case the methods responsible for retrieving the correct values should call your API for the values instead of the data set.

4.2 Energy functions

When calculating energy, for each energy producer we will call the generate energy method which can be as simple as returning a constant value, for the weather dependent energy producers these require extra steps. As mentioned previously in the section about weather we get weather values for a specific time. These time values are then passed to the generate energy method, where they are used to get weather values, which are in their turn used in a function. For now we have implemented energy producers solar and wind energy, both require a function to calculate the required energy production.
These functions are:
For wind energy [1, 14]

$$\begin{cases} P = 0.5 * A * p * v^3 * C_w \\ P_w = min(P, P_{max}) \\ E_w = P_w * \triangle t \end{cases}$$

Where A is the area, p is the air density, v the wind speed, C_w goes from 0 to 1 and is the efficiency of the windmill and finally P_{max} the maximum power output of the windmill. ($\triangle t$ is the amount of time in seconds, to get joule generated in that time span)

For solar energy [1, 15]

$$\begin{cases} P = A * r * Q * C_p \\ P_s = min(P, P_{peak}) \\ E_s = P_s * \triangle t \end{cases}$$

Again the surface area is A, solar irradiance is r, C_p which goes from 0 to 1 is the efficiency. Q is the quality factor goes from 0.5 to 0.9, and finally P_{peak} our max output in watt.

For wind energy we retrieve the wind speed (v) and for solar energy the solar irradiance (r) from the weather data set, air density (p) is a constant(for the area around Groningen [1]) the rest of the variables are options/settings dependent of the energy producer.

4.3 Energy loss

To increase the reality of the simulation we added energy loss to the cables, this is a small effect when looking at a single prosumer, however really takes an effect when looking at long time spans and many transactions. This energy loss is handled by taking the average cable resistance over the path to a Energy Offer, so $(\sum_{i=1}^{n}(resistance_i * length_i))/totallength$ where n is the amount of cables of the shortest path between starting point and end point.

We consider the length of each cable in this formula, and take the average cable resistance over the total length instead of taking only the average cable resistance. This has the advantage that we can use routing table cost to get the total resistance over length and the length of that path.(see 3.9)
Now that we know the average cable resistance, we can use this to calculate our energy loss. We use the following functions [16]:

$$P(z) = P(0)e^{-2\alpha z} = P(0)e^{-zR_l/(Lc)}$$

$$P_{Rloss}(0:z) = P(0) - P(z)$$

$$\%P_{Rloss} = \frac{P(0) - P(z)}{P(0)} = 1 - e^{-zR_l/(Lc)}$$

Where P at z is the amount of energy left after z meters, L is the inductance per meter, c the speed of light, R_l the resistance per meter and α the attenuation factor. The value induction per meter we set as constant value (2.6) where this value is representative of a power cable [16].
This loss we subtract whenever a deal is executed, thus it is also considered when buying energy, adding the energy loss on top of the required energy.

4.4 Pricing

An advanced pricing system for prosumers can easily be added. Before an offer is accepted the price that was send must be equal to the current selling price. Also when you would change the price and there is a surplus in energy every other prosumer will be notified. You don't want to change the selling price too often since all the deals currently in process will be discarded and every time each other prosumer will receive a message. If you run the system without any added values every prosumer will sell its energy for € 0.19, but you can change this value for all in ProsumerConstants.PROSUMER_PRICE or set an individual price by giving it as a parameter in the jade-agent-container.properties at the appropriate prosumer. The sell and buyback price of the big guy are defined in BigGuyConstants and are set to € 0.22 [17] for selling and € 0.15 for buying energy back.

4.5 Cost algorithms

Even though we know the shortest path from a prosumer to each other prosumer, this doesn't mean we send our produced energy to the closest house that needs the energy. It is possible to order on things we are interested in for example price, resistance, most sold energy etc.

The queue in each prosumer (see 3.2) is ordered by a java "Comperator", this "Comperator" is our comparison/cost algorithm for deals and compares eacht time two EnergyOffers. To create an ordering a Comparator must be constructed that compares two "RemoteEnergyOffer". The Comparator can then use the all the data present in the RemoveEnergyOffers allowing for sophisticated comparison.

Currently there are three cost algorithms implemented: "GreedyEnergy" ordering on who sells the most energy, "GreedyPrice" ordering on lowest price and "MinLostEnergy" ordering on lowest energy loss. This implementation allows prosumers to use a different comperators for their queue.

5

Results

As explained in the outputting data part (see 3.8) we are storing the output in CSV files containing the energy production, current energy status and all the deals. For gathering the results, we ran the simulations with the following settings

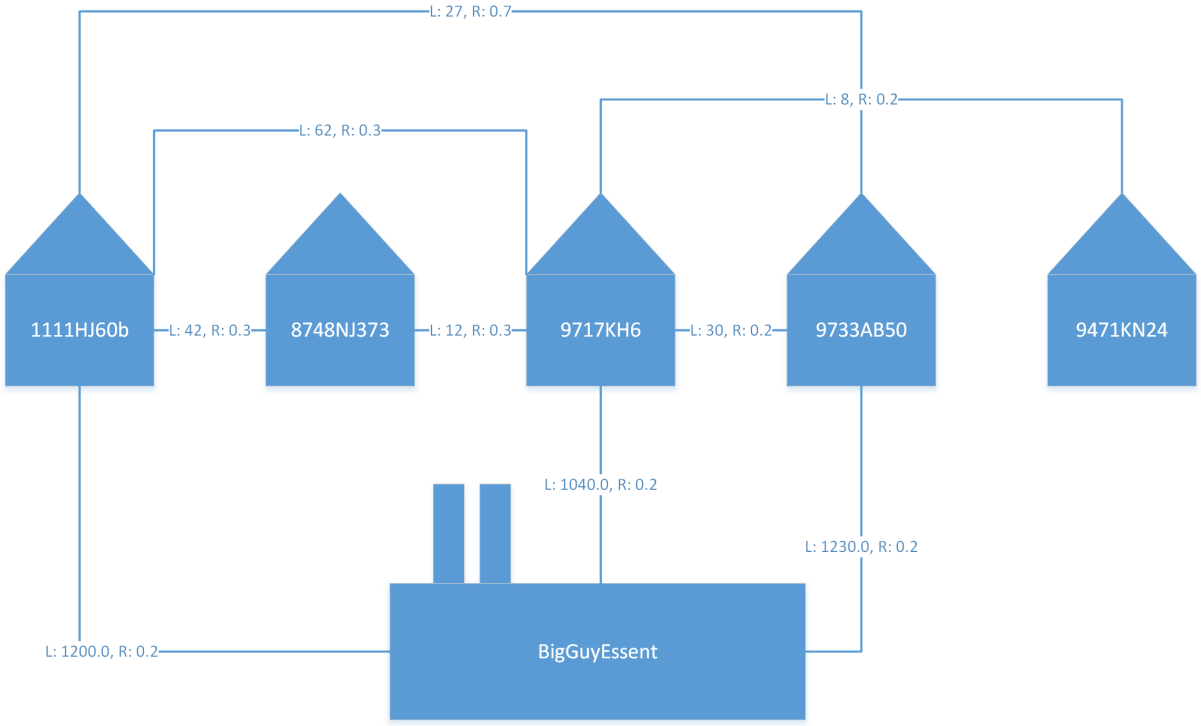
- A: A month of simulation with a speedup of 3,600 (one second is an hour) on grid A.
- B: A month of simulation with a speedup of 85,200 (one second is a day) on grid B.
- C: Six years of simulation with a speedup of 85,200 on grid B.

All the simulations use for weather the Example_KNI_Dataset dataset.

Grid A

Our first grid is a simple 5 house grid with no particular topology, it has one "big guy" with high length and low resistance cables. The houses with energy producers are 9471KN24 a mix between wind mills and solar panels, 9717KH6 a house with only windmills and 8748NJ373 a house with only solar panels.

Figure 5.1: Grid A



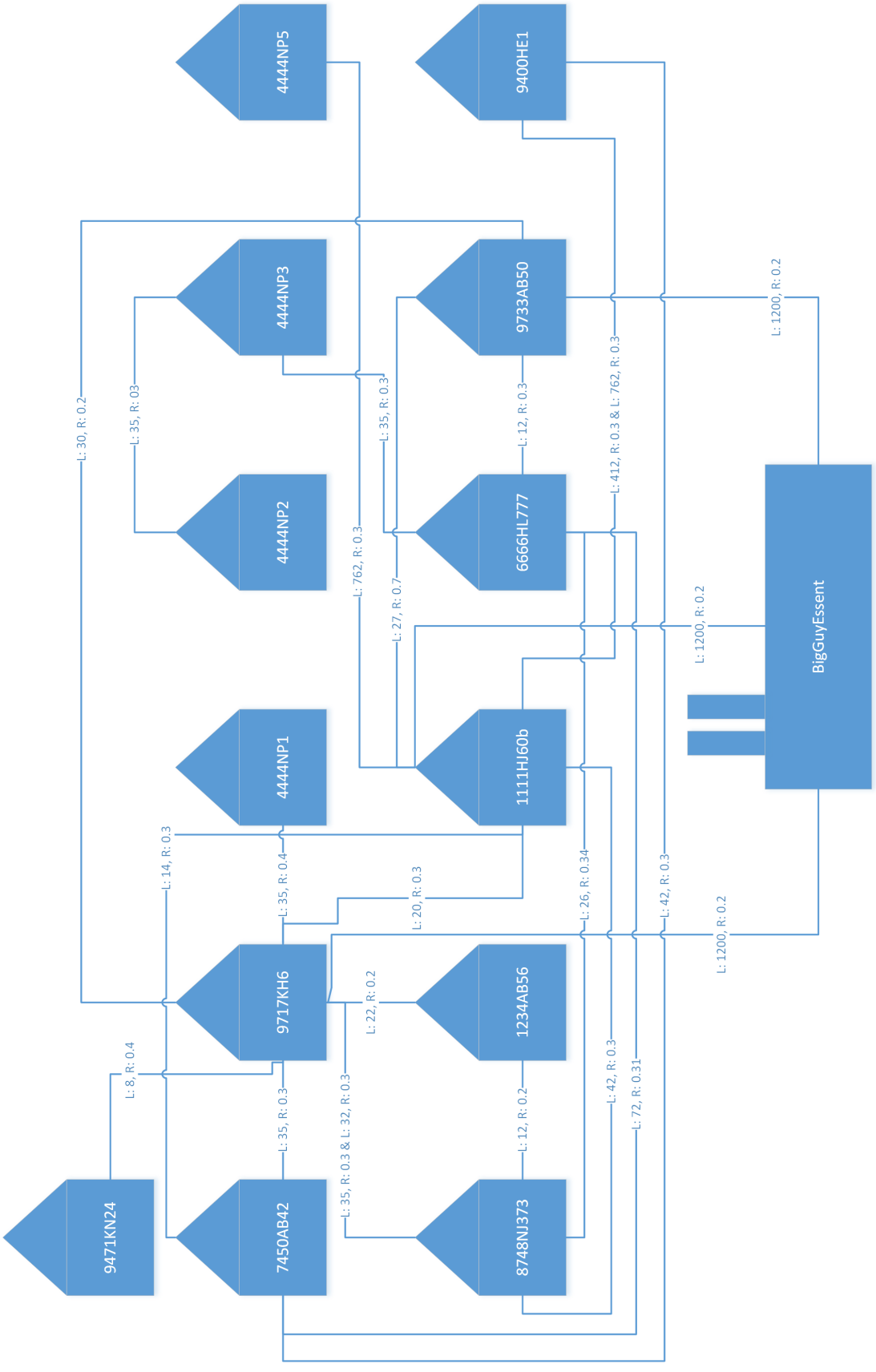
A simple grid generated by the initializer program

Grid B

Grid B has 13 houses and one big guy, again with no particular topology, it has 5 houses with energy producers:

- Two of which are a mix of solar and wind energy: 9471KN24 and 7450AB42.
- Two of which only contain solar panels: 9400HE1 a solar energy house with 2 extra large solar panels and 8748NJ373.
- And a house with only wind mills: 9717KH6.

Figure 5.2: Grid B



A more extended grid created by hand

5.1 Simulations

All simulations are results are ran on a Lenovo Thinkpad w540 with an Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz.

In most of our results you can clearly see which houses have means of generating green energy. When they produce more energy than they are consuming and you can also see that they immediately start with selling the surplus energy.

Simulation A

One month of simulation from 1-3-1995 till 31-3-1995 with a speedup of 3600 on grid A. Here you see nicely all the days pass as the weather changes since the KNI holds the weather data per day.

Prosumer	Produced (joule)	Sold (joule)	Bought (joule)
9733AB50	$-1.39 * 10^9$	0.0	$1.40 * 10^9$
9471KN24	$9.34 * 10^8$	$9.75 * 10^8$	$4.12 * 10^7$
9717KH6	$2.49 * 10^8$	$7.43 * 10^8$	$4.95 * 10^8$
8748NJ373	$1.57 * 10^9$	$1.65 * 10^9$	$7.39 * 10^7$
1111HJ60b	$-1.39 * 10^9$	0.0	$1.39 * 10^9$

The results per prosumer can be seen in figure 5.3. You can clearly distinguish the agents that have means of producing energy. Those are 9471KN24, 9717KH6, and 8748NJ373. The production and sales of all agents combined can be viewed in figure 5.4 and figure 5.5. Add the sales you see again nicely the passing of days and therefore the change in production.

Figure 5.3: Results per prosumer simulation A

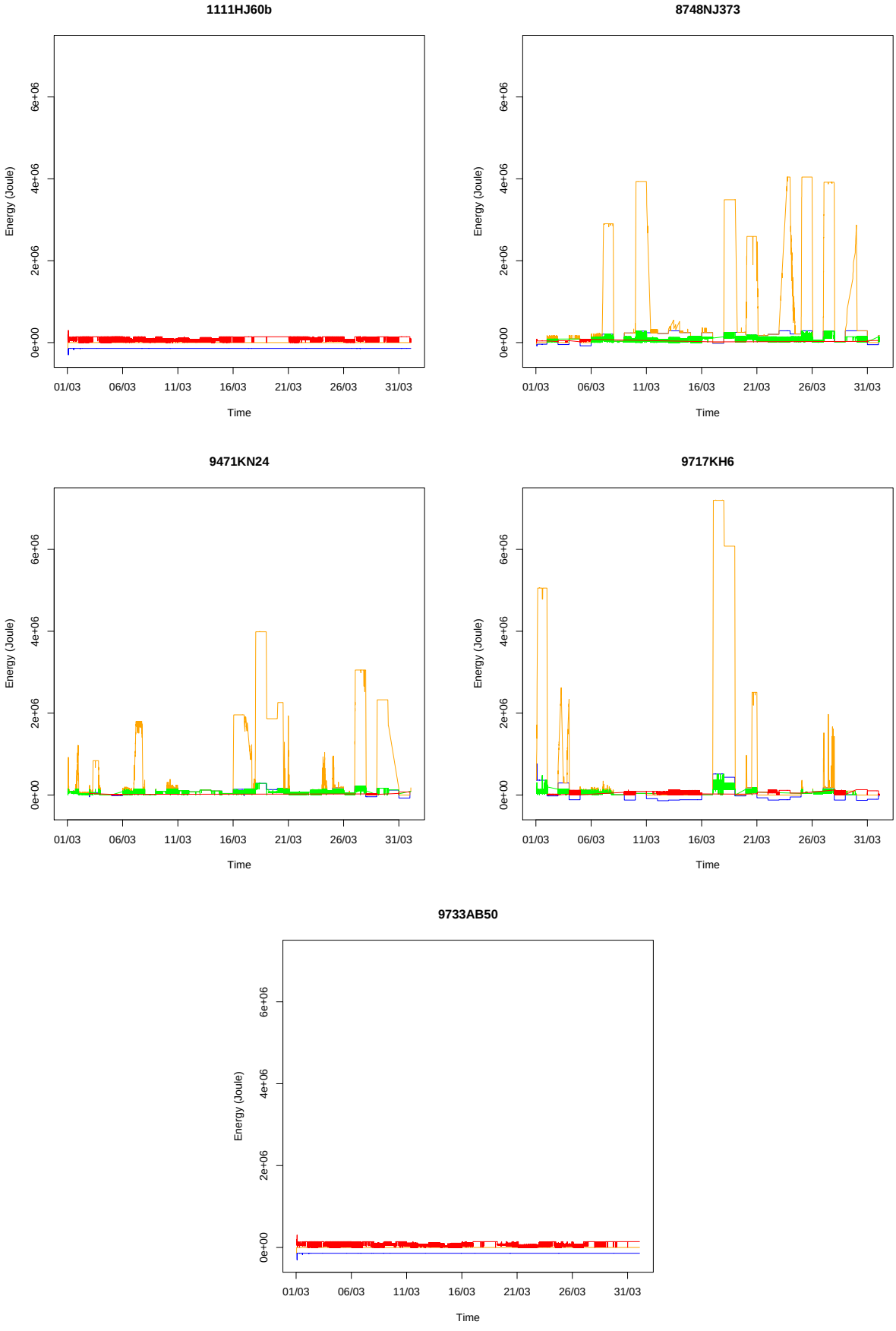


Figure 5.4:

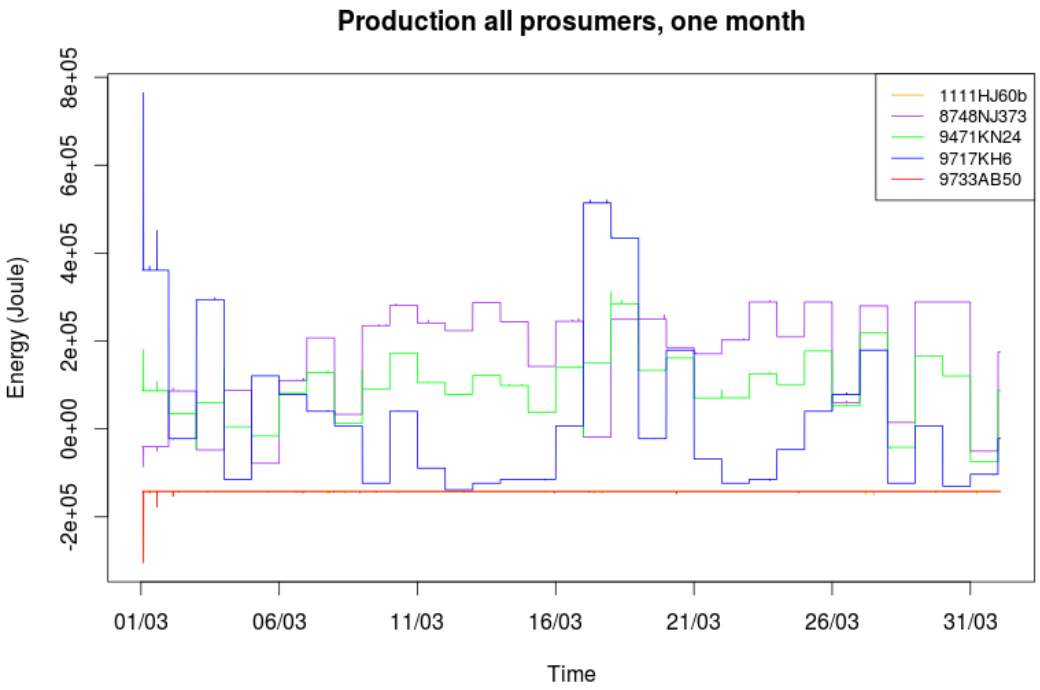
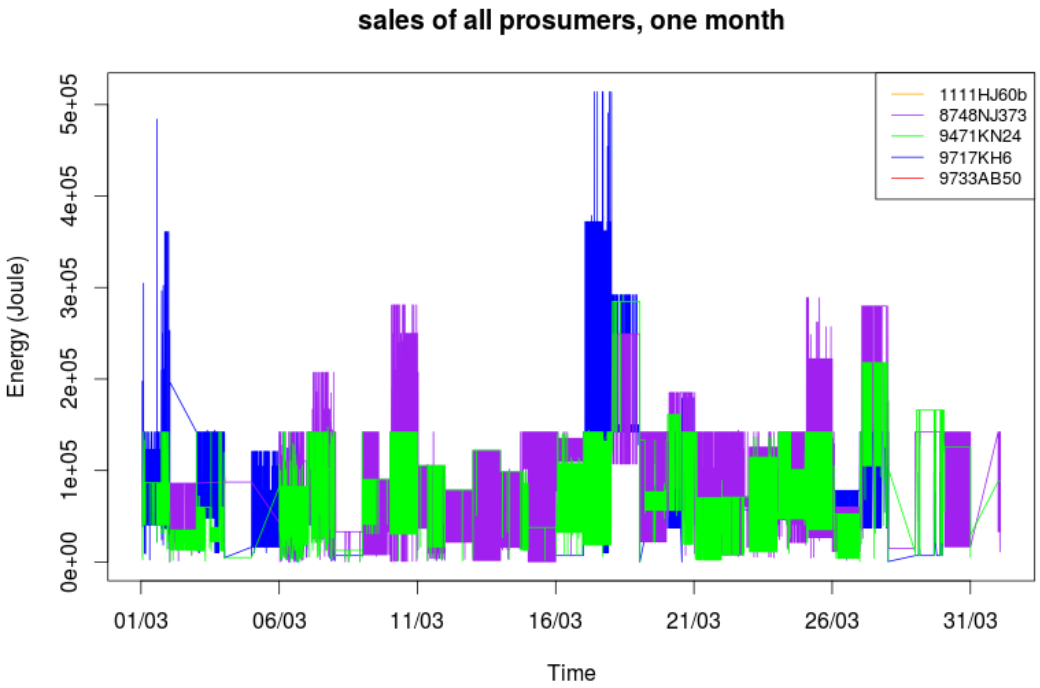


Figure 5.5:



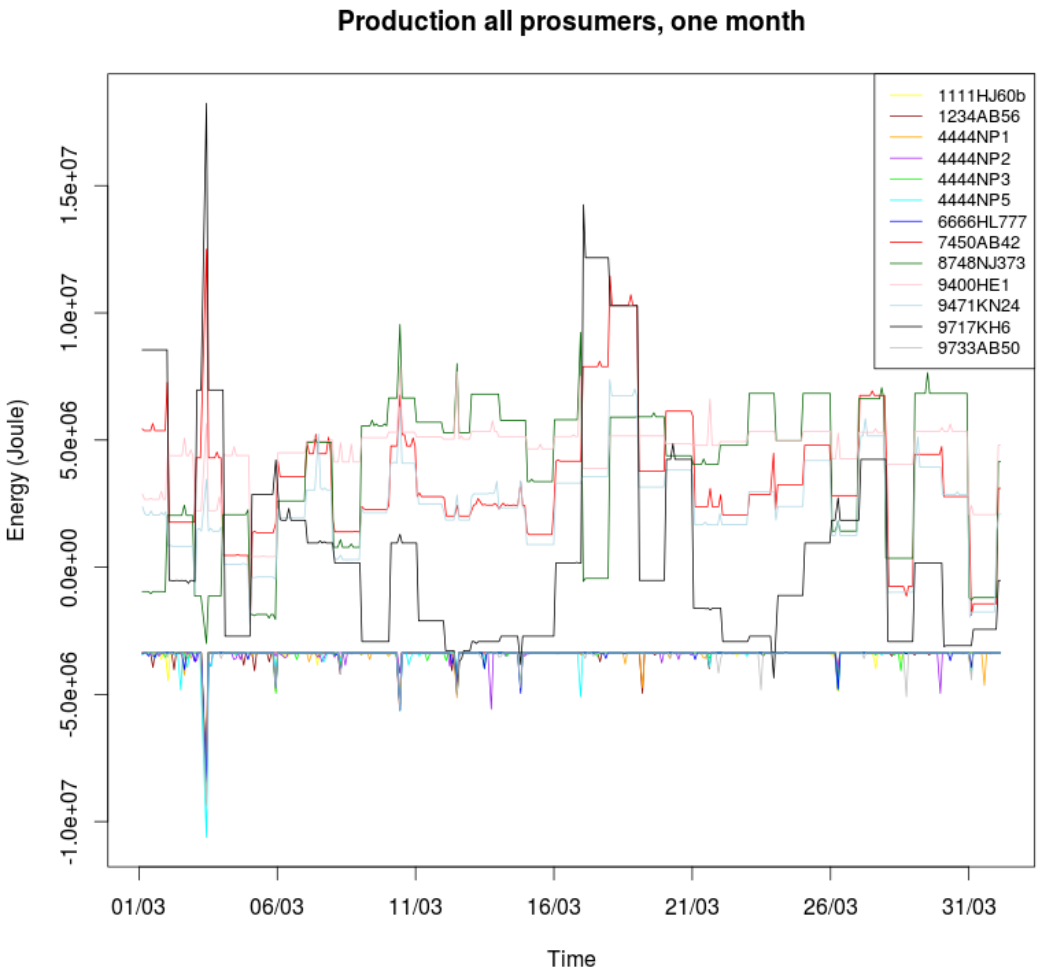
Simulation B

One month of simulation from 1-3-1995 till 31-3-1995 with a speedup of 85200 on grid B. Here you see nicely all the days pass as the weather changes. Every prosumer that doesn't have means of generating energy has a produced value of $-1.39 * 10^9$. This is exactly the same number as Simulation A. Their bought value is not everywhere equal, this is caused by the energy loss that occurs during transfer between buyer and seller. The buyer has to pay for this energy loss. You see small fluctuations in the amount energy bought for each prosumer that doesn't have means of producing energy although the amount of consumed

Prosumer	Produced (joule)	Sold (joule)	Bought (joule)
1111HJ60b	$-1.39 * 10^9$	0.0	$1.40 * 10^9$
1234AB56	$-1.39 * 10^9$	0.0	$1.43 * 10^9$
4444NP1	$-1.39 * 10^9$	0.0	$1.42 * 10^9$
4444NP2	$-1.39 * 10^9$	0.0	$1.41 * 10^9$
4444NP3	$-1.39 * 10^9$	0.0	$1.42 * 10^9$
4444NP5	$-1.39 * 10^9$	0.0	$1.41 * 10^9$
6666HL777	$-1.39 * 10^9$	0.0	$1.41 * 10^9$
7450AB42	$1.38 * 10^9$	$1.41 * 10^9$	$3.12 * 10^7$
8748NJ373	$1.57 * 10^9$	$1.64 * 10^9$	$7.60 * 10^7$
9400HE1	$1.86 * 10^9$	$1.86 * 10^9$	0.0
9471KN24	$9.33 * 10^8$	$9.74 * 10^8$	$4.30 * 10^7$
9717KH6	$2.58 * 10^8$	$7.43 * 10^8$	$5.00 * 10^8$
9733AB50	$-1.39 * 10^9$	0.0	$1.41 * 10^9$

The production plot of this simulation 5.6 shows a very similar pattern to the production plot of simulation A 5.4. The days are less consistent since there are fewer data points per day, making a small differences in simulation step time visible.

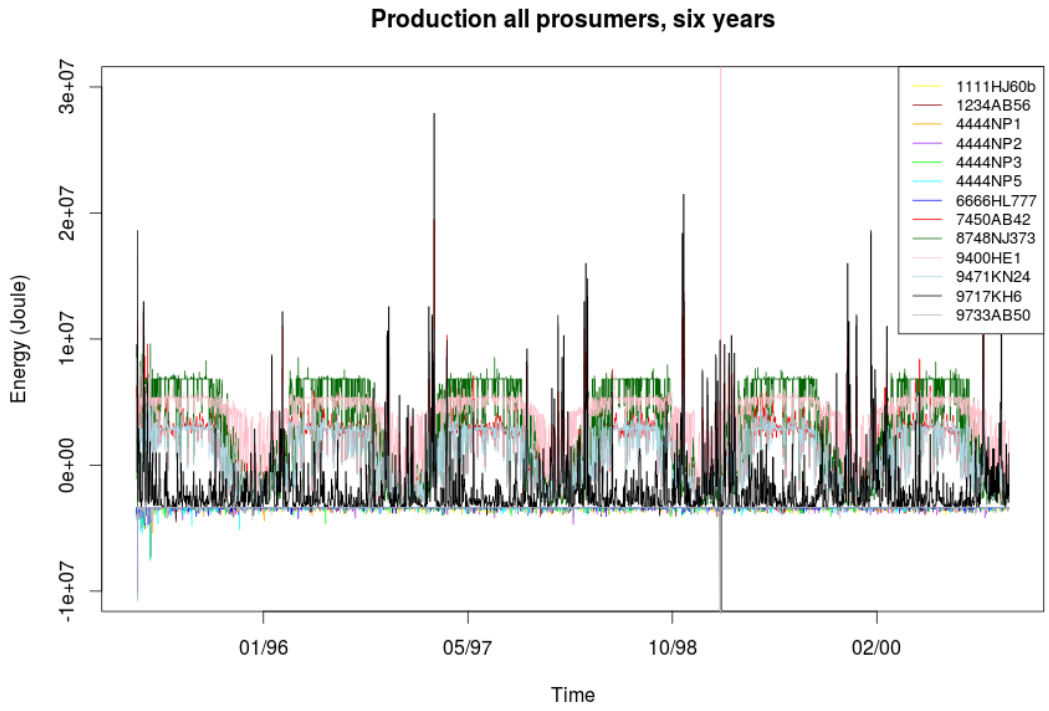
Figure 5.6:



Simulation C

Six years of simulation from 1-1-1995 till 31-12-2000 with a speedup of 85200 on grid B. You can clearly distinguish the seasonal weather that corresponds with the six years in plot 5.7. This simulation shows also that solar panels 5.8 are a more continuous source of energy than wind turbines 5.9. The advantage of wind turbines is that they produce power year round, in contrary with the solar panels that only produce decently in the summer.

Figure 5.7:



Prosumer	Produced (joule)	Sold (joule)	Bought (joule)
1111HJ60b	$-9.57 * 10^{10}$	0.0	$9.80 * 10^{10}$
1234AB56	$-9.57 * 10^{10}$	0.0	$9.83 * 10^{10}$
4444NP1	$-9.57 * 10^{10}$	0.0	$9.79 * 10^{10}$
4444NP2	$-9.57 * 10^{10}$	0.0	$9.85 * 10^{10}$
4444NP3	$-9.57 * 10^{10}$	0.0	$9.83 * 10^{10}$
4444NP5	$-9.57 * 10^{10}$	0.0	$9.77 * 10^{10}$
6666HL777	$-9.57 * 10^{10}$	0.0	$9.85 * 10^{10}$
7450AB42	$3.89 * 10^{10}$	$5.12 * 10^{10}$	$1.24 * 10^{10}$
8748NJ373	$8.00 * 10^{10}$	$9.47 * 10^{10}$	$1.49 * 10^{10}$
9400HE1	$1.04 * 10^{11}$	$1.09 * 10^{11}$	$4.53 * 10^{9}$
9471KN24	$2.55 * 10^{10}$	$4.16 * 10^{10}$	$1.63 * 10^{10}$
9717KH6	$-4.80 * 10^{10}$	$1.42 * 10^{10}$	$6.37 * 10^{10}$
9733AB50	$-9.57 * 10^{10}$	0.0	$9.80 * 10^{10}$

Figure 5.8:

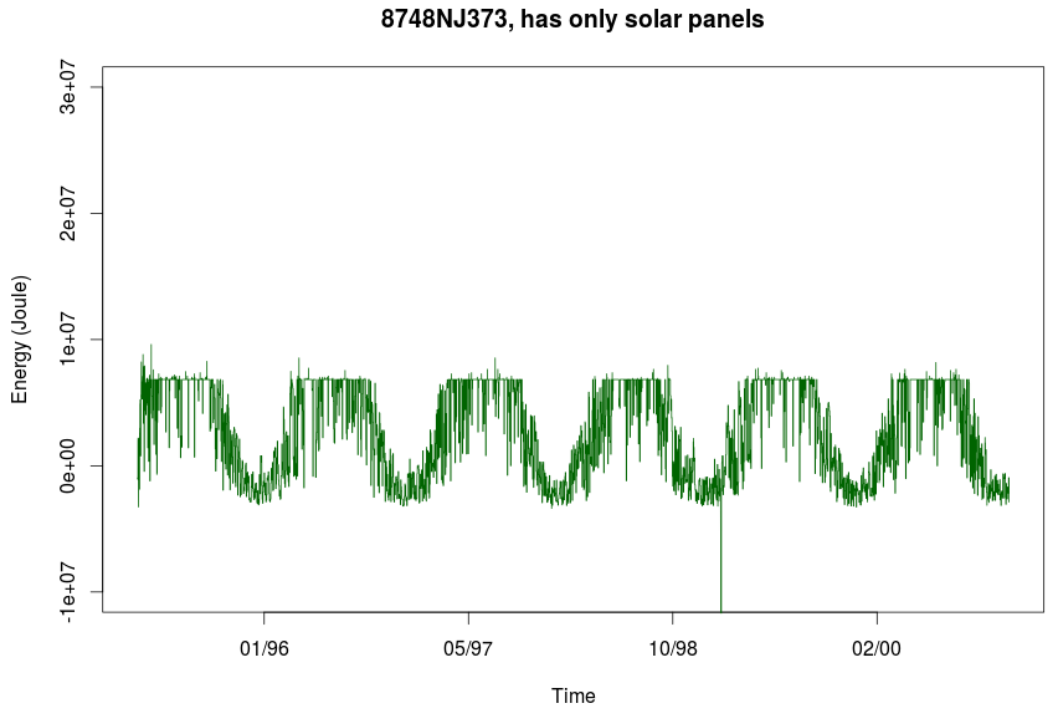
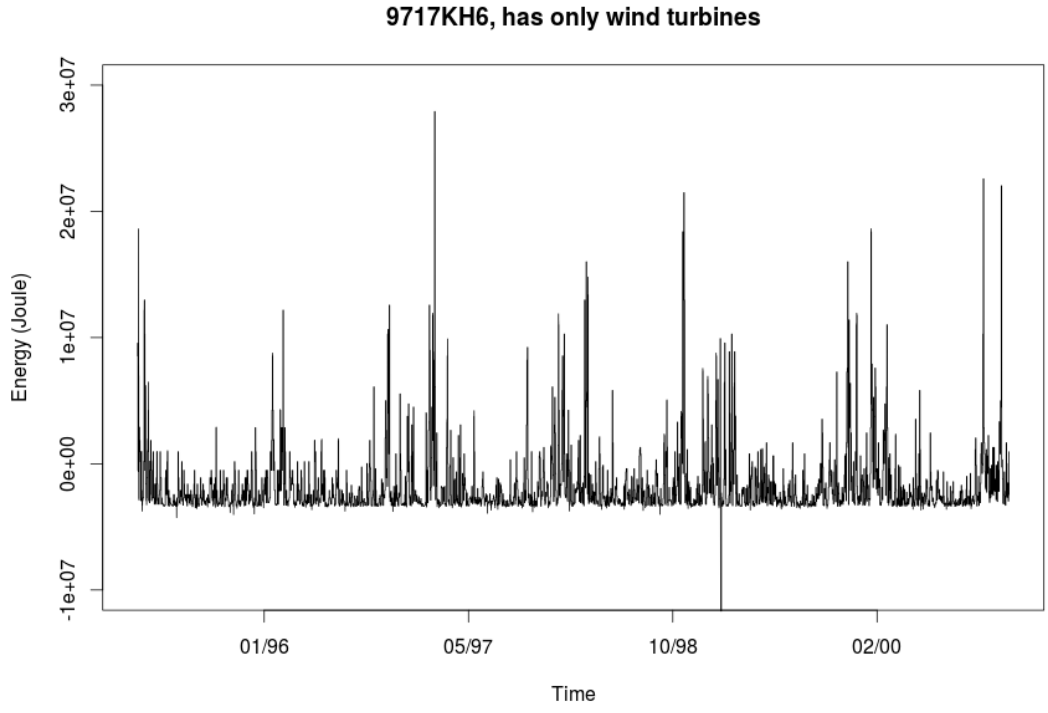


Figure 5.9:



5.2 Energy price

To test the pricing system we took grid B but doubled all the green energy producers to create a clear surplus in energy. we ran the system with the prices that can be found in table 5.2. The other houses are only consuming energy. The big guy still sold his energy for 22 cents and bought energy back for 15 cents.

Prosumer	price (€)	total profit (€)	total energy sold (joule)	profit per joule (€)
7450AB42	0.16	$5.17 * 10^9$	$3.23 * 10^{10}$	0.160
8748NJ373	0.18	$8.30 * 10^9$	$4.61 * 10^{10}$	0.180
9400HE1	0.19	$9.17 * 10^9$	$5.31 * 10^{10}$	0.173
9717KH6	0.20	$1.51 * 10^9$	$2.75 * 10^{10}$	0.156
9471KN24	0.21	4.27^9	$8.80 * 10^9$	0.172

The sales to prosumers are lower in amount of energy but occur very often. This is because you sometimes can't sell all your surplus energy to one prosumer, but you have to divide it between multiple houses. The big guy is able to buy all the energy at once, so there the transactions are larger and less often. In all the coming plots black means a deal with another prosumer and red a deal with the big guy. 7450AB42 (figure 5.10) has a very constant selling amount, due to being the lowest price and therefore the first choice. All non producing prosumers have an equal demand for energy and will try to get it in one buy. Sellers, 8748NJ373 (figure 5.11) also sells almost all its energy to other prosumers, you can see this nicely because its price is equal to the profit per joule. 9400HE1 (figure 5.12) sells a little bit less than halve of its energy to the Big guy. 9717KH (figure 5.13) and 9471KN24 (figure 5.14) sell almost all their energy to the big Guy. This can also be seen in the decrease of profit per joule at 9717KH6 and 9471KN24 in the table, since the big guy only pays € 0.15 for energy.

We used all the prices per joule instead of per kWh, this makes the energy in the simulation very expensive, but since all the decision making is based larger and smaller comparison and all the values are scaled with this same difference this doesn't influence the system.

5.3 Results independence simulations

These are the results after running multiple simulations with a different amount of solar panels and windmills in the simulations with 13 houses. Because we want to know how independent we are from the big guy, the output is from the viewpoint of a big guy and shows the amount of bought and sold energy to the big guy. This means the lowest point in the graph of the sum is the amount of energy producers with highest independence. The x-axis represent the amount of a certain energy producer and the y-axis the total amount of sold/bought energy in joule. The 3D plot is similar to the 2d plots, here x-axis goes from 0 to 30 and contains the amount of solar panels while the y-axis goes from 0 to 50 and contains the windmills, this time z represent the total amount of energy bought and sold to the big guy in joule.

To keep consistent environment over all simulations we added all the energy producers to one prosumer. In addition to this we also ran the first 22 simulations twice to average any small differences that might occur because of the concurrency of the simulation. The differences between these simulations where always smaller than 2% therefore we concluded that one simulation for the rest of the data is sufficient. Finally we chose to take a year of simulation (1995 to be exact) so to keep in mind all seasons

Figure 5.10:

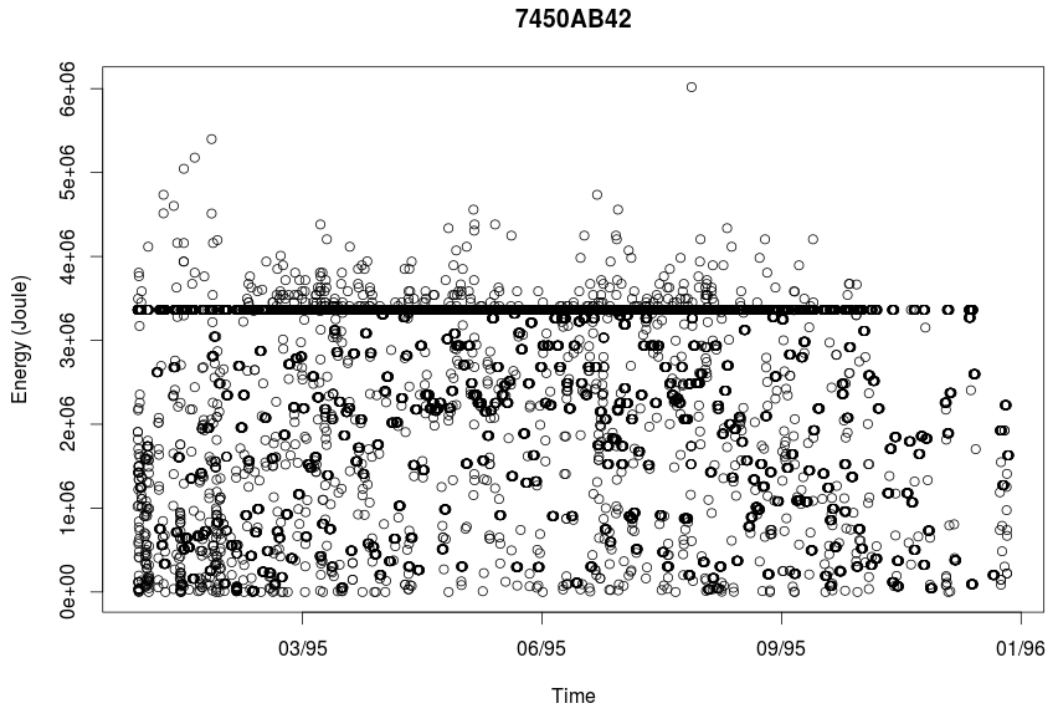


Figure 5.11:

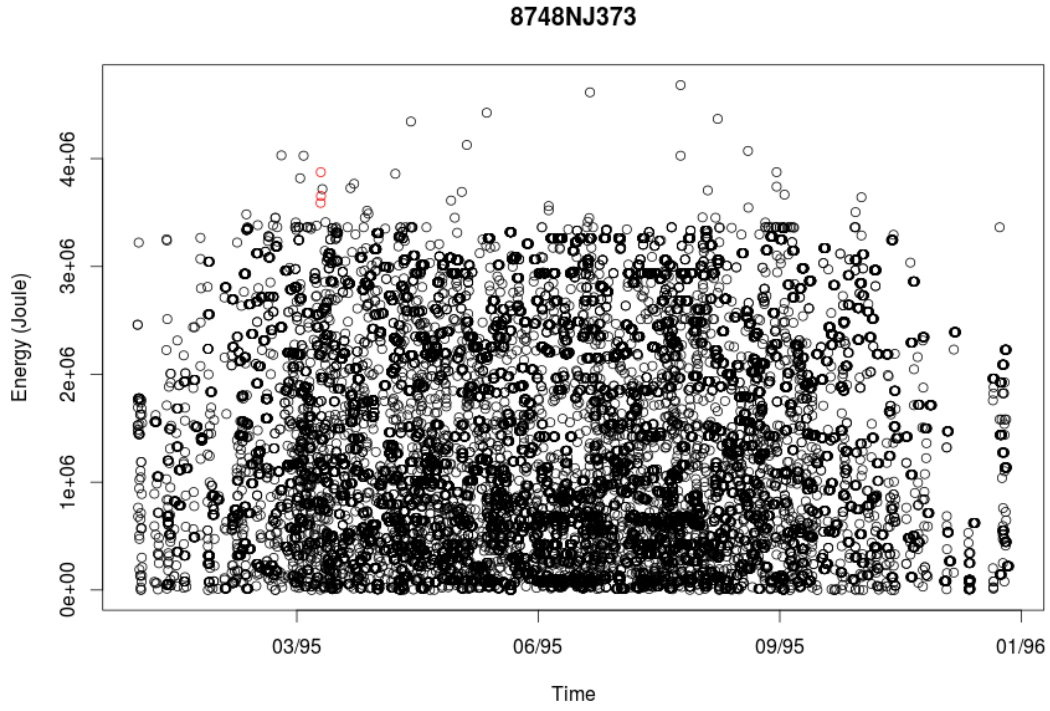


Figure 5.12:

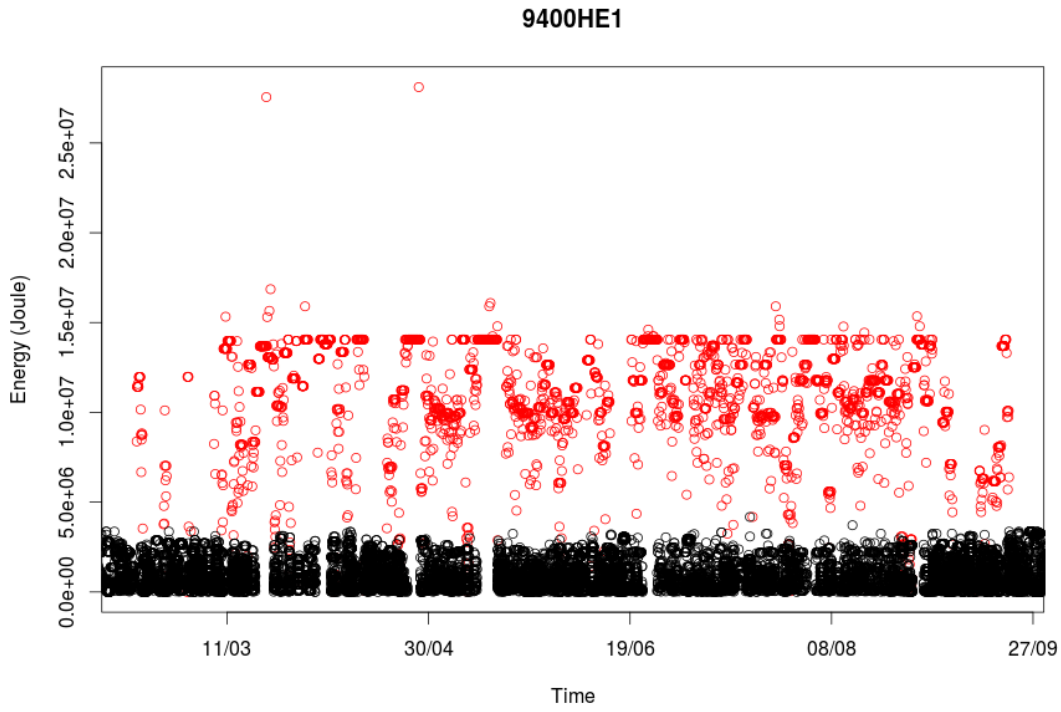


Figure 5.13:

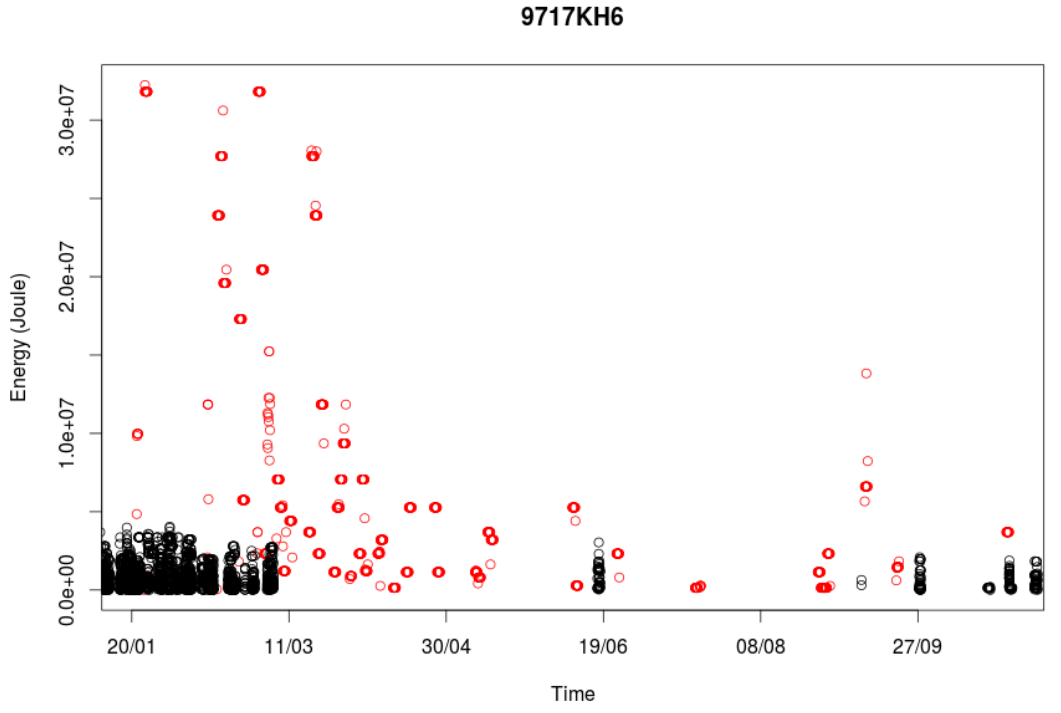
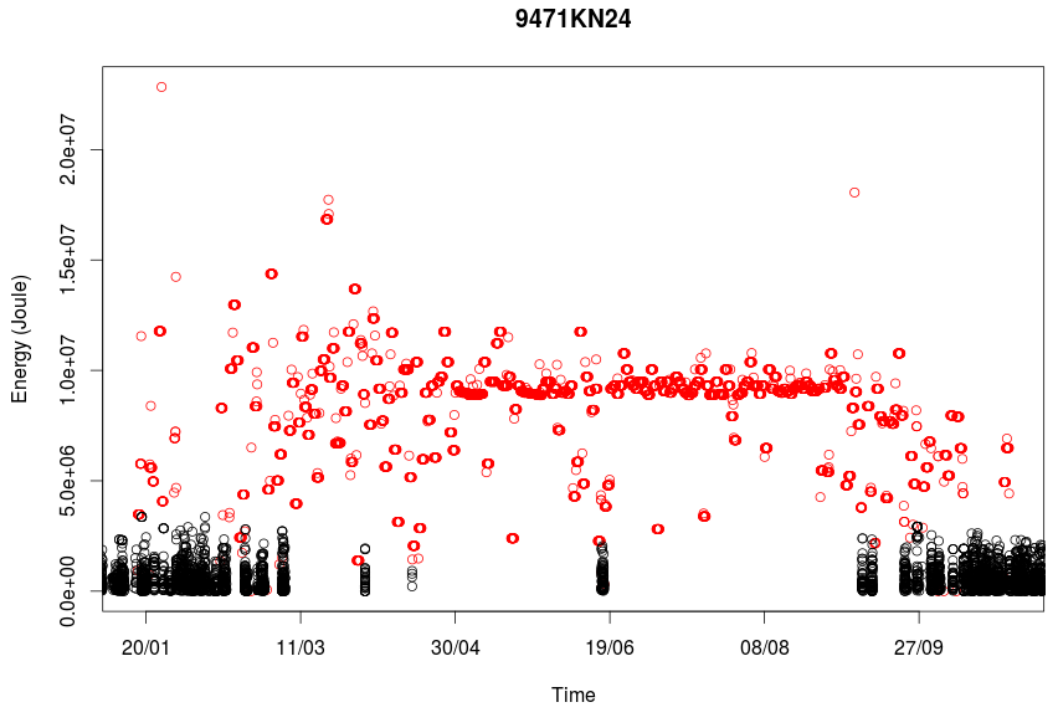


Figure 5.14:



of the year and we took a speedup of 1 second is two days (172800) in the simulation so that we wouldn't average the total to be bought and sell too much.

Here 1 solar panel has the settings:

```
"surfaceArea": 1.65,
"efficiency": 0.192,
"peakOutput": 315.0,
"qualityFactor": 0.75
```

While a windmill has settings:

```
"areaBlades": 2.269800692218625,
"maxPower": 5700.0,
"efficiency": 0.35
```

Taking information out of figure 5.3 you see that the amount of energy bought decreases as the amount of solar panels increases. This behaviour can also be seen at the windmills at figure 5.3 although less steep. With solar panels you have to start selling energy back around 20 and increasing thereafter linear. The windmill sell back looks also very linear and is initiated around 10 windmills. At the plot containing both solar panels and windmills in figure 5.3 you can clearly see that having no windmills or solar panels gives the highest dependence. If you move along the x axis you see in almost all cases until 20 solar panels a steady decrease of dependence. Increasing the amount of windmills reduces this a bit but not very drastically. A ravine can be seen in the red area giving combinations of windmills and solar panels with the most independence from the big guy.

Figure 5.15: Results simulations with only solar panels, 13 houses

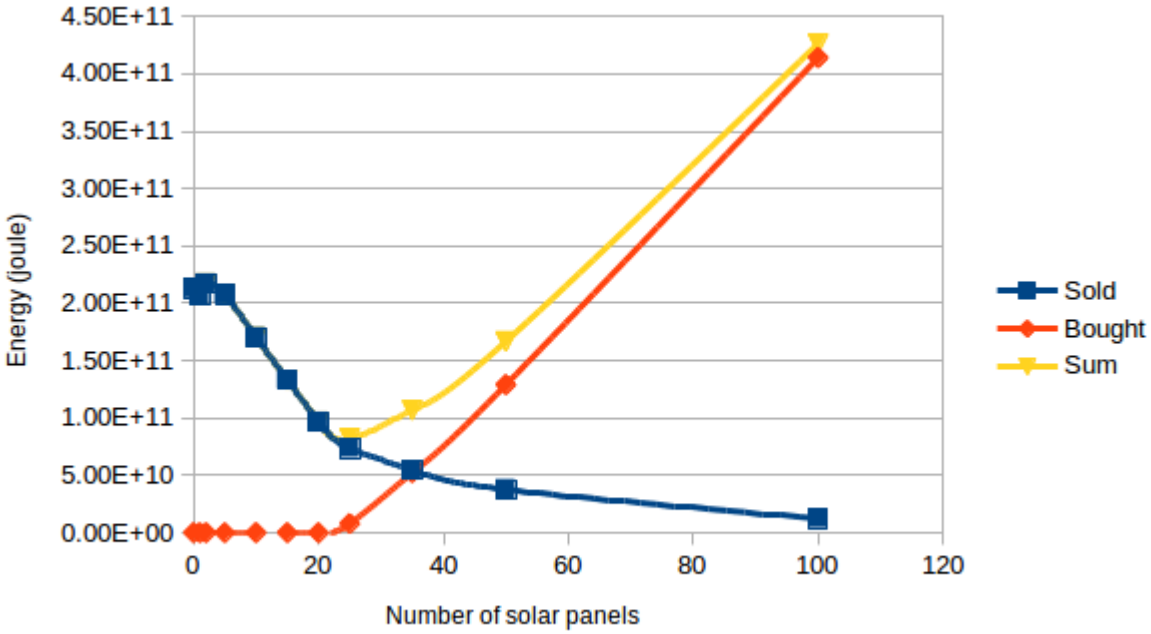


Figure 5.16: Results simulations with only windmills, 13 houses

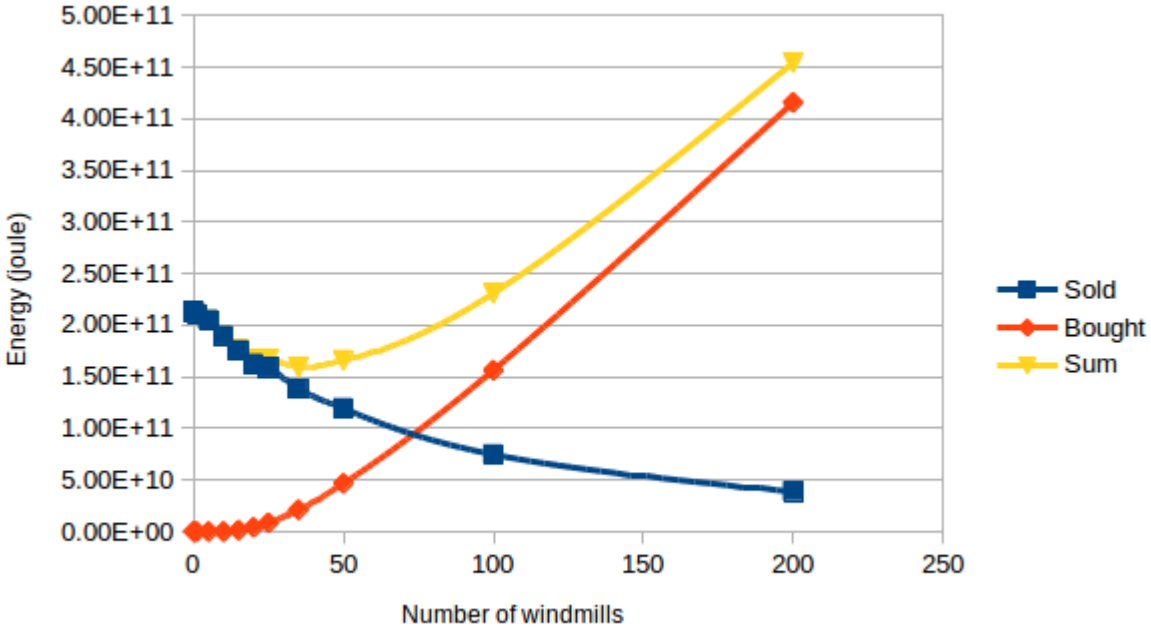
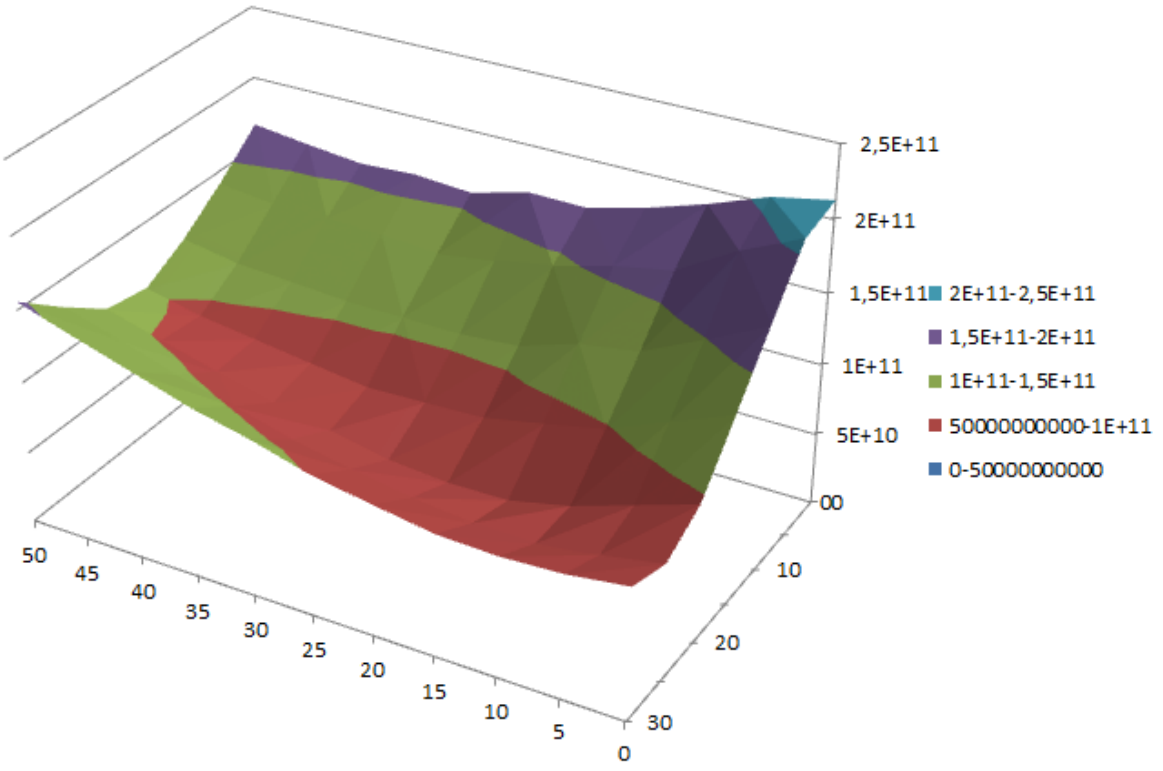


Figure 5.17: Results simulations with combination of windmills and solar panels, 13 houses



6

Conclusion

The results of our simulation confirm that, selling energy and buying energy when needed occurs in the model. The energy level remains constant, while energy is produced and sales are being made. This can be seen in figure 5.3. When the amount of energy is occasionally in excess and nothing can be sold, prosumer sell to the big energy producer. When there exists a shortage in energy, then the prosumers will buy from the energy suppliers. This has the effect of flattening the energy levels of current energy, see 8748NJ373 of figure 5.3. This shows that we are able to successfully simulate a smartgrid.

Simulating weather and energy loss works, however there is room for an expanded and more detailed system. An example of changes happening because of weather is the extra production of solar power in the summer. Figure 5.8 shows this result clearly. Figure 5.9 shows that windmills are not seasonally bound. An example of energy loss is more subtle but can be seen when looking at the raw data, energy bought is a little higher then consumed, because a house needs to buy more to compensate for the loss along the energy lines.

The priority queue where you can set the buy tactic of the prosumer works nicely, as shown in 5.2. A prosumer that asks too much money for its energy will have to sell it to the big guy and take its losses. At this example we only order on price, but since whole offers are compared you can also add more sophisticated orderings.

When trying to find out the optimal number of solar panels and windmills to be as independent as possible of the energy provider, we looked at our resulting graphs in section 5.3. Using the trends that become apparent in the solar panels graph 5.3 and that of the windmill 5.3, we can clearly see that there are signs of more independence with solar panels then with windmills, because the sum reaches a lower point. This observation is supported by figures 5.8 and 5.9. The energy produced by solar panels is more constant then that of windmills. Windmills do do produce a decent amount of energy, but this production is very inconsistent and spiky resulting in temporary high productions. This large surplus must then be sold to the energy company resulting in less independence.

Looking at a combination of both solar panels and windmills. The 3d plot in figure 5.3 shows more independence than solar panels alone. Also a ravine can be spotted going through the plot holding ideal combinations between solar panels and wind mills. The lowest point in this ravine resides at 20 solar panels and 20 windmills. A combination between solar panels and windmills flourish, because solar panels perform only decently in the summer and the windmills can counter some of the needed energy of the winter. If you would acquire 20 solar panels and 20 windmills every house still needs to buy 996.5 kWh per year from the energy provider and 254 kWh has to be sold back.

To put these results into perspective with the other results, when looking at solar panels alone we needed about 2 solar panels per house and were left with 1573 kwh per house to buy and 171 kwh per house to sell every year. For windmills only we needed 2.7 windmills a house and had 2950 kwh to buy and 451 kwh to sell per house per year.

Finally, The goal to make the simulation run in parallel is achieved. The MAS balances the simulation nicely. Each agent uses at max one thread of the cpu. When the simulation is ran with grid A there are only have five prosumers, resulting in an average cpu (4 cores 8 threads) load of 80%. When grid B is ran the 13 prosumers use 100%. The multi-thread requirement is achieved.

7

Future work

Our simulation considers many basic elements and some detailed elements, like energy loss. There is still a lot of room for improvement and extension. Some interesting additions or improvements to the simulation are the following ones:

- Batteries that are able to store energy. This is a very realistic addition, for instance, Tesla is currently already manufacturing them [18]. It allows even more independence from the big guy, but also adds a lot more complexity to the system and simulation. You would need a prediction of the coming energy and then decide if you would want to store your energy or sell it for the current price, which involves advanced market behaviours and strategies.

  Adding Batteries will also dramatically increases the potential of windmills, since they are inconsistent but do produce a decent amount of energy. These inconsistencies can than be absorbed by the batteries.

- Dynamically selling (and storing) energy based on market price, as explained in the previous addition would increase the complexity of the system a lot, because of different strategies and price calculations [19]. However this would also mean that prosumers will be able to sell their own energy at a competitive price.

- Advanced dynamic energy consumption based on for example different parameters like time and temperature. This would make the simulation more realistic, for example in our system we assume that a house consumes the same amount of energy in the summer and in the winter. This is not realistic and will probably influence the results significantly. Two approaches can be taken here, one is calculating a general consumption pattern for certain houses and applying that as we currently are using the wheater data. The second approach would be to model every individual significant electronic device in a house. The heating for example could then also use the weather data to calculate how much energy it needs to heat the house.

Bibliography

[1] Ang Sha and Marco Aiello. A novel strategy for optimising decentralised energy exchange for prosumers. *Energies*, 9(7):185–205, 2016.

[2] Generalmicrogrids. Micro grids. `https://www.generalmicrogrids.com/about-microgrids`. [Online; accessed 6-July-2017].

[3] W. Shi, N. Li, C. C. Chu, and R. Gadh. Real-time energy management in microgrids. *IEEE Transactions on Smart Grid*, 8(1):228–238, Jan 2017.

[4] D. E. Olivares, A. Mehrizi-Sani, A. H. Etemadi, C. A. Cañizares, R. Iravani, M. Kazerani, A. H. Hajimiragha, O. Gomis-Bellmunt, M. Saeedifard, R. Palma-Behnke, G. A. Jiménez-Estévez, and N. D. Hatziargyriou. Trends in microgrid control. *IEEE Transactions on Smart Grid*, 5(4):1905–1919, July 2014.

[5] S. Karnouskos and T. N. d. Holanda. Simulation of a smart grid city with software agents. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, pages 424–429, Nov 2009.

[6] Nicola Capodieci, Giuliano Andrea Pagani, Giacomo Cabri, and Marco Aiello. An adaptive agent-based system for deregulated smart grids. *Service Oriented Computing and Applications*, 10(2):185–205, 2016.

[7] Jade. Jade Tilab. `http://jade.tilab.com`. [Online; accessed 15-June-2017].

[8] Jade API. Agent. `http://jade.tilab.com/doc/api/jade/core/Agent.html`. [Online; accessed 13-June-2017].

[9] Giovanni Cair. JADE tutorial. `http://www.agilemethod.csie.ncu.edu.tw/download/agent/JADETutorial.pdf`. [Online; accessed 15-June-2017].

[10] Jade API. Behaviour. `http://jade.tilab.com/doc/api/jade/core/behaviours/Behaviour.html`. [Online; accessed 15-June-2017].

[11] Jade API. DFService. `http://jade.tilab.com/doc/api/jade/domain/DFService.html`. [Online; accessed 15-June-2017].

[12] Energieleveranciers. Gemiddeld energieverbruik. https://www.energieleveranciers.nl/gemiddeld-energieverbruik. [Online; accessed 13-June-2017].

[13] Thomas H. Cormen, Charles E. Leiserson, Ronald Rivest, and Clifford Stein. *Introduction to algorithms*. Mit Press Ltd, 3rd revised edition edition, 2009.

[14] K. Grogg. Harvesting the Wind: The Physicsof Wind Turbines. *In Physics and Astronomy Comps Papers*, 2005.

[15] How to calculate the annual solar energy output of a photovoltaic system. http://photovoltaic-software.com/PV-solar-energy-calculation.php/. [Online; accessed 13-June-2017].

[16] Curt Harting. AC Transmission Line Losses. http://large.stanford.edu/courses/2010/ph240/harting1/. [Online; accessed 14-June-2017].

[17] KwH price. http://www.energiesite.nl/begrippen/kwh-prijs/. [Online; accessed 15-June-2017].

[18] Tesla powerwall. https://www.tesla.com/nl_NL/powerwall. [Online; accessed 3-Juli-2017].

[19] G. A. Pagani and M. Aiello. Generating realistic dynamic prices and services for the smart grid. *IEEE Systems Journal*, 9(1):191–198, 2014.