ULBS
Universitatea "Lucian Blaga" din Sibiu

"Lucian Blaga" University of Sibiu – Romania

Faculty of Engineering

Department of Computer Science,
Electrical and Electronics Engineering

Dissertation

Scientific advisor: Prof. Dr. Ing. Lucian Vinţan
Coordinator: Dr. Ing. Arpad Gellert

Graduate:

Stefan Feilmeier
Embedded Systems

- Sibiu, 2015 -

ULBS
Universitatea "Lucian Blaga" din Sibiu

"Lucian Blaga" University of Sibiu – Romania

Faculty of Engineering

Department of Computer Science,
Electrical and Electronics Engineering

Loads management based on Photovoltaic and Energy Storage System

Scientific advisor: Prof. Dr. Ing. Lucian Vințan
Coordinator: Dr. Ing. Arpad Gellert

Graduate:

Stefan Feilmeier
Embedded Systems

- Sibiu, 2015 -

Table of Contents

Abstract 5

I. Introduction 7

I.1 "Energiewende": Energy transition in Germany 7

I.2 Purpose of the thesis 9

I.3 The company FENECON GmbH & Co. KG 10

I.4 Example installation and data 11

II. Concept 14

II.1 Energy sources, availability and costs 14

II.1.1 Photovoltaics electricity production 14

II.1.2 Energy Storage System (ESS) 17

II.1.3 Power grid 18

II.1.4 Energy source management 18

II.2 Loads management and Added Value 20

II.2.1 Loads management 20

II.2.2 Electrical loads and Added Value 21

II.2.3 Feed-in tariff 22

II.3 Proposed architecture 23

III. Theory 26

III.1 State of the art 26

III.1.1 Prediction of photovoltaic energy production 26

III.1.2 Prediction of electricity consumption 28

III.1.3 Conclusion 28

III.2 Prediction using Machine Learning 29

III.2.1 Artificial Neural Networks 29

III.2.2 Time-Series prediction 32

III.2.3 Implementation of generic predictors 33

III.2.4 Encog machine learning library 34

III.2.5 Optimization using FADSE 35

III.2.6 Development toolchain 36

IV. Implementation 38
IV.1 Implementation of the development toolchain 38
IV.1.1 Definition of the optimization problem 38
IV.1.2 Implementation of FemsPredictorSimulator 39
IV.2 Implementation of the energy management architecture 41
IV.2.1 Basic types: Value, Prediction, Timestamp and Field 41
IV.2.2 EssListener and ProHybridSimulator 41
IV.2.3 Prediction Agent and Predictor 42
IV.2.4 Source and Consumption Prediction Agents 42
IV.2.5 Load Agent 42
IV.2.6 Scheduler Agent 43
IV.3 FEMS and FENECON Online-Monitoring 44
IV.3.1 FEMS Hardware 44
IV.3.2 FEMS Software 44
IV.3.3 FENECON Online-Monitoring 48
IV.4 Source Code 48
V. Result 50
V.1 Development toolchain for Predictors 50
V.1.1 Creation of individuals 50
V.1.2 Training of Multi-Layer perceptron networks 51
V.1.3 Evaluation of individuals 51
V.2 Simulation of proposed architecture 53
V.2.1 Predictor performance 54
V.2.2 Scheduler Agent 56
V.3 Management of real loads 57
V.4 Next steps 58
VI. Conclusion 60
References 61
List of illustrations 65
List of equations 67
List of abbreviations 67
Annexes 68

Abstract

"Urgent and concrete action is needed to address climate change" – this statement starts the chapter about "Climate Change, Energy, and Environment" in the final declaration of the G7 Summit 2015 Declaration (G7 2015). The seven major advanced economies recognize, that the global dependency on fossil fuels and its effect on climate change and global security is one of the biggest threats of our time.

With the German energy transition ("Energiewende"), followed by subsidy programs in many other countries, a lot of effort and money was put in the development of alternative forms of electricity production using renewable sources like wind and solar.

Today photovoltaics (PV) installations enable the decentralized production of electricity from solar at competitive costs in many places around the world, but innovative ideas are required to overcome the fluctuating nature of renewable sources. With properly designed PV installations, only up to 30 % (Quaschning, Weniger and Tjaden 2012) of the produced electricity can be self-consumed. Reasonably sized energy storage systems based on batteries increase this ratio up to 70 % (SolarServer). To further increase it, a more intelligent approach is required.

This dissertation proposes a new approach for an energy management system that optimizes the local, decentralized production and consumption of electricity, based on an energy storage system and a previously developed basic energy monitoring and management system ("FEMS").

The proposal uses prediction techniques to estimate the future electricity production and consumption. It also takes into account the Added Value of available, manageable loads, in order to establish an energy management schedule. Furthermore a basic toolchain is proposed to support the development and optimization process. Finally a first implementation of the software is provided to evaluate the feasibility of the approach.

The document is structured in six chapters:

Chapter I. "Introduction" (from page 7) describes the evolution of the renewable energy market and how the recent changes are enabling new technologies. It continues with an introduction to the purpose of this thesis. The chapter finishes with a brief overview on the company behind this paper and a presentation of the photovoltaics installation example.

Chapter II. "Concept" (from page 14) discusses, what can be done to optimize the production and consumption of electricity in a scenario with photovoltaics and an energy storage system. It concludes with a proposed architecture for an intelligent energy management system.

Chapter III. "Theory" (from page 26) gives an overview on the state of the art in the fields that are influencing the proposed system. It continues with a description of general functionality and optimization of the applied machine learning system for predictions and finishes with a detailed view on the development toolchain.

Chapter IV. "Implementation" (from page 38) discusses the realization of the proposed architecture. It also presents the "FENECON Energy Management System" (FEMS) device as well as the "FENECON Online-Monitoring", which were designed and developed by the author prior to this paper, and shows, how the proposed system can enhance the existing functionality to control external devices.

Chapter V. "Result" (from page 50) presents the results of the development toolchain and the simulation of the proposed energy management architecture. It also gives a preview of the expected results when managing real loads and suggests next steps for further studies on the subject.

The paper finishes with a "Conclusion" in Chapter VI. (from page 60).

I. Introduction

"Urgent and concrete action is needed to address climate change" – this statement starts the chapter about "Climate Change, Energy, and Environment" in the final declaration of the G7 Summit 2015 Declaration (G7 2015). The leaders of seven major advanced economies, including United States President Barack Obama and German Chancellor Angela Merkel, recognize that "decarbonisation of the global economy" is required to keep the "increase in global average temperature below 2 °C".

But the global dependency on fossil fuels is not only responsible for global warming, but also for destabilization and conflicts in countries with oil and gas resources. Michael Klares prediction in his book from 2002, "Resource wars will become, in the years ahead, the most distinctive feature of the global security environment" (Klare 2002) proved correct in numerous conflicts like in Iraq, Sudan, Syria, South China Sea and more.

I.1 "Energiewende": Energy transition in Germany

Many governments and organizations agree nowadays, that the future of electricity production must be decentralized, using renewable energy sources.

Germany started its first subsidy program for renewable power production in 1991 (Bundesanzeiger 1990), followed by the extremely effective Renewable Energy Act ("Erneuerbare-Energien-Gesetz", EEG) in the year 2000 (Bundesanzeiger 2000), which was revised several times in the meantime. As "Energiewende", German for "energy transition", it became a role model for many similar laws in other countries.

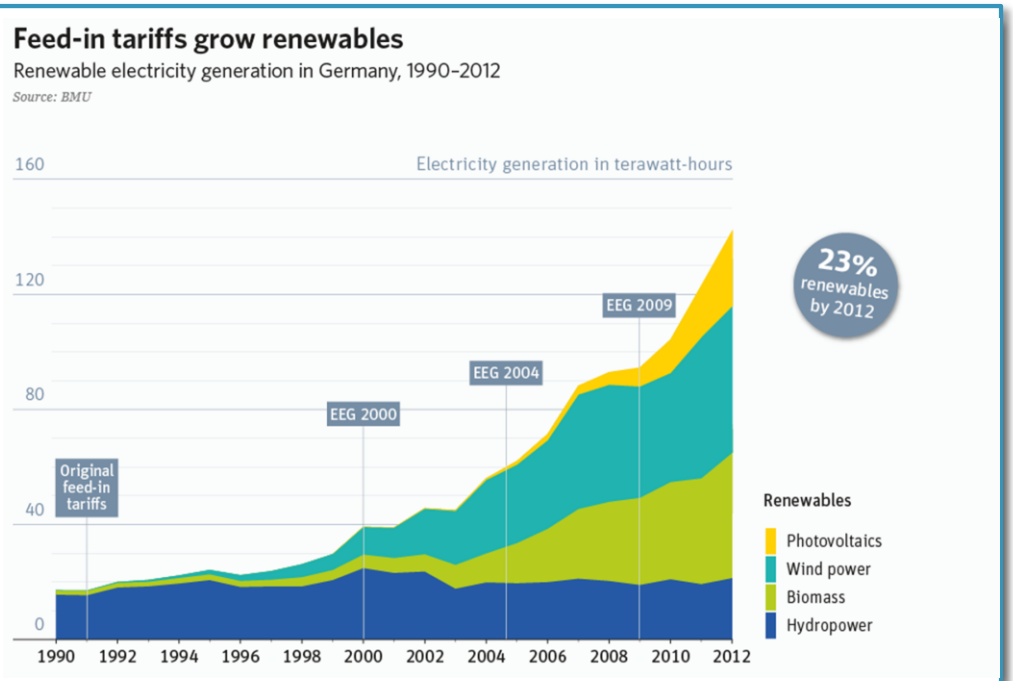


Figure I.1-1 Development of electricity generation from renewable souces in Germany (Heinrich Böll Foundation 2012)

The development of electricity generation in Germany since 1990, visualized in Figure I.1-1, is remarkable, but comes at a high price for the German industry and population: the electricity prices are among the highest in the world. Figure I.1-2 shows, that the household price per kWh including taxes in some European countries like Romania is only about one third of the price in Germany.

While this is an obvious challenge for the whole German economy, it effectively enabled a new era of electricity production. Accompanied by constantly decreasing investment prices for renewable electricity generators like wind-mills and photovoltaics (PV) systems (see Figure I.1-3), Germany was one of the first countries in the world to reach

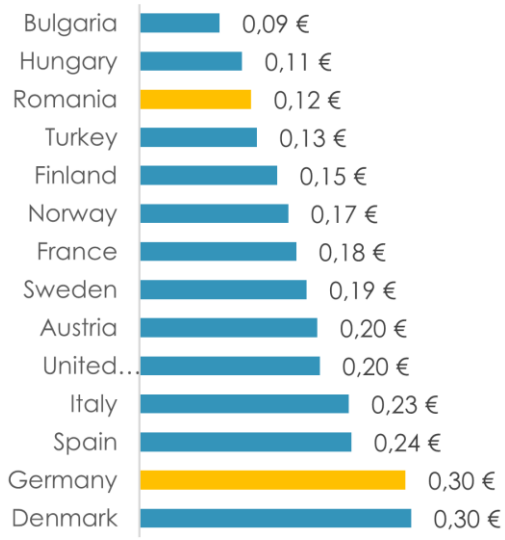


Figure I.1-2 Household-prices per kWh of electricity including taxes in a selection of European countries in 2014 (Eurostat 2015)

"grid-parity" for photovoltaics plants in 2011 (Fraunhofer Institute for Solar Energy Systems ISE 2015).

"Grid-parity" describes the fact that the Levelized Cost of Electricity (LCoE) for alternative energy is less than or equal to the purchasing price from the power grid. Consequently, everybody who has access to an appropriate area, like a roof-top can, produce his own electricity in a small, decentralized installation, for less cost per kWh as if he bought it from the grid.

This is certainly only valid for energy that is consumed directly at the moment when it is produced. Because of this and due to the volatile nature of photovoltaics, new innovative solutions need to be found to optimize the production, consumption and storage of electricity.

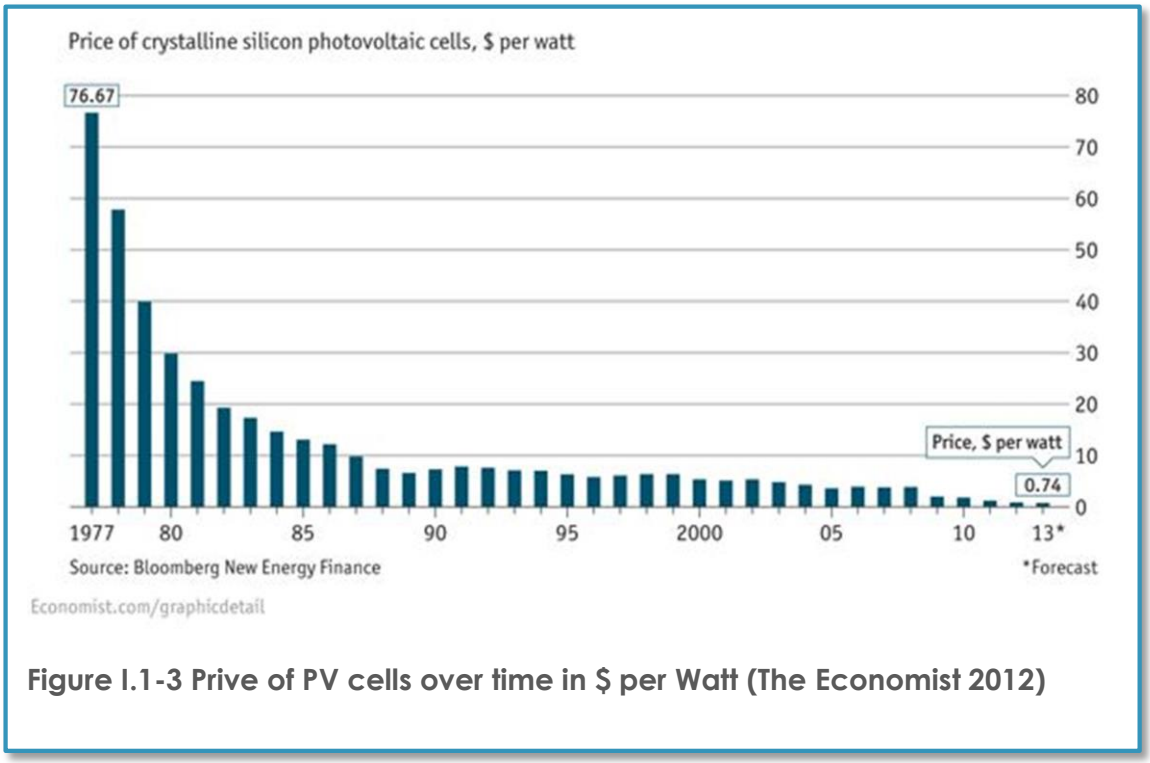


Figure I.1-3 Prive of PV cells over time in $ per Watt (The Economist 2012)

I.2 Purpose of the thesis

This paper is influenced by and applied to the market reality in south Germany, where solar is the most important renewable energy source. A few years ago, when investments were largely driven by state subsidies, the single purpose of photovoltaics installations was to sell a maximum of electricity to the grid, in order to gain high revenue. It was the time, when photovoltaics installations were built exactly facing south in a 30° angle to the ground, no matter how expensive and complex the basement construction needed to be, for the single reason to maximize total annual power production.

With rising electricity prices, decreasing investment costs and decreasing state subsidies, the industry is now changing. The new target is to adjust and synchronize energy production and consumption in order to increase the self-consumption ratio, to reduce strain on the power grid and to optimize the overall economic usage of renewable energy. Figure I.2-1 shows the electricity production curve of PV systems facing half east and half west compared to only south. While the overall production is less with east/west, this production pattern is much closer to the usual electricity consumption.

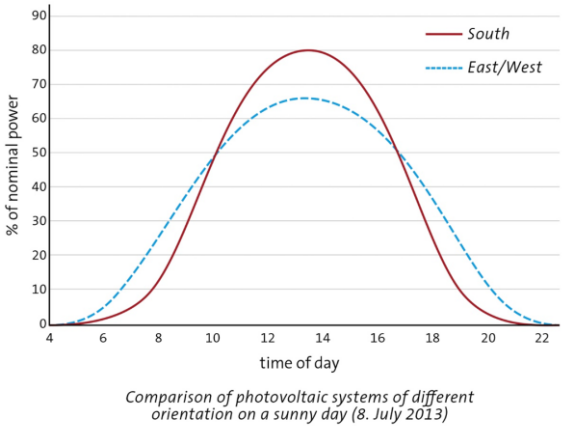


Figure I.2-1 Comparing PV systems directing East/West and South (Renusol GmbH 2014)

Statistics show, that self-consumption ratios up to 30 % can be achieved with properly designed photovoltaics installations (Quaschning, Weniger and Tjaden 2012). This ratio may be increased up to 70 % (SolarServer) with economically reasonably sized energy storage systems. To further increase it, a more intelligent approach that is considering both the energy production and consumption side needs to be implemented.

The purpose of this thesis is to propose a software architecture for an energy management system that optimizes the local, decentralized production and consumption of electricity. The proposal uses prediction techniques to estimate the future electricity production and consumption. It also takes into account the Added Value of available, manageable loads, in order to establish an energy management Schedule. Furthermore a basic toolchain is proposed to support the development and optimization process. Finally a first implementation of the software is provided to evaluate the feasibility of the approach.

I.3 The company FENECON GmbH & Co. KG

In 2011, after the Fukushima Daiichi disaster in Japan showed again that it is impossible to completely eliminate the risk for catastrophes in nuclear power plants, the German government decided to successively shut down all remaining nuclear power plants in the country until 2022.

In the same year FENECON was founded with the target to revolutionize the way electricity is produced and consumed. The company was built on the idea of decentralized, renewable energy production using photovoltaics, energy efficiency using led lighting and energy storage using battery based energy storage systems. Having its headquarter in Deggendorf, Bavaria, FENECON is placed in one of the most vibrant areas for photovoltaics in the world[1].

Figure I.3-1 FENECON Logo

In a close partnership with the Chinese corporation BYD Company Limited ("Build Your Dreams"), the world-biggest manufacturer of lithium-based batteries with 187.000 employees and an annual turnover of 9,2 billion USD (FENECON 2015), technologically advanced energy storage systems were developed and fitted to German market.

"FENECON by BYD" energy storage systems are now covering a wide range of applications from small systems for single-family houses to medium sized systems for companies to big solutions for industrial and communal areas and grid backup.

In 2015, the company has 25 employees, with a high ratio of university grade engineers.

I.4 Example installation and data

The theoretical concept and proposed implementation discussed in this paper is applied to an example installation. It is consisting of a 12,24 kWp photovoltaics installation and a "FENECON by BYD PRO Hybrid" energy storage system with 8,5 kWh usable capacity, which is supplying the electricity for a private household.

Figure I.4-1 shows the utility room in the house with the energy storage system on the right side, connected to an early FEMS prototype in the middle. The laptop on the left displays the current state of the installation using FENECON Online-Monitoring. It also shows numerous electrical meters and devices, which are not required for the operation of the system, but were added for testing and development purposes. They are not related to this thesis.

[1] "The southern German state of Bavaria, population 12.5 million, has three photovoltaic panels per resident, which adds up to more installed solar capacity than in the entire United States." (Curry 2013)

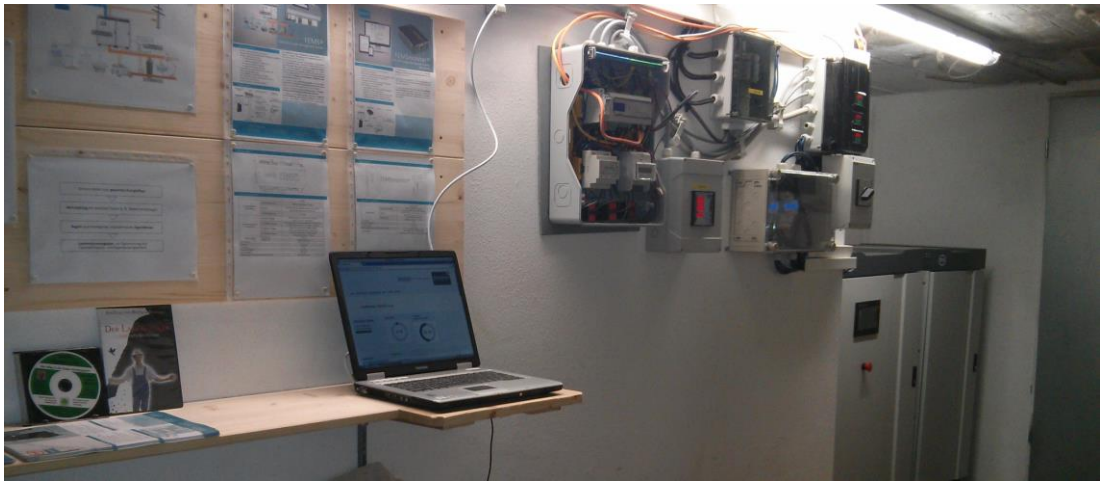


Figure I.4-1 Example installation: FENECON by BYD PRO Hybrid with FEMS

Figure I.4-2 shows a schematic plan of the energy storage system in this example installation. It is connected with two PV systems (PV 1 and PV 2), three phases for connection to the power grid (Grid Ph 1, Grid Ph 2 and Grid Ph 3) and three phases for loads (Ph 1, Ph 2 and Ph 3). All data used for calculations and experiments in this paper is related to this example installation and uses this naming convention.

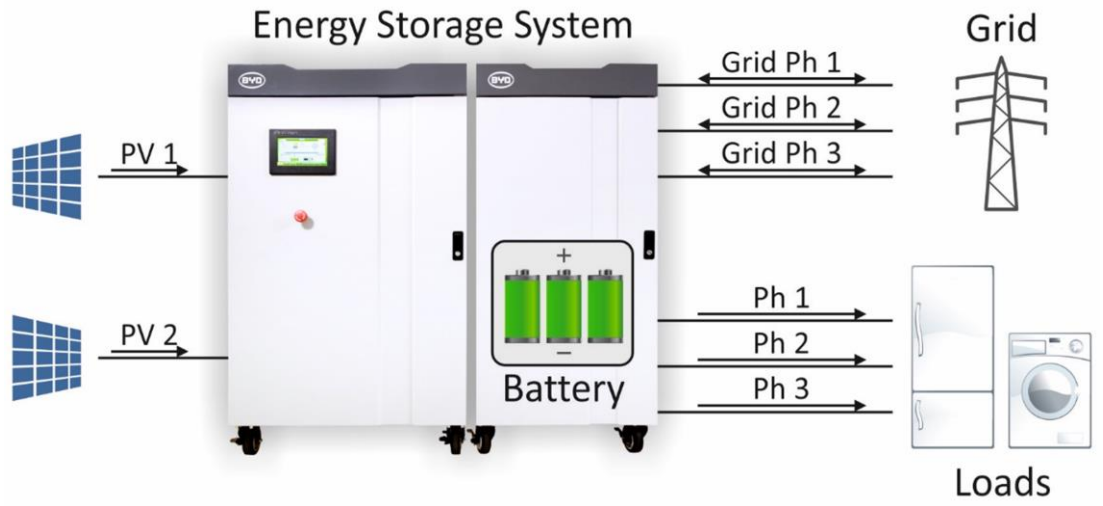


Figure I.4-2 Schema Energy Storage System

The source data is covering the timespan from 1st January to 31st May 2015. FEMS was used to record the data from the energy storage system, to apply the energy management and visualize final results in Chapter V.

Figure I.4-3 shows a small subset of the available data that was used as input to calculations and predictions. Each record is representing a five minutes average value and is consisting of a timestamp, standardized to seconds since 1st January 1970, and columns for each production (PV1, PV2) and consumption (Ph1, Ph2, Ph3) field in Watt as used by the proposed architecture[1].

	A	B	C	D	E	F
1	timestamp	PV1	PV2	Ph1	Ph2	Ph3
1313	1420632600	486,00	463,60	150,00	201,80	277,40
1314	1420632900	447,00	417,40	59,20	192,80	250,60
1315	1420633200	442,40	417,20	45,20	181,60	57,40
1316	1420633500	437,00	407,20	49,60	184,60	152,80
1317	1420633800	409,40	377,40	0,00	183,60	178,00
1318	1420634100	399,20	367,00	0,00	182,40	214,20

Figure I.4-3 Example data recorded by FEMS

[1] See Chapter II.3 "Proposed architecture" on page 23

II. Concept

As stated in the introduction, the investment costs for electricity generators using renewable energy sources, especially photovoltaics, decreased substantially during the last years. This chapter starts with a brief overview on how this changed the situation of electricity production using photovoltaics, how it enabled new technologies like energy storage systems and what can be done to further optimize the production and consumption of electricity in such an environment. It concludes with a proposition of an architecture for an intelligent energy management system.

II.1 Energy sources, availability and costs

When buying electricity from a power grid, the price per kWh is usually fixed over a long period of time, like for a month or a year. Many renewable energy sources are not available at a continuous, constant level, so consequently the cost per consumed kWh is not fixed anymore. Instead, it is highly dependent on the actual mix of electricity from photovoltaics (Chapter II.1.1), energy storage system (Chapter II.1.2) and power grid (Chapter II.1.3) at a very specific point in time.

II.1.1 Photovoltaics electricity production

Photovoltaics (PV) is the most direct transformation of solar energy into electricity known today. It is based on the photovoltaic effect that causes materials to create voltage or electric current upon exposure to light, which is then exhibited using thin slices of semiconductor materials, known as wafers. Upon establishment of a closed electric circuit, direct current (DC) becomes available. To use it for standard electric devices or to feed it into the public power grid, it has to be converted to alternating current (AC) using an inverter.

Wafers are the basis for solar cells, which are composed to photovoltaic modules. A photovoltaics system usually consists of several to hundreds of PV modules, being quantified by the nominal output power in watt (W) during Standard Test Conditions (STC). The unit is colloquially referred to as “watt-peak” (Wp) or more commonly “kilowatt-peak” (kWp).

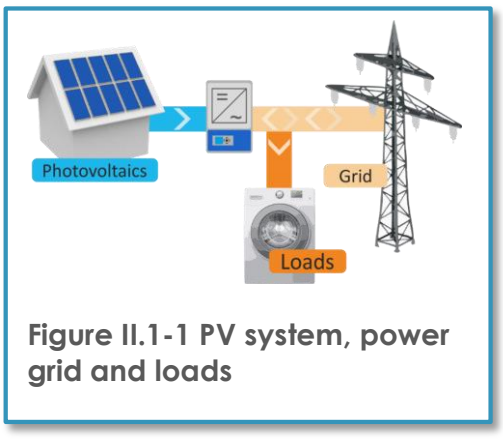


Figure II.1-1 PV system, power grid and loads

STC define a set of conditions like a specific light intensity of 1000 W/m², a cell temperature of 25 °C, and others (TÜV SÜD 2009). This is necessary, because the photovoltaic effect, and as such the total resulting power, is highly depending on the module's exposure to light and the cell temperature. Consequently, peak power generation is usually around midday, it declines considerably during cloudy or rainy weather or when the PV module is covered with shadow or snow and drops to zero during night. Figure II.1-2 shows a typical PV production curve for two systems (PV1 and PV 2) for a sunny day recorded by FEMS[1].

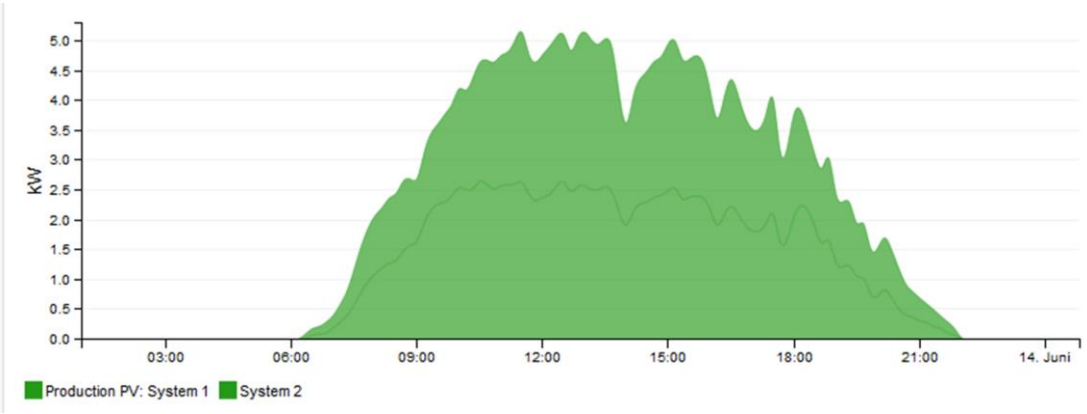


Figure II.1-2 Typical PV production curve recorded by FEMS

In recent years, the cost for photovoltaic systems decreased enormously and in many countries achieved "grid-parity" (European Commission 2013). That is, the Levelized Cost of Electricity (LCoE) from photovoltaics is less than or equal to the purchasing price from power grid.

Based on realistic average prices Equation II.1-1 shows the calculation of the total life-cycle cost (LCC) of a complete photovoltaic system with regard to the example installation from Chapter I.4. The LCoE per kWh of

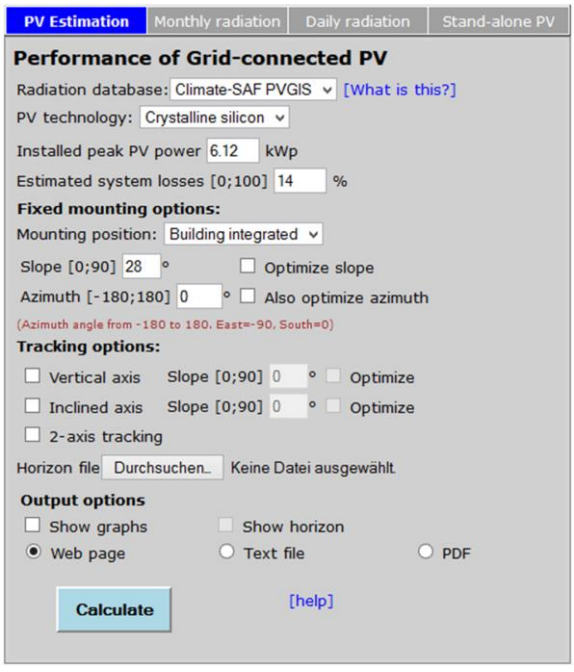


Figure II.1-3 PV potential estimation utility

[1] See Chapter IV.3.3 "FENECON Online-Monitoring" on page 48

this PV installation is calculated by depreciating the LCC over the expected lifespan and considering the expected total output power.

The expected production per year can be estimated using the "PV potential estimation utility" (Figure II.1-3) as part of the publically available "Photovoltaic Geographical Information System" (PVGIS)[1].

Estimated photovoltaics electricity production

Approximate location (latitude/longitude)		48,6/13,1
1st PV system		
Installed peak power		6,12 kWp
Azimuth		0° (South)
Slope		28°
Expected production from PVGIS	+	5.840 kWh/year
2nd PV system		
Installed peak power		6,12 kWp
Azimuth		0° (South)
Slope		15°
Expected production from PVGIS	+	5.660 kWh/year
Total expected production	=	11.500 kWh/year

Photovoltaics total life-cycle cost

Turnkey photovoltaics purchasing price		1.666 €/kWp
Total installed peak power	*	12,24 kWp
Total purchasing price	+ =	20.391,84 €
Maintenance and insurance		200 €/year
Expected life span	*	20 years
Total maintenance cost	+ =	4.000 €
Total LCC	=	20.391,84 €

Cost per produced kWh

Total LCC		20.391,84 €
Expected life span	/	20 years
LCC per year	=	1.019,59 €/year
Total expected production	/	11.500 kWh/year
Cost per produced kWh	=	8,87 €Ct./kWh

Equation II.1-1 Cost per produced kWh from photovoltaics

The example shows, that for the described PV system, the cost to produce one kWh of electricity can be approximated to 8,87 €Ct. This is only about 30 % of the applicable grid price of 29,81 €Ct/kWh in Germany in 2015[2], but it is

[1] "Photovoltaic Geographical Information System" (PVGIS) is provided by the European Commission, Joint Research Centre, Institute for Environment and Sustainability, Renewable Energies Unit at http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php

[2] See Chapter II.1.3 "Power grid" on page 18

important to understand, that this cost per kWh is only valid for energy that is consumed directly at the moment when it is produced.

If electricity cannot be used directly, it needs to be stored (Chapter II.1.2 "Energy Storage System (ESS)") or fed into the grid (Chapter II.1.3 "Power grid"). If neither of those options is possible, the PV system will reduce its power or turn off the production completely.

II.1.2 Energy Storage System (ESS)

With the growing importance of alternative power production, it becomes vital to overcome the fluctuating nature of renewable energy sources. The idea is to keep excessive energy, which cannot be immediately consumed, in storage systems for later usage. The technologies are categorized according to their storage period from short-termed, like flywheels, supercapacitors or batteries, to seasonal, like power-to-gas, pumped-storage hydroelectricity and others.

Figure II.1-4 PV system, energy storage system, power grid and loads

While many different approaches for storing energy are discussed and being developed, only few of them do currently fulfil the requirements for small, decentralized installations with minimum maintenance and reasonable economic efficiency. One of those few approaches is the lithium-ion battery technology, as used in the energy storage systems by FENECON[1].

At the example site, a FENECON by BYD PRO Hybrid 9-10[2] energy storage system is installed. It provides a usable storage capacity of 8,5 kWh with a guaranteed number of 6.000 cycles until a remaining capacity of 80 %. This equates to an average of 300 cycles per year for an expected life span of 20 years. The inverter and charger efficiencies are specified as $\eta_{inverter} = 93\%$ and $\eta_{charger} = 97{,}3\%$, which results in a round-trip efficiency of $\eta = 90{,}5\%$ per charged kWh.

[1] See Chapter I.3 "The company FENECON GmbH & Co. KG" on page 10
[2] The "Datasheet: FENECON by BYD PRO Hybrid" is attached to this document as Annex A. on page 70

II.1.3 Power grid

It is generally expected that any household, industrial site or other consumer is connected to an electricity supplier via a public power grid. In most countries, this power grid is constantly available with high reliability and electricity can be purchased for a static price that is usually fixed over a long period of time, like for a month or a year.

In Germany, being a country with electricity prices way above average European prices[1], in 2015 the price per one kWh is 29,81 €Ct for medium size households and companies (Eurostat 2015). As electricity production costs differ more and more throughout the day and year because of the ongoing transition to renewable sources, flexible tariff models with varying electricity prices are planned and may be put in place within the next years.

In a scenario with photovoltaics and energy storage system a power grid is not ultimately required and only serves as a backup for the times when the local, decentralized electricity production is not sufficient.

It is also possible to sell excess power from the PV system to the electricity supplier. From the perspective of an energy management system, this can be considered a load and is further discussed in the "Loads management" subchapter II.2.3 "Feed-in tariff".

II.1.4 Energy source management

Technically advanced energy storage systems are located between the energy production units, the power grid and the electric loads, so that every energy flow is managed by the system (Figure II.1-5).

[1] See Figure I.1-2 "Household-prices per kWh of electricity including taxes in a selection of European countries in 2014 (Eurostat 2015)" on page 8

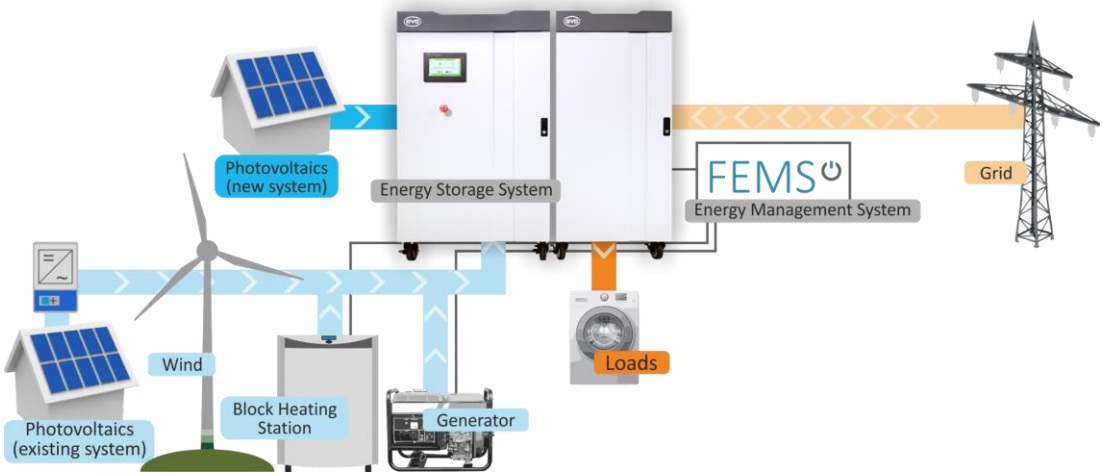


Figure II.1-5 PV system and additional energy sources, energy storage system, power grid and loads, in combination with an energy management system

At any specific time, the system is combining an energy mix, consisting of the currently available self-produced electricity, electricity that is available from a storage system and electricity obtained from the power grid. The basic prioritization is handled by the energy storage system in close communication with its Battery Management System (BMS) and is generally not supposed to be controlled from an external management system.

It is however possible to influence the internal behavior by actively adding and removing further energy production units, like diesel generators or block heating devices. An effective energy management system needs to be able to optimize the production from available electricity sources and to estimate the valid LCoE of each kWh with regard to the current energy mix. Therefore, the proposed architecture maintains a prioritization of energy sources according to their specific Levelized Cost of Electricity, as visualized in Figure II.1-6.

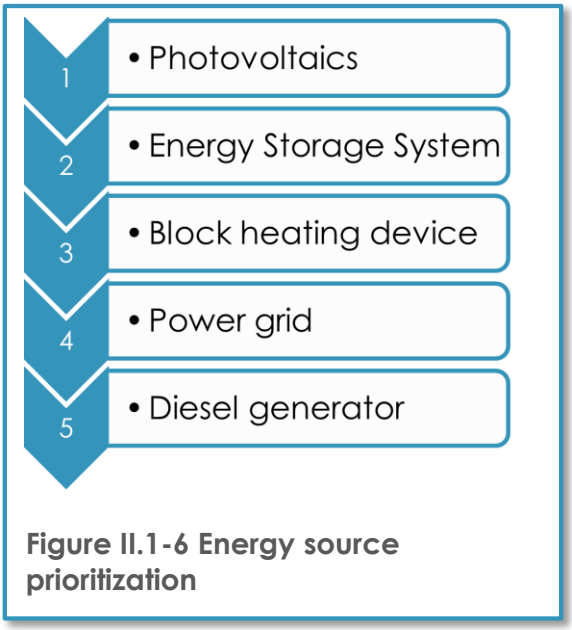


Figure II.1-6 Energy source prioritization

II.2 Loads management and Added Value

In many cases the energy consumption curve shows very different patterns compared to the photovoltaic production. While the PV production curve is typically smooth with a peak around noon[1], the consumption is highly dependent on individual parameters. Private households for example often show peaks in the morning and evening hours, as in the example in Figure II.2-1, which shows the load consumption curve on three phases (Ph1, Ph2 and Ph3) recorded by FEMS[2].

This chapter describes the idea of loads management with the target of matching the production and consumption curves[3] and proposes the "Added Value" as an effective measure for evaluating the usefulness of a certain load at a specific point in time.

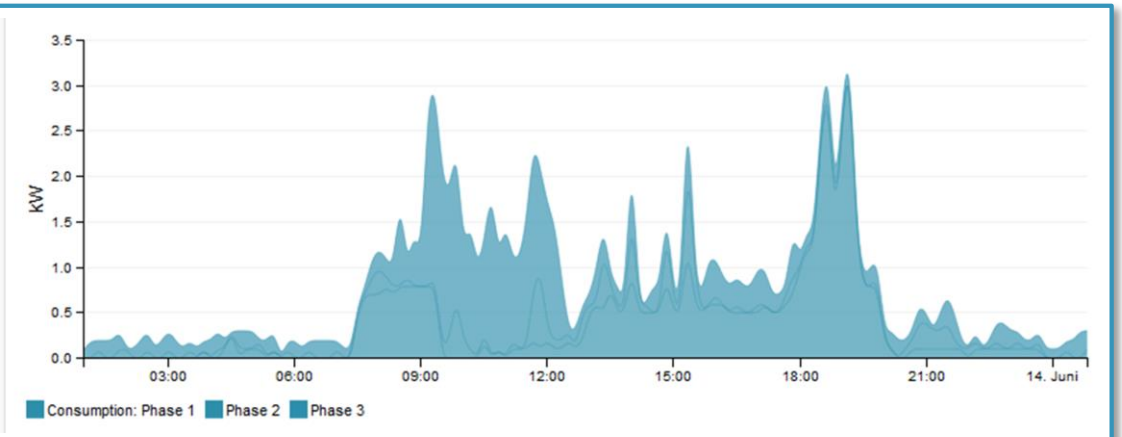


Figure II.2-1 Energy consumption curve

II.2.1 Loads management

A way to achieve more economic reasonable usage of electricity is to move electrical loads to periods that are covered by high-priority[4] energy production units with a low LCoE. Vice versa, loads are to be avoided in times with low-priority production units and a high LCoE. This approach is referred to as "Load Shifting" or "Peak-Load Shifting" (Eckl 2014).

In the situation of a photovoltaics system this implies in practice that certain electrical consumers are to be activated when cheap, currently produced

[1] As visualized in Figure II.1-2 "Typical PV production curve recorded by FEMS" on page 15
[2] See Chapter IV.3.3 "FENECON Online-Monitoring" on page 48
[3] See Figure V.3-1 "Effect of an SOC based load management" on page 58
[4] See Figure II.1-6 "Energy source prioritization" on page 19

electricity from PV is available ("Valley Filling") in order to avoid supplying them from grid at high cost times ("Load Shifting"). As a result, this reduces the amount of consumed electricity during high cost times ("Peak Clipping"). This is a major change from how electrical loads are used traditionally.

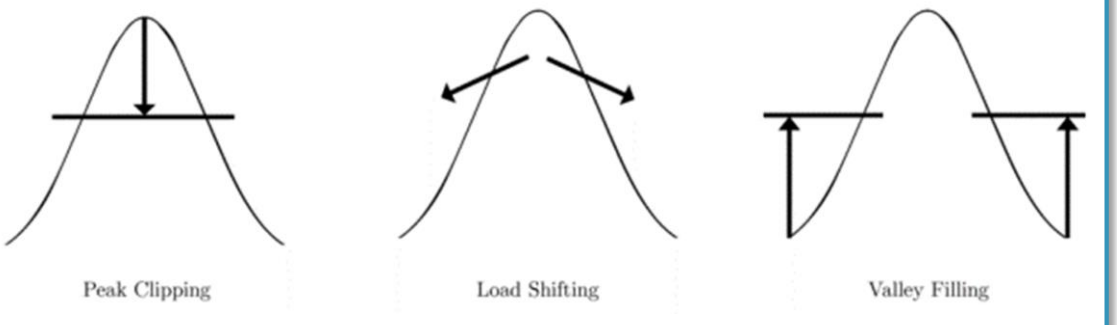


Figure II.2-2 "Peak-Load Shifting" approach (Eckl 2014)

II.2.2 Electrical loads and Added Value

Only certain loads in a household or company can be reasonably controlled by an energy management system to shift their electrical consumption to another period. First requirement is being a fairly big load to justify the effort. Secondly, the shift needs to be possible without loss of comfort for the user. This excludes consumer electronics like TV and radio and other on-demand loads like a stove, machines like a circular saw and so on from the list.

Nevertheless, there are a number of loads that fulfil these requirements and where an automatic control is certainly reasonable. This paper introduces the term "Added Value" to indicate the additional value of turning on a load at a specific point in time.

Many appropriate loads for energy management are related to central water heating, from simple heating-devices to heat pumps, or loads like air-conditioning systems and electric cars. Analyzing these kinds of loads shows, that the Added Value per kWh of invested electricity is dynamic. The example in Figure II.2-3 visualizes this effect with regard to the charge cycle of an electric car. When the car's battery is close to empty, each kWh charged adds a high value, so that even charging from grid is reasonable. The more the battery fills, the less additional value is achieved, expressed in less Added Value in €Ct per kWh.

The example is a simplification, because the threshold values, the Added Value and the availability of the electric car for charging depend on further parameters, but it shows the general conception.

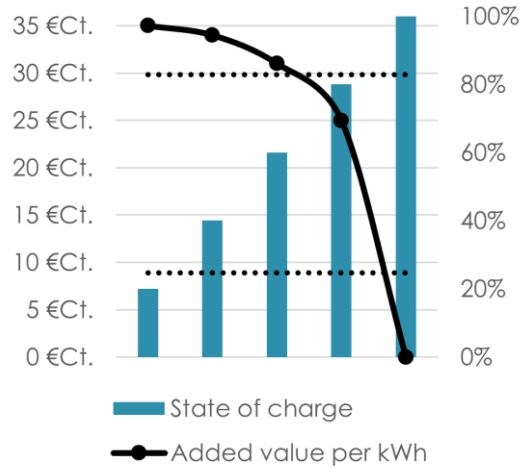


Figure II.2-3 Added value per kWh for electric car

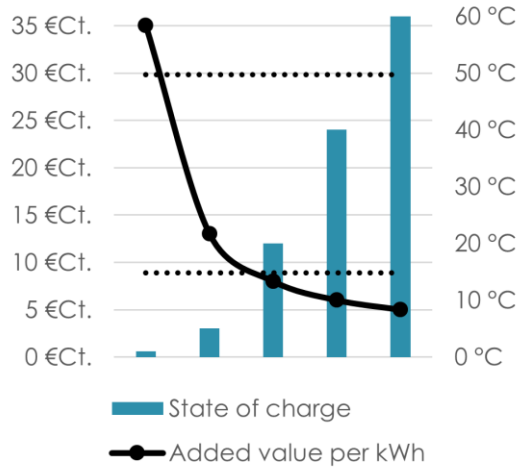


Figure II.2-4 Added value per kWh for water heating device

Similarly, Figure II.2-4 shows a graph for Added Value for a water heating device attached to a central warm water cistern. To avoid freezing, the Added Value around 0 °C is high, but generally it is not practicable to use electricity from grid, energy storage or photovoltaics for water heating. Still, in cases with very low cost per kWh, as discussed in Chapter II.2.3 "Feed-in tariff", even water heating can be reasonable.

Both figures show the electricity prices from grid[1] and the calculated LCoE of photovoltaics[2] in dotted lines. The proposed energy management system uses the energy source prioritization as discussed in Chapter II.1.4 "Energy source management" to decide if a load should be scheduled or not.

II.2.3 Feed-in tariff

In many countries in the world, owners of photovoltaics systems can sell electricity to the power grid. As one of the first countries, Germany introduced the Renewable Energy Act in the year 2000, which guarantees a fixed "feed-in tariff" per each kWh that is sold to the grid for the following 20 years. With up to 50,6 €Ct/kWh (net) (Bundesanzeiger 1990) in the beginning, this subsidy

[1] Electricity price from grid: 29,81 €Ct./kWh; see Chapter II.1.3 "Power grid" on page 18
[2] LCoE of photovoltaics: 8,87 €Ct./kWh; see Equation II.1-1 "Cost per produced kWh from photovoltaics" on page 16

program jump-started the usage of photovoltaics in Germany and had a big impact on the price-drop of PV modules and enabling grid-parity.

The feed-in tariff decreased over the years and is as low as about 12 €Ct (net) in 2015, depending on the size of the installation and the month of commissioning. For this reason, and because of additional bureaucracy and taxes, many installations with photovoltaics and energy storage systems nowadays are configured to not sell excess energy to the grid anymore.

This leads to a situation where, in case of a full battery and a current electricity production that is bigger than the energy consumption, electricity is wasted, because it cannot be used. In other words, if loads can be automatically activated to consume this excess electricity, the cost can be considered to be 0 €Ct per kWh.

While the examples in this paper are based on a system that is not selling to grid, the proposed concept is prepared to handle systems with this option activated. In this case, "sell-to-grid" can be considered a load with an Added Value according to the guaranteed feed-in tariff.

II.3 Proposed architecture

A system as shown in Figure II.3-1, including photovoltaics, additional energy sources, an energy storage system and manageable and non-manageable loads, leads to a very complex optimization problem for an energy management system.

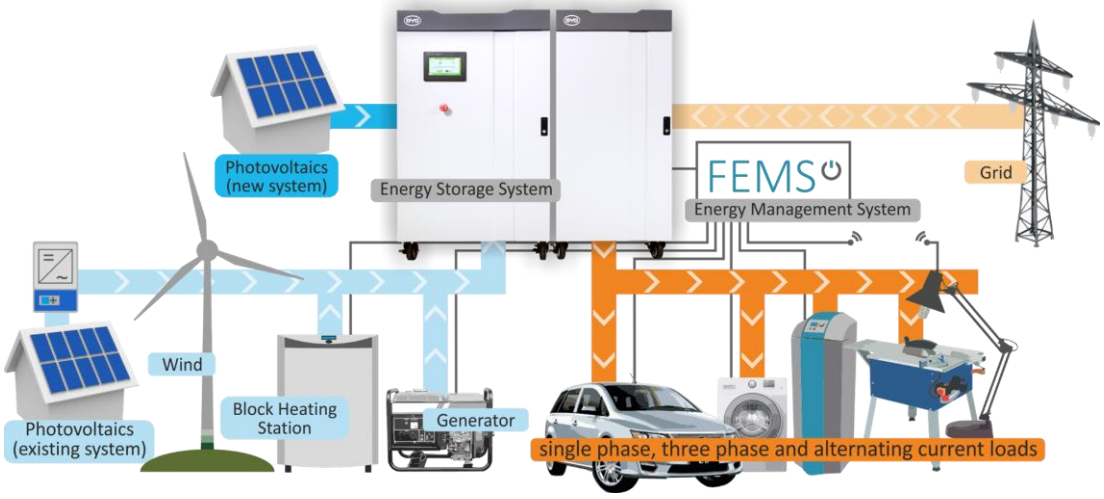


Figure II.3-1 PV system and additional energy sources, energy storage system, power grid and manageable loads, in combination with an energy management system

To tackle it, the problem needs to be split in smaller tasks that can be handled by simple, modular units. For the proposed architecture, as visualized in Figure II.3-2, therefore the concept of software agents as concurrent and autonomous units is suggested.

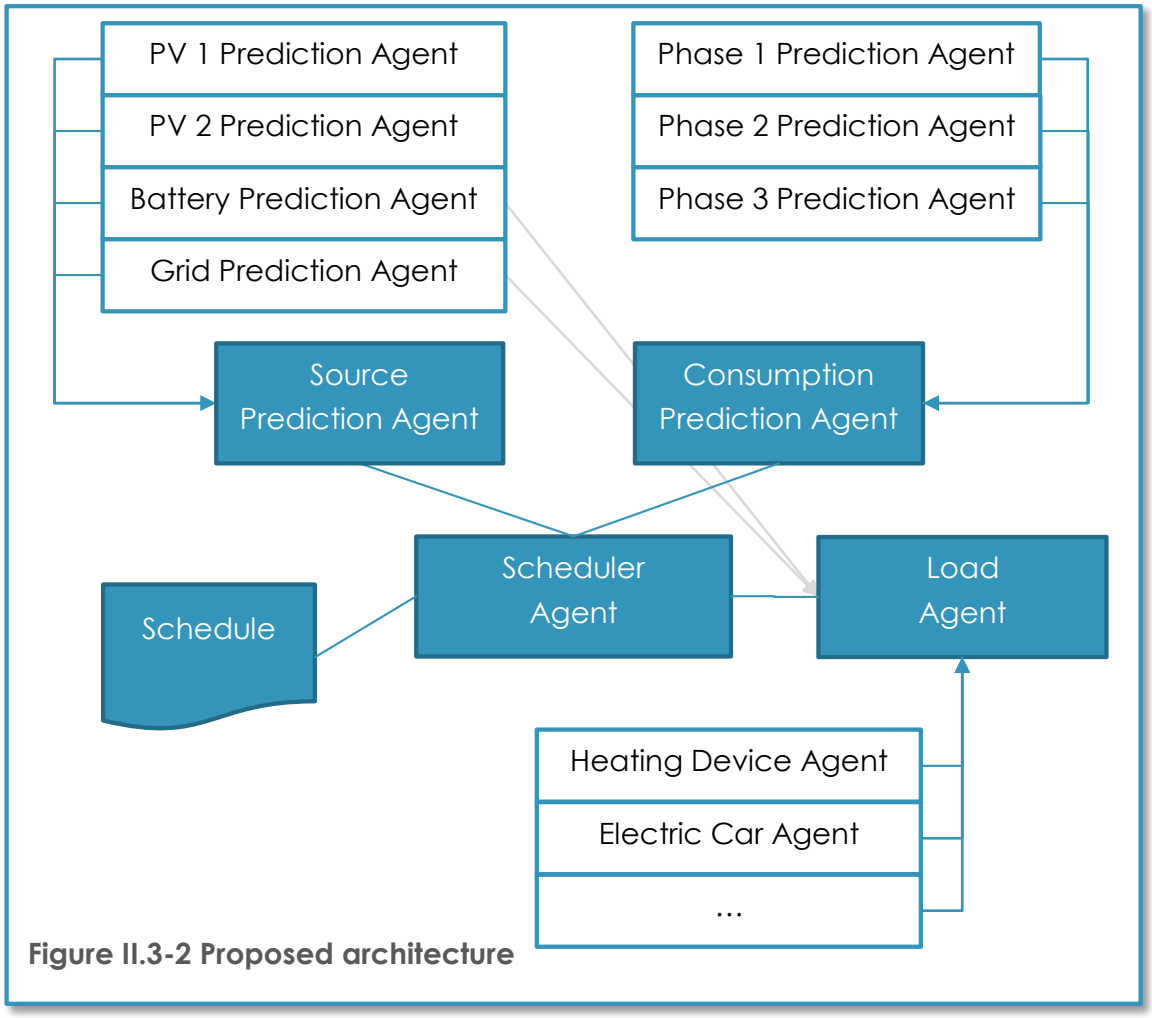


Figure II.3-2 Proposed architecture

An energy management system needs to have an overview on the current and future energy source situation. This information is required to generate a dynamic, time-based model of the availability of energy sources with regard to their prioritization and LCoE. The architecture maps each energy source to a "Source Prediction Agent" that uses its current and historic data to create an individual prediction for future available energy.

Additionally an overview on the current and future energy consumption situation is required. Therefore, individual "Consumption Prediction Agents" per electric phase generate a dynamic, time-based model of the future non-manageable energy consumption.

Individual "Load Agents" per specific manageable load provide information about the specific Added Value, properties and requirements in a third dynamic, time-based model.

Finally, all the prediction models are collected and combined by the "Scheduler Agent". It executes a scheduling algorithm that generates an optimal energy management "Schedule" of how and when manageable energy generators and loads should be controlled.

In a simple example scenario (Figure II.3-3), the "PV 1" and "PV 2 Prediction Agents" report that within the next hour 5 kW of electricity will be produced. "Battery Prediction Agent" states, that the battery of the energy storage system is full. At the same time, the total consumption reported by the "Phase 1", "Phase 2" and "Phase 3 Consumption Prediction Agents" sums up to 2 kW for the timespan. The "Scheduler Agent" collects this information and combines it with the requirements and Added Values of the registered "Load Agents". As the "Electric Car Agent" reports, that the car is nearly fully charged already, it figures, that it is feasible to activate the "Heating Device Agent" during this period to consume the excessive 3 kW. It is therefore adding the load to its "Schedule".

Time / **Agent**	**9:00**	**9:05**	**9:10**	**9:15**	**9:20**	**9:25**	**9:30**	**9:40**	**9:50**
PV1 Prediction Agent	2,6 kW								
PV2 Prediciton Agent	2,4 kW								
Battery Prediction Agent	8,5 kW (100 %)								
Grid Prediction Agent	∞ kW								
Source Prediction Agents (available power per priority)	**5,0 kW PV; 8,5 kW Battery; ∞ kW Grid**								
Phase 1 Prediction Agent	0,9 kW								
Phase 2 Prediction Agent	0,5 kW								
Phase 3 Prediction Agent	0,6 kW								
Consumption Prediction Agents	**2,0 kW**								
Heating Device Agent	Consumption: 3,0 kW; Added Value: 8 €Ct/kWh								
Electric Car Agent	Consumption: 2,5 kW; Added Value: 5 €Ct/kWh								
Scheduler Agent	**Schedule "Heating Device Agent"**								

Figure II.3-3 Scheduling example

III. Theory

The proposed concept builds on scientific fields that were discussed in many previous papers. This chapter is giving an overview on the state of the art in scientific research in these fields. It continues with the chosen approach for handling the prediction problems using machine learning systems and applies it on the Source and Consumption Prediction Agents. Finally, the theoretical implementation of the proposed architecture is being discussed.

III.1 State of the art

The dynamic time-based models introduced in Chapter II.3 "Proposed architecture" require the prediction of future energy production and consumption. This chapter analyses a selection of scientific papers on these topics and concludes with a suggestion for an approach to tackle the mentioned prediction problems.

III.1.1 Prediction of photovoltaic energy production

Conventional prediction of photovoltaics power generation is based on computer simulations and empirical modelling. An adequate mathematical model of a PV generator requires the precise understanding of inter-relationships and interactions, which is difficult to achieve and hardly reasonable for small, decentralized installations, considering the variety of external influencing parameters and the total complexity. Consequently, the trend is that most scientific literature using statistical or mathematical approaches on the topic is dedicated to large-scale photovoltaic power plants where the huge effort is justified, while artificial intelligence systems are used for small to medium sized installations.

A study on "Short-term prediction of photovoltaic energy generation by intelligent approach" (Chow, Lee and Li 2012) suggests the usage of artificial neural networks (ANN) models, because they are able to "mimic complex and nonlinear systems without any assumption towards input/output variable correlation". The paper applies a commonly used Multi-Layer Perceptron (MLP) network with one input, one hidden and one output layer to 10 and 20 minutes short-term predictions on approximately half a year of input data with reasonable success.

$$N_{hidden} = \frac{N_{input} + N_{output}}{2} + \sqrt{N_{samples}}$$

Equation III.1-1 Rule of thumb for the number of hidden neurons

The study uses a rule of thumb (Equation III.1-1), which results in an approximation of 36 neurons for that specific case. After running a sensitivity test by varying the number of neurons on the hidden layer between 31 and 41, it concludes, that "the rule of thumb […] is justified statistically". While correct for this study, this is not automatically true for other applications. Even the source of that rule of thumb states, that "there are no absolute formulae or even extremely successful rules of thumb" (Ward Systems Group Inc. 1996). Secondly, the rule of thumb has a parameter $N_{samples}$, which is often unknown at the time of definition of the MLP architecture.

Many papers about photovoltaics prediction are originated in Spain, because owners of big photovoltaics plants need to present the production figures for the following day to the electricity market operator. In the paper about a "Short-term power forecasting system for photovoltaic plants" (Fernandez-Jimenez, et al. 2011) the prediction is carried out at 9:00 a.m., consequently targeting to predict the production of the upcoming 39 hours. Several forecasting models are evaluated on the problem using the root mean square error (RMSE), namely the persistence model, time series model, k-nearest neighbors (k-NN), artificial neural networks (ANN) and adaptive neuro-fuzzy (ANFIS) networks. An MLP network with 7 neurons on the first and 7 neurons on the second hidden layer is found to provide the best results. The paper also suggests creating additional input variables to the networks, to represent the moment of the year and the moment of the day.

Numerous further papers prefer different forms of artificial intelligence methods, like Artificial Neural Network (ANN) ((Izgi, et al. 2011) and (Adel Mellit 2010)), Radial Basis Function Network (RBFN) (Chen, et al. 2011), State Vector Machine (SVM) (Bouzerdoum, Mellit and Pavan 2013) or Nonlinear Autoregressive Neural Network (NAR-NN) (Almonacid, et al. 2014), over statistical or mathematical approaches on similar prediction problems.

It is notable and reasonable that all cited papers use historic data of the photovoltaics generator as well as weather or solar irradiance data as input data to the methods. Unfortunately the recording of weather data for the example installation was not reliable, resulting in a non-sufficient quality. Consequently, the weather data could not be used for the algorithms in this thesis.

III.1.2 Prediction of electricity consumption

Most scientific papers on energy consumption are referring to the aggregated consumption in large areas or whole countries, while forecasts for small environments and microgrids are only recently becoming a topic in scientific research.

A thorough analysis on the topic is provided by the paper "Artificial neural networks for short-term load forecasting in microgrids environment" (Hernandez, et al. 2013). While the range of electricity consumptions in the paper is still between 7 to 39 MW, which is certainly much higher than the expected values for decentralized installations as discussed in this thesis, the paper shows that an MLP network is also feasible for consumption prediction with reasonable results. The advanced methods of classification and clustering before applying the MLP model as well as the inclusion of temperature and holiday settings as additional inputs are worth mentioning and suggestive for ways to further improve simple ANNs.

The study on "New artificial neural network prediction method for electrical consumption forecasting based on building end-uses" (Escrivá-Escrivá, et al. 2010) uses a bottom-up approach in identifying so called "end-uses" (EU) as "independent energy processes that can be identified and […] measured in consumption". This approach is very well applicable to the different electrical phases of a decentralized system as identified in Chapter II.3 "Proposed architecture" as "Phase 1/2/3 Consumption Prediction Agent". The paper suggests classifying the inputs with regard to the "type of day" and to remove obvious "unpredictable factors" before applying the MLP and concludes that ANNs are an appropriate tool to predict energy consumption per EU.

Another applicable paper using SVM and MLP network is the study on "Short term electricity forecasting using individual smart meter data" (Gajowniczeka and Zątkowski 2014). The authors approve the feasibility of their approach with the conclusion that "the results […] are promising" but admit that the "forecasting on individual household level is a difficult task since the daily household behavior may change drastically due to different circumstances".

III.1.3 Conclusion

The state of the art shows, that Artificial Neural Networks are a well-established and feasible approach for electricity production and consumption prediction problems.

As with every machine learning system, the prediction accuracy depends on the selection of significant input data. Apart from historic production and

consumption data and weather data, some papers therefore suggest adding additional input parameters like the moment of the year and the moment of the day, temperature and holiday settings and so on. In addition, filtering and pre-classification of the input data is recommended to improve the overall prediction accuracy.

Apart from that, finding the best architecture for an MLP network is critical to the performance of the method. Unfortunately, there is not yet a widely accepted, reliable way to determine the optimal architecture, like the number of neurons on the hidden layer. Rules of thumb can help to calculate a first approximation, but more commonly, a trial and error approach is used.

III.2 Prediction using Machine Learning

The development of a system to mimic human behavior is studied in the scientific field of Artificial Intelligence (AI). The standard literature identifies four subfields of AI: (Russell and Norvig 2010)

- "Natural language processing" to enable communication in a human language.
- "Knowledge representation" to store information provided before or during the interrogation.
- "Automated reasoning" to use the stored information to answer questions and to draw new conclusions.
- "Machine learning" to adapt to new circumstances and to detect and extrapolate patterns.

A well-known quote by Arthur Samuel from 1959 describes Machine Learning (ML) as the "field of study that gives computers the ability to learn without being explicitly programmed".

This chapter provides an introduction to the ML method of Artificial Neural Networks (ANN), the architecture and application of ANNs to time-series prediction problems, a way of implementing general, reusable predictors for the prediction problems in the proposed architecture, the usage of the Encog Java framework and the methodic search for an optimized ANN architecture using FADSE.

III.2.1 Artificial Neural Networks

As shown in Chapter III.1 "State of the art", an ANN is a well-established and widely used Machine Learning method. Its architecture is inspired by the way

the human brain is using a set of simple, interconnected units, called neurons, to process and learn information.

III.2.1.1 Perceptron and forward-propagation

The mathematical model of a neuron is called a "perceptron" as visualized in the schema in Figure III.2-1.

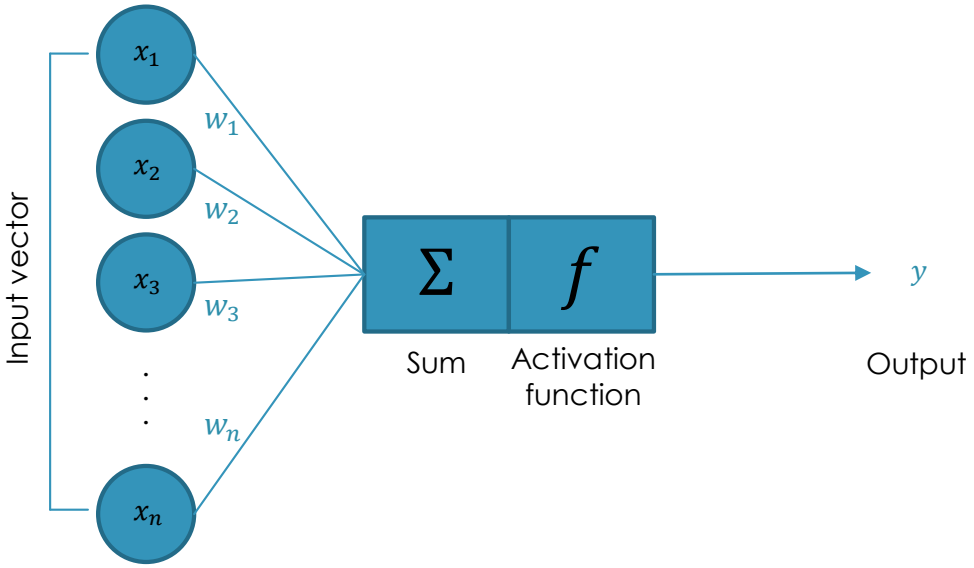


Figure III.2-1 Schema of a perceptron

The output value of a perceptron is calculated using "forward-propagation". Each value x_i of an input vector is amplified or deamplified by multiplying it with an individual weight w_i. The sum of all weighted signals is then passed to the activation function f, which returns the output value y.

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Equation III.2-1 Perceptron forward-propagation

III.2.1.2 Learning through back-propagation

The interesting characteristic of a perceptron is that it is able to learn from data by adjusting its weights. A typical approach is the back-propagation algorithm (Rumelhart, Hinton und Williams 1986) that is calculating the error E as the difference between the calculated output y of the perceptron and the ideal target output t.

$$E = t - y$$

Equation III.2-2 Calculation of the error during back-propagation

The error E is then used to adjust the perceptron's weights by a defined learning rate r.

$$\acute{w_i} = r \times E \times w_i$$

Equation III.2-3 Adjustment of the perceptron's weights

With each iteration of this algorithm, the perceptrons output comes closer to the ideal targeted output: the perceptron learns from the data.

III.2.1.3 Multi-Layer Perceptron

A Multi-Layer Perceptron (MLP) network uses the basic perceptron model to construct an architecture of input, hidden and output layers (Figure III.2-2). In this architecture, the calculated output y of each perceptron is used as an input for all perceptrons on the next layer. By applying the forward- and back-propagation algorithms on a big set of example data pairs with input and ideal output vectors, the MLP network is able to learn advanced, non-linear functions.

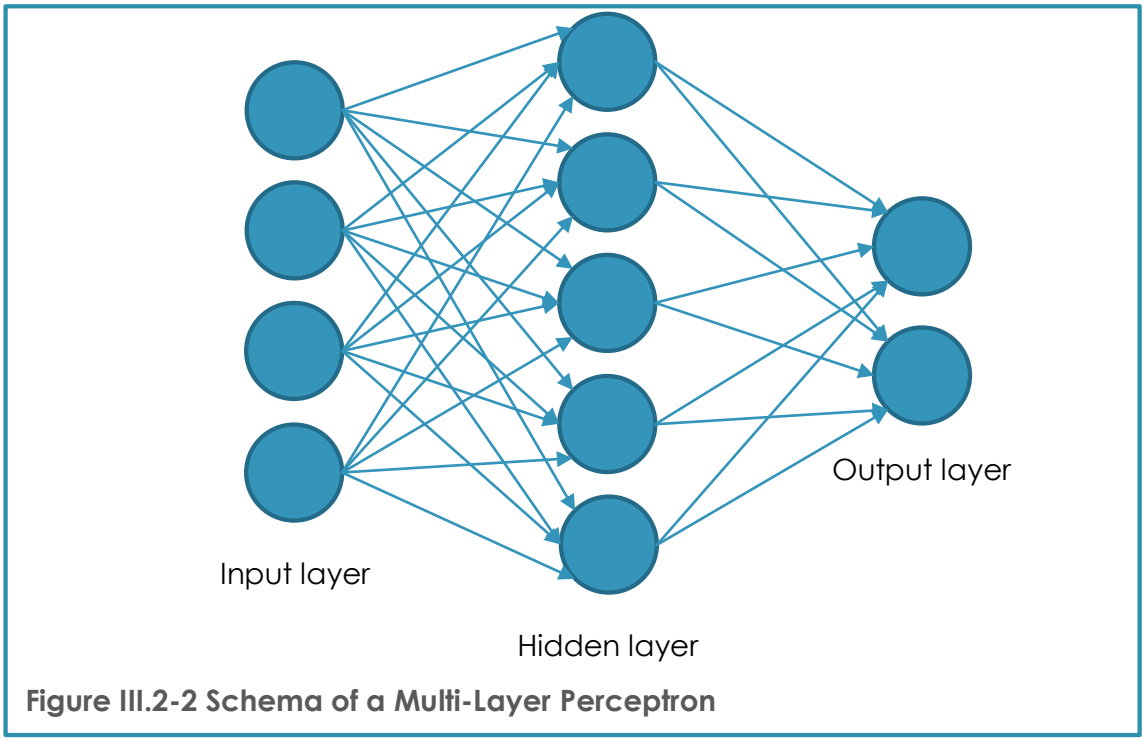


Figure III.2-2 Schema of a Multi-Layer Perceptron

While Multi-Layer Perceptron networks are an ongoing topic in scientific research, and more sophisticated and efficient algorithms, interconnections and implementations exist, they are all based on this fundamental architecture.

III.2.1.4 Training set, validation set and overfitting

In principle, each iteration of the back-propagation algorithm possibly reduces the output error, effectively improving the prediction accuracy on the input dataset more and more. However, when iterating too often, the MLP network can "overfit", resulting in a small error on the known input data set, but a poor prediction performance ("generalization") on unknown data.

To avoid overfitting and to improve generalization of the network, the initial dataset can be split in a training set and a validation set. Only the training set is used to learn the MLP using back-propagation. After every iteration the error for both training and validation set are calculated to track the performance on both datasets. Figure III.2-3 shows a typical outcome of such an analysis. The optimal performance of the network would be found at the global minimum of the Validation Error.

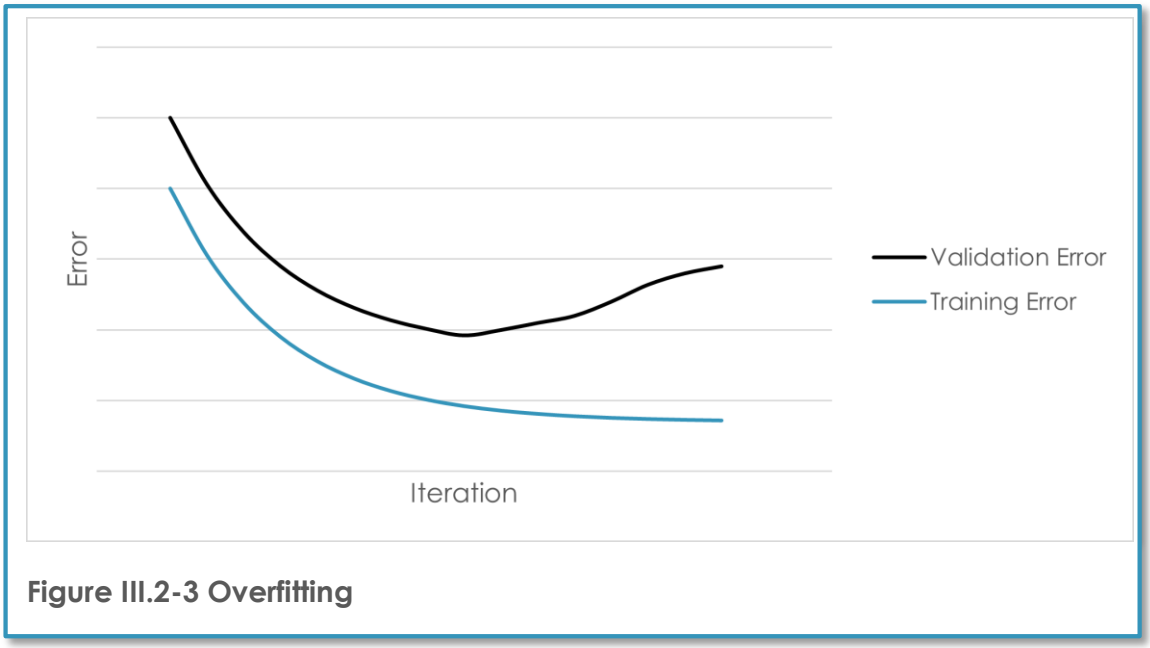


Figure III.2-3 Overfitting

III.2.2 Time-Series prediction

The data on electricity production and consumption is essentially a continuous sequence of values per time, a so-called "time-series". To use an MLP network for time-series prediction, the data needs to be fitted to the input and output vector format using the "Sliding Window" technique.

This technique uses one time "window" per vector, which is holding the required number of values.

The "Lag window" is providing the input vector for the MLP network with one value per input neuron. It is filled using the past values. The "Lead window" describes the predicted or ideal output vector of the network. Its size is determined by the number of output neurons and it is filled using the future values.

The first line of Figure III.2-4 exemplifies the creation of $lag\ window_t$, taking into account the past three values x, x_{t-1} and x_{t-2}, and of the appropriate prediction output $lead\ window_t$, consisting of the output values y_{1_t}, y_{2_t} and y_{3_t}. The figure also continues with the concept for creation of the upcoming lag and lead windows for $t+1$ and $t+2$. The resulting vectors are fitting for an MLP network with three input and three output neurons.

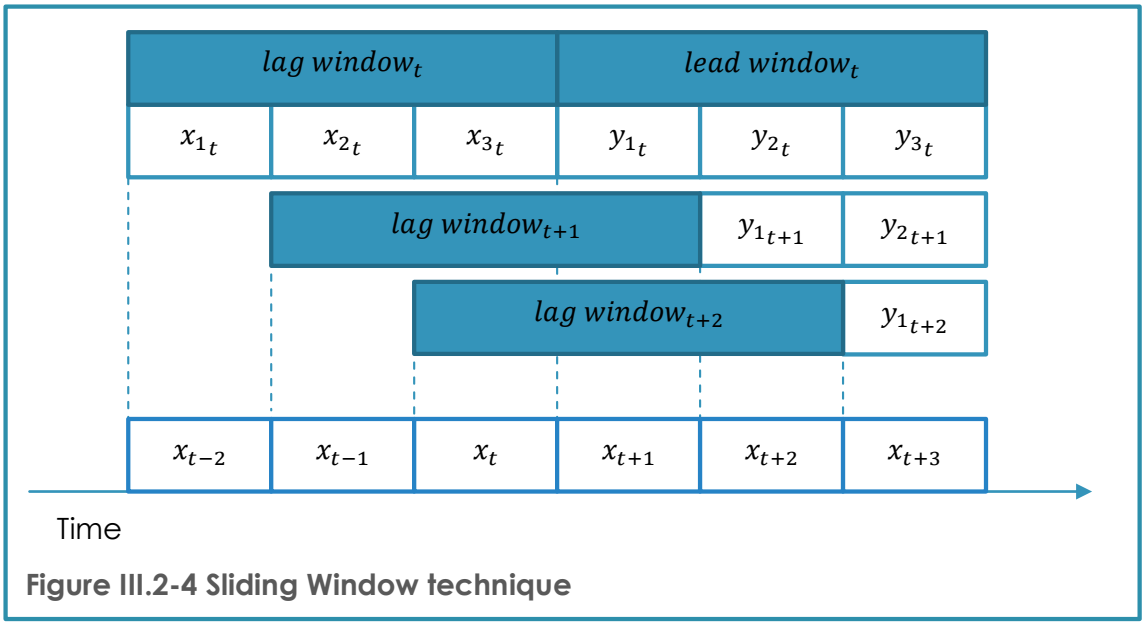


Figure III.2-4 Sliding Window technique

III.2.3 Implementation of generic predictors

Even if every energy storage system, every photovoltaic installation and every energy consumption at every customer has different, individual parameters and repeating patterns, the prediction problems have many similarities that can be combined to a generic implementation of a reusable "Prediction Agent" as basis of "Source" and "Consumption" and possibly "Load Prediction Agents".

An individual prediction problem can be defined as the prediction of one specific "Prediction Agent" and is carried out by a generic, reusable

"Predictor". Figure III.2-5 shows this structure with an individual "Predictor" per "Prediction Agent".

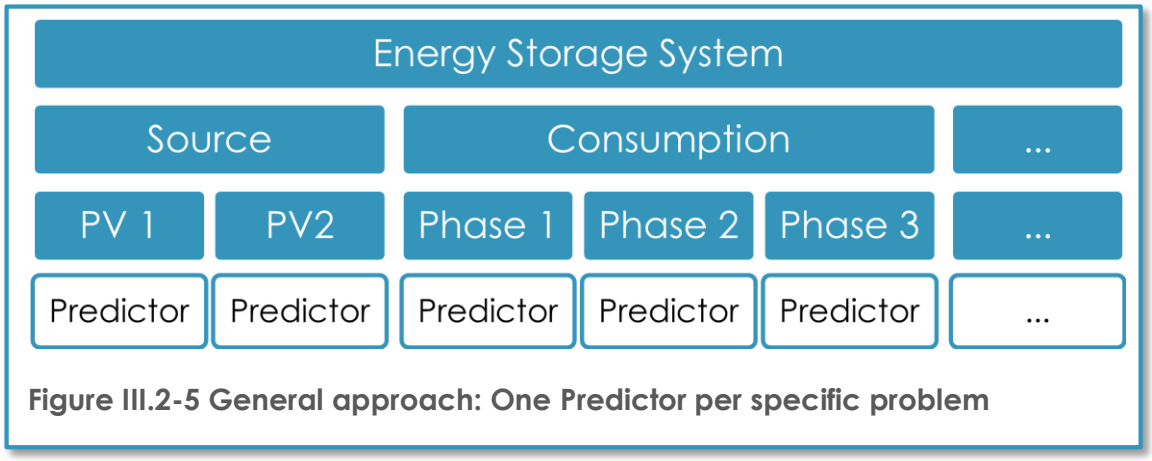


Figure III.2-5 General approach: One Predictor per specific problem

Each "Predictor" embeds a concrete machine learning method for its individual prediction problem as well as additional information about data pre and post processing like normalization.

III.2.4 Encog machine learning library

The actual operation of the machine learning algorithms is handled by the external open source library Encog. This java library is "a scalable, adaptable, multi-platform machine learning framework that […] allows a variety of machine learning models […] (which) can be used interchangeably with minimal recoding" (Heaton 2015). It provides implementations for a number of Artificial Neural Networks like Feedforward and Radial Basis Function (RBF) as well as Support Vector Machines (SVM) and others.

The Encog library is handling the creation of the applicable method, like a Multi-Layer Perceptron network, training of the network and calculation of training and validation errors.

The resulting Java object of the method can then be serialized to a file. This approach is used to learn and evaluate optimal networks separately and simply inject the optimized, individual ML method on demand in the proposed energy management system[1].

[1] See Figure III.2-7 "Toolchain for creation and injection of an Encog ML method" on page 37

III.2.5 Optimization using FADSE

As discussed in Chapter III.1 "State of the art" and the subsequent chapters, many variations of machine learning networks are possible. To find the best performing network for a problem, different methods, like MLP network, RBF, SVM and many more, need to be evaluated. For each of them a number of specific parameters, which again highly influence the accuracy of the algorithm, have to be defined before the actual training can be started.

Scientific papers[1] often apply a machine learning method based on personal preference and use a combination of rule of thumb and a brute force approach –trying all different possible configuration combinations – to find a proper parameter configuration.

To apply a more methodic search for an optimal network configuration within this huge design space, a powerful tool developed at "Lucian Blaga" University of Sibiu called "Framework for Automatic Design Space Exploration" (FADSE) (Calborean 2011) is used in this thesis.

III.2.5.1 Purpose of FADSE

FADSE was developed as a scientific approach for multi-objective optimization of advanced computer architectures. Based on a defined domain knowledge about an optimization problem, FADSE generates "Individuals", unique combinations of parameters, and explores their feasibility by means of system metrics. Using genetic algorithms new generations of individuals are created repeatedly, targeting to finding the Pareto frontier consisting of the globally optimal individuals.

FADSE also provides advanced features like the definition of relations using Fuzzy rules, client-server architecture to distribute the workload of simulations on a network and more. (Calborean 2011)

III.2.5.2 Predictor design space

The simplified example in Figure III.2-6 shows, that there is a high number of total possible parameter combinations for the suggested MLP network, even if only the "Lag window size" and the number of neurons on one hidden layer ("NoOfHiddenNeurons") are varied.

The sum of all these different practicable combinations is called the "design space".

[1] See (Chow, Lee and Li 2012)(Chow, Lee and Li 2012) for an example

Parameter name	Range	Description
LagWindowSize	$1 \dots 900$	Size of lag window; with one dataset record per 5 minutes: $900 \approx 3\ days$
NoOfHiddenNeurons	$1 \dots 270$	Number of neurons on the hidden layer; approximated using the rule of thumb (see Equation III.1-1 on page 27 for more details and the limitations of this approach)
No. of variations	$\mathbf{900 \times 270 = 243.000}$	

Figure III.2-6 Simplified predictor design space

In fact for finding the best performing machine learning method configuration, even more parameters, like the activation function, and even the machine learning method itself, with again a high number of specific parameters, would have to be varied. In conclusion, the actual feasible design space is even bigger than in this example and should be explored.

Using a tool like FADSE is justified on problems with big design spaces like this, as it avoids having to test each combination separately.

With regard to the purpose of this thesis – providing a basic development toolchain to support the optimization process – the before mentioned example is applied in this thesis as a first step, which could easily be extended in further studies.

III.2.6 Development toolchain

Evaluating an optimal machine learning method using the Encog library and FADSE is a task that involves heavy computations and a lot of processing time on a powerful computer. As the energy management system itself is supposed to run on an embedded device, it is necessary to provide the machine learning method separately.

Figure III.2-7 shows the concept of a toolchain based on FADSE that is creating the optimized method as an Encog object in Java and serializes it to a file. This file is then only injected it into the energy management system.

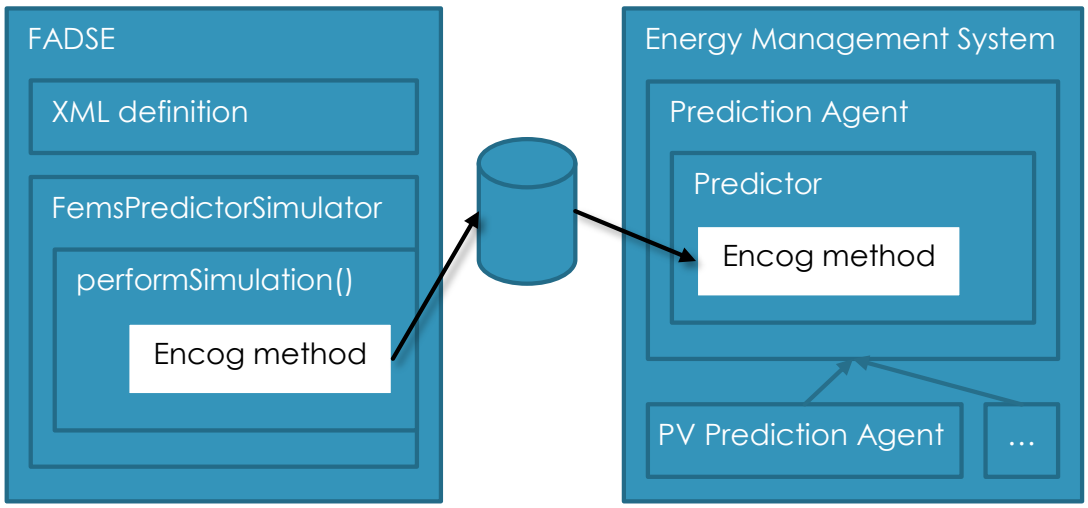


Figure III.2-7 Toolchain for creation and injection of an Encog ML method

IV. Implementation

After the specification of the "Concept" (Chapter II.) and the "Theory" (Chapter III.), this chapter discusses the implementation of the proposed development toolchain as well as the proposed energy management architecture. It continues with a presentation of the "FENECON Energy Management System" (FEMS) device as well as the "FENECON Online-Monitoring" which were designed and developed by the author prior to this paper. It finishes with an illustration on how the proposed system can enhance the existing functionality to control external devices.

IV.1 Implementation of the development toolchain

Figure III.2-7 on page 37 introduced the concept of a development toolchain for the separate creation and evaluation of an optimized machine learning method using FADSE. This chapter shows the actual definition of the optimization problem and the implementation of the simulator in Java[1].

IV.1.1 Definition of the optimization problem

FADSE allows the definition of a problem in an XML configuration file. This file facilitates the definition of benchmarks that need to be run per individual, the design space with parameter names and ranges and the target system metrics. Further settings like additional rules, relations between the parameters and more are also possible. Figure IV.1-1 shows the XML file[2] that is used to define the prediction problems.

"Simulator" is referring to the custom "FemsPredictorSimulator"[3] implementation with parameters for the example device "fems20" and a "lead window size" of three hours.

The specific prediction problems as used by the "PV 1/2" and "Phase 1/2/3 Prediction Agents" are defined as "benchmarks".

[1] See Annex B. "Source code: development toolchain" on page 72
[2] For the complete XML file, see Annex B.5 "fems.FemsPredictor.xml" on page 80
[3] See Chapter IV.1.2 "Implementation of FemsPredictorSimulator" on page 39

"NSGAII" is configured as the metaheuristic, providing a genetic algorithm for creation of new individuals according to the given, separate configuration file, which is coming with FADSE.

Using the "parameters" element, the actual design space is configured as discussed earlier[1].

To evaluate the resulting machine learning system, the "Training Error" and "Validation Error" are defined as system metrics.

Eventually the "output_path" sets the file system directory for the serialized Encog Java objects and other files created by the simulator.

```
<design_space>
    <simulator name="FemsPredictorSimulator" type="simulator">
        <parameter name="Device" value="fems20" />
        <parameter name="LeadWindowSize" value="36" /> <!-- 3 hours @ 5 minutes: 3*(60/5)
    </simulator>
    <database ip="localhost" port="3306" name="FemsPredictor"
        user="fadse" password="password" />
    <benchmarks> <!-- which field to predict -->
        <item name="PV1" />
        <item name="PV2" />
        <item name="Ph1" />
        <item name="Ph2" />
        <item name="Ph3" />
    </benchmarks>
    <metaheuristic name="NSGAII"
        config_path="D:\fems\fadse\trunk\src\main\java\ro\ulbsibiu\fadse\extended\problems
    <parameters>
        <parameter name="NoOfHiddenNeurons"
            description="Number of neurons on the hidden layer of the neural network"
            type="integer" min="1" max="270" /><!-- Applying rule of thumb: Nh = (Ni+No)/
        <parameter name="LagWindowSize" description="Size of lag window for time series"
            type="integer" min="1" max="900" /><!-- up to about 3 days -->
    </parameters>
    <virtual_parameters></virtual_parameters>
    <system_metrics>
        <system_metric name="TrainingError" type="float" unit="" desired="small" />
        <system_metric name="ValidationError" type="float" unit="" desired="small" />
    </system_metrics>
    <rules></rules>
    <relations>
    </relations>
    <output output_path="D:\fems\fadse\output" />
</design_space>
```

Figure IV.1-1 FADSE XML definition

IV.1.2 Implementation of FemsPredictorSimulator

The Java object that is carrying out the actual simulation of an optimization problem is implementing the abstract "SimulatorBase" class. FADSE is

[1] See Figure III.2-6 "Simplified predictor design space" on page 36

generating new individuals according to the optimization problem description in the XML file and calls the "performSimulation" method of the simulator[1]. This method can be used to execute a simulation in Java code – like in this thesis – or by calling an external parameterized Matlab-Script or other specific simulators. The simulator finally returns one or more system metrics that are used by FADSE to evaluate the individual and build the Pareto frontier.

The simulator for this thesis is completely implemented in a Java class named "FemsPredictorSimulator". Per each individual configuration, the "performSimulation" method is called. This method starts with preparing a specific training data set with 70 % and a validation data set with the remaining 30 % of the total data. It also applies the "Sliding Window"[2] technique according to the given lag and lead window size of the simulation.

Then it instantiates an Encog "BasicNetwork" object that represents a Multi-Layer perceptron network with three layers: an input layer with a size according to the lag window size, a hidden layer with the individual number of hidden neurons and an output layer, defined by the lead window size.

To train the network using back-propagation[3] Encog provides a number of advanced techniques like "resilient back-propagation" (Rprop) or the "Levenberg Marquardt algorithm" (LMA). The implementation of "ResilientPropagation" was chosen for this thesis, as the LMA performance was too slow for big MLP networks on the testing machine. The "ResilientPropagation" eases the usage of the back-propagation algorithm, as it is able to find and adjust internal parameters like the learning rate automatically. (Riedmiller and Braun 1992)

Encog also provides a way to define a strategy on how and when to stop the training process. The implementation of the "SimpleEarlyStoppingStrategy" stops the training, when the error on the validation set no longer improves. This is an effective way to find the global minimum of the validation error and to avoid overfitting[4].

After running through the back-propagation iterations and finishing the training, the trained machine learning method is serialized to a file in the output path that can be injected into the energy management system. In addition, the final training and validation errors are calculated and returned as system metrics back to FADSE.

---

[1] See Annex B.1 "FemsPredictorSimulator.java" on page 72
[2] See Figure III.2-4 "Sliding Window technique" on page 33
[3] See Chapter III.2.1.2 "Learning through back-propagation" on page 30
[4] See Chapter III.2.1.4 "Training set, validation set and overfitting" on page 32

IV.2 Implementation of the energy management architecture

To test the general functionality of the proposed architecture[1] including the injection through the development toolchain and to simulate a real energy management system, a basic implementation is required. This chapter describes the building blocks of this implementation[2].

IV.2.1 Basic types: Value, Prediction, Timestamp and Field

Some data types need to be defined for basic, repeating information throughout the application.

Each concrete numeric Value is stored in as data type double.

A Prediction object holds one specific predicted Value together with its corresponding lag window size, expressing the accuracy of the prediction.

Each Value in a time-series, concrete Prediction object or any other timed information is referred to by a Timestamp of data type long, measuring the seconds since 1st January 1970.

A Field is identifying and referencing a specific field in the communication protocol of a real energy storage system, like "Photovoltaic 1 Output Power" or "Phase 2 Load Power".

IV.2.2 EssListener and ProHybridSimulator

The Energy Storage System Listener ("EssListener"[3]) interface is the junction point to receive data about the energy storage system from the FEMS software.

For the demonstrative implementation, the real storage system is replaced by a "ProHybridSimulator"[4] which is simulating a FENECON by BYD PRO Hybrid system. On the basis of previously recorded data, the simulator supplies the EssListener with real-time data in a time-lapse mode of 1 to 3.000, sending a new 5-minutes average value every 100 milliseconds. This approach enables effective testing of the agent interactions and event handling.

---

[1] See Figure II.3-2 "Proposed architecture" on page 24
[2] See Annex C. "Source code: implementation of the proposed architecture" on page 88
[3] See Annex C.6.1 "EssListener.java" on page C.6.1
[4] See Annex C.6.2 "ess.prohybrid.ProHybridSimulator.java" on page 121

IV.2.3 Prediction Agent and Predictor

The Encog method generated by the supporting toolchain is injected into a generic Predictor object. The abstract "Prediction Agent"[1] references one specific Predictor[2], to carry out the actual prediction process. This agent also implements the "EssListener" to receive notifications from the storage system.

On a notification via the EssListener, the Prediction Agent calls its referenced Predictor with the new data. The Predictor uses the injected Encog method to calculate the next predictions. The result is returned back to the Prediction Agent and published on its internal predictions table.

IV.2.4 Source and Consumption Prediction Agents

Specific Source and Consumption Prediction Agents implement the abstract "Prediction Agent" for a certain Field.[3]

"PV 1" and "PV 2 Prediction Agent"[4] handle the photovoltaic electricity production on both photovoltaic installation strings connected to the energy storage system.

Three "Grid Agents"[5] simulate the grid power per phase, which is in principle assumed to be infinite.

Three "Consumption Prediction Agents"[6] hold information on the consumption per single phase.

IV.2.5 Load Agent

An abstract "Load Agent"[7] represents one physically manageable load, like a heating element, an electric car, washing machine and others.

Each load has a specific load profile and properties that vary in time. Therefore, the implementation of the "Load Agent" needs to provide information on the required power and the specific Added Value per timestamp[8]. The Load Agent

---

[1] See Annex C.7.5 "PredictionAgent.java" on page 132 and Annex C.7.7 "PredictionAgentImpl.java" on page 135
[2] See Annex C.7.8 "Predictor.java" on page 138
[3] See Figure III.2-2 "Schema of a Multi-Layer Perceptron network" on page 31
[4] See Annex C.5.6 "pv.PvAgent.java" on page 117
[5] See Annex C.5.3 "grid.GridAgent.java" on page 113
[6] See Annex C.2.1 "ConsumptionAgent.java" on page 94
[7] See Annex C.3.3 "LoadAgent.java" on page 100
[8] See Figure II.2-3 "Added value per kWh for electric car" and Figure II.2-4 "Added value per kWh for water heating device" on page 22

is designed for being easily extendible with further sensible parameters, like a minimum switch-on time, as demanded by future requirements.

The architecture is prepared to allow "Load Agents" to solve prediction problems like "When is the electric car plugged in and available for charging?" or "Which minimum water temperature is required per time of the day?" by implementing a "Predictor" that can be prepared and injected by the development toolchain.

The sample implementation provides a "HeatingDeviceLoadAgent"[1], which simulates a simple heating device like the one in the "Scheduling example"[2], with a static Added Value below the priority of the photovoltaics source category[3] and a constant required power of 500 W.

IV.2.6 Scheduler Agent

The "Scheduler Agent"[4] regularly polls all registered Source and Consumption Prediction Agents to create a dynamic time-based model of the electricity generation and consumption. By applying each Load Agent's requirements, it then identifies scheduling opportunities and finally generates an energy management "Schedule".

In the current implementation, this "Schedule" is only displayed on the console output, but the data is structured in a way to enable the visualization using a Gantt chart as in the mockup in Figure IV.2-1.

Time / Load	9:00	9:05	9:10	9:15	9:20	9:25	9:30	9:40
Electric Car	✗	✓	✓	✓	✓	✓	✓	✗
Water heater	✗	✗	✗	✗	✗	✓	✗	✗

Figure IV.2-1 Mockup: Visualized schedule using Gantt chart

[1] See Annex C.3.1 "HeatingDeviceLoadAgent" on page 98
[2] See Figure II.3-3 "Scheduling example" on page 25
[3] See Figure II.1-6 "Energy source prioritization" on page 19
[4] See Annex C.4.1 "SchedulerAgent.java" on page 102

IV.3 FEMS and FENECON Online-Monitoring

Prior to this thesis "FEMS", FENECON Energy Management System, and "FENECON Online-Monitoring" were developed.

This chapter provides a brief introduction to the hardware and software used for FEMS and how basic control rules are programmed. It also gives a preview on how the architecture proposed in this paper can be attached to the existing system. Finally, the FENECON Online-Monitoring web portal is presented.

Figure IV.3-1 FEMS device

IV.3.1 FEMS Hardware

The FEMS device comprises a powerful system-on-a-chip with a 1 GHz ARM CPU, 512 MB RAM and 4 GB flash storage with a reasonable consumption of only 6 W.

Using an external USB-to-RS485 adapter, it is able to establish a serial connection to the energy storage system. The onboard Ethernet socket is used for the connection to the internet.

The optionally available, modular "FEMS Fieldbus" and "FEMS Wireless" extensions enable the actual management of external loads using cable or wireless connection.

IV.3.2 FEMS Software

An energy storage system is a long-term investment and expected to be in service for 20 years and more. As it is an integral part of an electrical installation, it is required to be well integrated in current and future home and industry automation systems.

Already today, that market is scattered with a variety of different systems and protocols and it can be expected to become worse in future. Furthermore, energy management is not the core business model of an "Energy Engineering"[1] company like FENECON.

[1] "Energy Engineering" is the slogan of FENECON, see the "FENECON Logo", Figure I.3-1 on page 11

Because of this, it was decided to build the FEMS software upon an open source software stack and provide the FENECON implementation itself also as open source.

The platform of choice is the Eclipse SmartHome project with its reference implementation openHAB.

IV.3.2.1 Eclipse SmartHome/openHAB

The openHAB[1] project was founded in 2010 by Kai Kreuzer for his own home automation requirements.

His philosophy of an "Intranet of Things", where a local "Home Automation Bus" (HAB) controls all appliances, and the clearly structured and modular Java OSGi architecture quickly attracted many developers and made openHAB one of the best-known open source home automation platforms.

Figure IV.3-2 Eclipse SmartHome and openHAB logos

With the development of openHAB version 2 the core functionality was converted to Eclipse SmartHome[2], which is developed under the umbrella of the non-profit Eclipse foundation under the permissive Eclipse Public License[3].

Eclipse SmartHome provides an extensive architecture for handling different kinds of hardware connectors, automation logic, user interfaces, logging, and event handling and so on. Everything is developed as OSGi bundles, simplifying the development and testing process of new features and handling of dependencies (Figure IV.3-3).

The embedded XText engine provides a flexible scripting language to define core elements of a smart home setup like Things, Items, user interface Sitemaps, executable Scripts and event based control Rules without touching the Java source code.

[1] openHAB – Open Home Automation Bus: http://www.openhab.org/
[2] Eclipse SmartHome: https://www.eclipse.org/smarthome/
[3] See Chapter IV.4 "Source Code" on page 48 and the license text in Annex D. on page 141

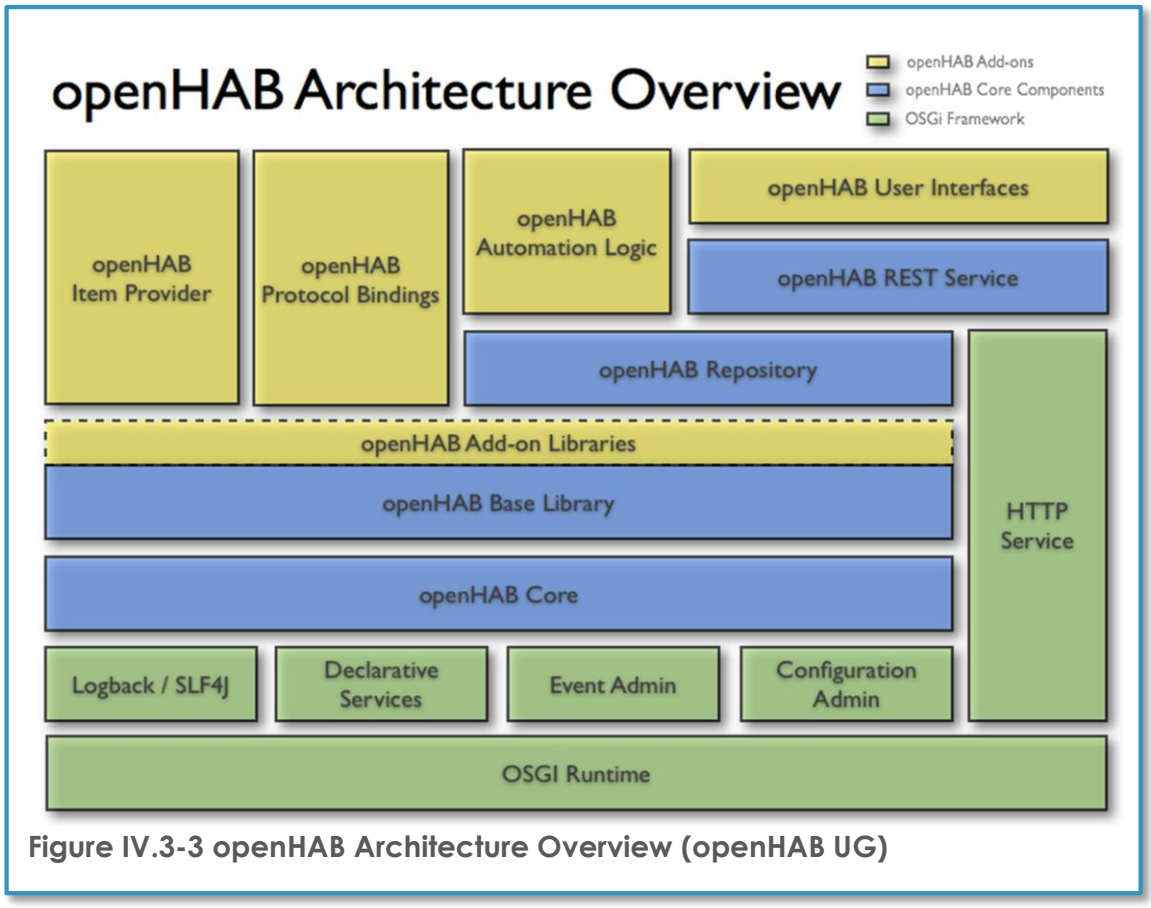


Figure IV.3-3 openHAB Architecture Overview (openHAB UG)

IV.3.2.2 FEMS binding for openHAB

Every additional hardware interface for openHAB or Eclipse SmartHome is developed as an OSGi bundle called a "binding".

The custom FEMS binding implements the protocols of several FENECON by BYD energy storage systems. It provides the data to the internal openHAB event bus as well as to the FENECON Online-Monitoring server.

It also takes care of a number of FEMS specific initialization routines and watchdog functionality to ensure stability in an unattended operating mode.

IV.3.2.3 Controlling external devices

In the existing installations of FEMS, the XText scripting language is used to create rules based on current data from the energy storage system. The example in Figure IV.3-4 shows a simplified rule to control a water heater via a relay output. The water heater is turned on when the state of charge (SOC) of the battery is above 97 % and turned off when it falls below 95 %. This is a typical rule to effectively use excess electricity in an energy storage system that is configured to not sell to the grid.

In case of a photovoltaics electricity production higher than the current consumption, the battery will be charged. If the production were still higher than the consumption when the battery is full, electricity would be wasted[1]. Enabling a manageable load when the battery is full effectively uses this excess electricity.

```
rule "Water heater"
when
    Item BSMU_Battery_Stack_Overall_SOC changed
then
    if(fems_dess_f_BSMU_Battery_Stack_Overall_SOC.state > 97) {
        logInfo("Water heater", "SOC > 97 % => start Water heater")
        RelayOutput_1.sendCommand(ON)
    } else if(fems_dess_f_BSMU_Battery_Stack_Overall_SOC.state < 95) {
        logInfo("Water heater", "SOC < 95 % => stop Water heater")
        RelayOutput_1.sendCommand(OFF)
    }
end
```

Figure IV.3-4 openHAB rule "Water heater"

IV.3.2.4 Attaching the proposed architecture

While the approach based on the current SOC is sufficient for this specific and common use case, it misses a lot of flexibility to solve problems that are more complex.

The architecture proposed in this thesis overcomes many of those limitations by its dynamic prioritization of loads according to their Added Value, optimized loads management even when the battery is not full, predictive integration of additional generators and more.

While the actual connection of the FEMS binding with the proposed architecture implementation is out of scope for this thesis, the general feasibility is proven by the simulation[2]. In the real application, the "ProHybridSimulator" will be replaced by the FEMS binding itself, which is notifying the EssListeners on arrival of new data. The resulting Schedule can be applied via the openHAB event bus.

[1] See Chapter II.2.3 "Feed-in tariff" on page 22
[2] See Chapter V. "Result" on page 50

IV.3.3 FENECON Online-Monitoring

FEMS is transferring the collected data to a dedicated webserver via a TLS encrypted connection, where it is stored in the dedicated InfluxDB[1] time series database.

The server then processes and visualizes the relevant information like photovoltaic production, energy consumption and state of charge of the battery and others in an easily accessible and user-friendly interface, the "FENECON Online-Monitoring", within the personal customer portal.

Figure IV.3-5 FENECON Online-Monitoring

Certain images in this thesis were taken directly from that portal, like Figure II.1-2 "Typical PV production curve" (page 15), Figure II.2-1 "Energy consumption curve" (page 20) and Figure V.3-1 "Effect of an SOC based load management" (page 58).

The "Example data recorded by FEMS" (Figure I.4-3 on page 13) is extracted directly from the time series database on the webserver.

IV.4 Source Code

The development toolchain is implemented directly within FADSE and the source code is available in Annex B. "Source code: development toolchain" from page 72 and in the GitHub repository on https://github.com/horiacalborean/fadse/.

The complete source code of the implemented architecture is available in Annex C. "Source code: implementation of the proposed architecture" from page 88 and in the GitHub repository on https://github.com/sfeilmeier/de.fenecon.fems[2]. It is licensed under the Eclipse

[1] InfluxDB time series database – https://influxdb.com/
[2] The state of the code discussed in this thesis is identical with the commit hash "3d85cb0e01becd0fb1ff4799e0ae33184dc5c9fb": https://github.com/sfeilmeier/de.fenecon.fems/commit/3d85cb0e01becd0fb1ff4799e0ae33184dc5c9fb

Public License 1.0. The license text is attached as Annex D. “Eclipse Public License - v 1.0” from page 141.

The “Encog Machine Learning Framework” is licensed under the Apache License 2.0 and is available at http://www.heatonresearch.com/encog.

More information on Eclipse SmartHome and openHAB and the respective source codes and licenses are available from https://www.eclipse.org/smarthome/ and http://www.openhab.org/.

V. Result

This chapter presents the results of the development toolchain and the simulation of the proposed energy management architecture. It also gives a preview of the expected results when managing real loads and suggests "next steps" for further studies on the subject.

V.1 Development toolchain for Predictors

The results presented in the following chapters show the outcome of the development toolchain using the discussed configuration[1]. The simulation was run on a dedicated computer for about three days.

V.1.1 Creation of individuals

Prior to every simulation step, FADSE creates a new individual according to the problem configuration[2] as visualized in Figure V.1-1.

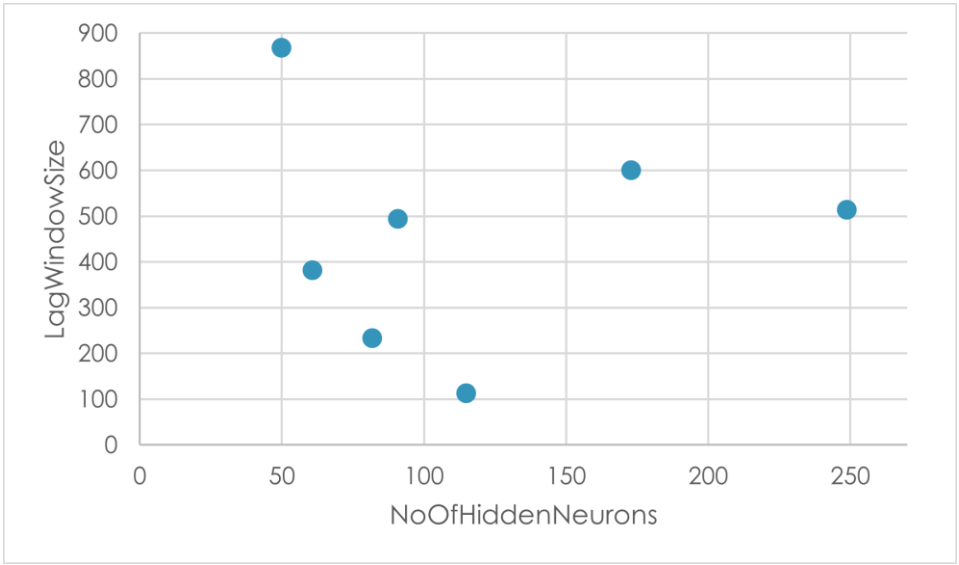


Figure V.1-1 Individuals generated by FADSE

Each individual is then applied to each benchmark for the fields "PV 1", "PV 2", "Phase 1", "Phase 2" and "Phase 3".

[1] See Chapter IV.1 "Implementation of the development toolchain" on page 38
[2] See Figure IV.1-1 "FADSE XML definition" on page 39

V.1.2 Training of Multi-Layer perceptron networks

While training separate Multi-Layer perceptron networks per benchmark and individual configuration, the training and validation errors are constantly recorded and checked for the stop condition[1] to avoid overfitting.

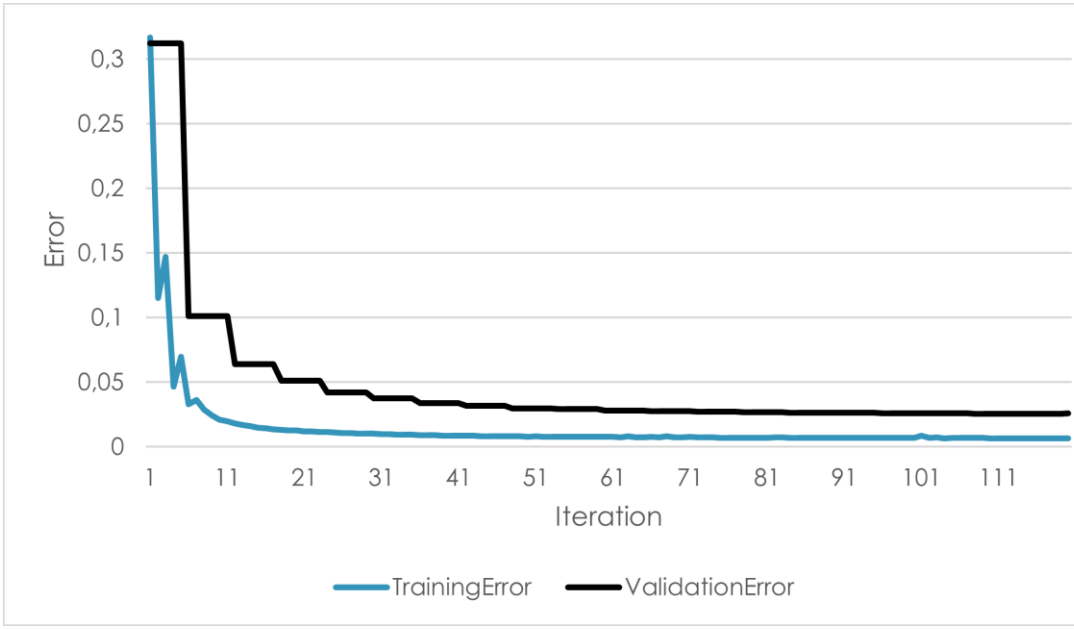


Figure V.1-2 Training and validation error per training iteration

Figure V.1-2 shows the training of a predictor for the "PV 1" benchmark with 249 hidden neurons and a lag window size of 513. The training stopped when the validation error reached its global minimum after 119 iterations.

V.1.3 Evaluation of individuals

After the training is successfully completed, the resulting system metrics are returned back to FADSE and used to evaluate each individual.

Figure V.1-3 visualizes the metrics per each individual on the "PV 1" benchmark, with regard to its number of hidden neurons ("H") and lag window size ("L"). The optimal configuration found by FADSE is the one with the lowest training and validation error. In the example, this is the individual with 249 hidden neurons and a lag window size of 513.

[1] See Chapter IV.1.2 "Implementation of FemsPredictorSimulator" on page 39

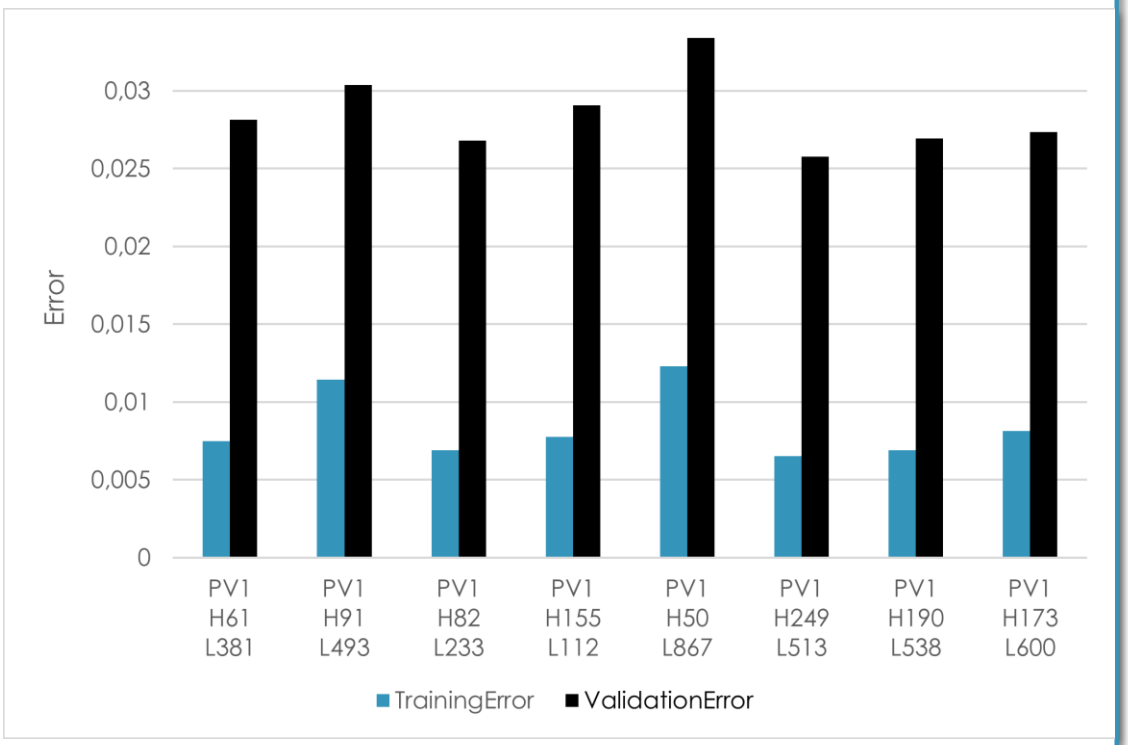


Figure V.1-3 Evaluation of individuals according to the training and validation error

In the same way, all the optimal configurations for MLP networks per benchmark as listed in Figure V.1-4 could be found.

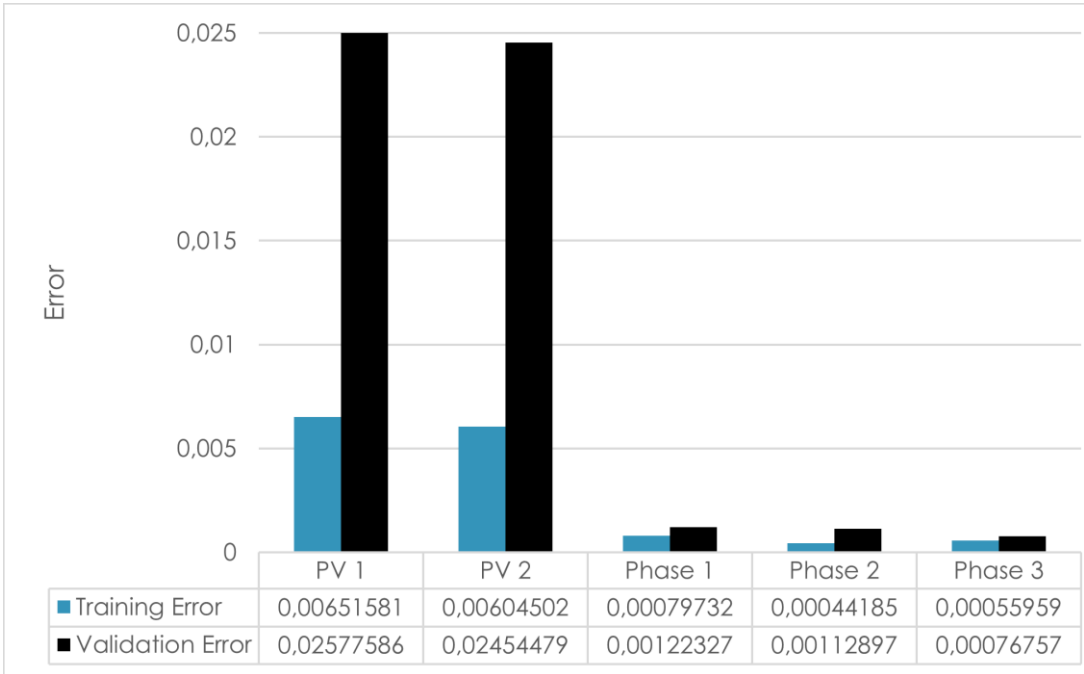
Benchmark	Training Error	Validation Error	LagWindowSize	NoOfHiddenNeurons
PV 1	0,006515	0,025775	249	513
PV 2	0,006045	0,024544	249	513
Phase 1	0,000797	0,001223	190	538
Phase 2	0,000441	0,001128	249	513
Phase 3	0,000559	0,000767	61	381

Figure V.1-4 Optimal MLP networks per benchmark (table)

Visualizing this data in a chart (Figure V.1-5) shows, that the prediction performance for photovoltaics production is much worse than the performance of energy consumption prediction.

While both performances are still reasonably good, this outcome is somewhat surprising, because PV production is supposed to have clearer repeating patterns. A closer look reveals, that the error is related to erroneous or missing

input data. As discussed in the conclusion to the "State of the art" chapter[1], enhancing the preprocessing and cleaning of the input data is likely to improve the prediction performance and is therefore suggested to be further examined in the "Next steps" (Chapter V.4 on page 58).



	PV 1	PV 2	Phase 1	Phase 2	Phase 3
Training Error	0,00651581	0,00604502	0,00079732	0,00044185	0,00055959
Validation Error	0,02577586	0,02454479	0,00122327	0,00112897	0,00076757

Figure V.1-5 Optimal MLP networks per benchmark (chart)

The result, with different optimal configurations for different prediction problems, also approves the "State of the art" conclusion, that "there is not yet a widely accepted, reliable way to determine the optimal architecture". This in turn justifies the application of a methodic search within the design space using a specialized toolchain based on FADSE.

V.2 Simulation of proposed architecture

The best machine learning networks generated by the development toolchain are serialized to files and injected to the implementation of the proposed architecture. While running the simulation using the ProHybridSimulator[2] the predictions are recorded. This data is visualized in the following tables and charts.

[1] See "Conclusion" within Chapter III.1 "State of the art" on page 28
[2] See Chapter IV.2.2 "EssListener and ProHybridSimulator" on page 41

V.2.1 Predictor performance

On each arrival of new data, the Predictor is calculating the output of its machine learning method and generates its predictions.

At each timestamp, "PV 1 Prediction Agent" creates a prediction for the next three hours in five minutes steps as defined in the FADSE configuration[1]. Figure V.2-1 shows an example output with three predictions, each covering a period of up to 25 minutes.

Timestamp of prediction	***Now***	**Prediction for**					
		in 5 m	**in 10 m**	**in 15 m**	**in 20 m**	**in 25 m**	…
07.01.2015 11:00	*588,20*	631,11	720,58	665,97	464,27	541,07	
07.01.2015 11:05	*654,60*	682,62	781,99	710,30	562,06	577,57	
07.01.2015 11:10	*641,00*	692,48	781,52	731,98	476,15	585,47	
…							

Figure V.2-1 Direct Predictor output for PV 1 per prediction timestamp

To visualize the prediction accuracy in the demonstrative application, these predictions are collected and transformed to another format, from the perspective of the predicted timestamp.

Figure V.2-2 shows the transformed table. The colored cells illustrate the new perspective with regard to the previous table. The difference between the "Now" column and the respective "Prediction from" column is the prediction error.

Predicted Timestamp	***Now***	**Prediction from**					
		5 m ago	**10 m ago**	**15 m ago**	**20 m ago**	**25 m ago**	…
07.01.2015 11:00	*588,20*	551,64	543,77	362,92	391,15	682,39	
07.01.2015 11:05	*654,60*	631,11	525,87	531,83	368,17	456,30	
07.01.2015 11:10	*641,00*	682,62	720,58	606,57	467,90	388,88	
…							

Figure V.2-2 Transformed Predictor output for PV 1 per predicted timestamp

The following figures show the transformed Predictor output for a period of three days:

[1] See Chapter IV.1.1 "Definition of the optimization problem" on page 38

- Figure V.2-3 shows PV 1 with predictions from 5 minutes ago
- Figure V.2-4 shows PV 1 with predictions from 60 minutes ago
- Figure V.2-5 shows Phase 1 with predictions from 60 minutes ago

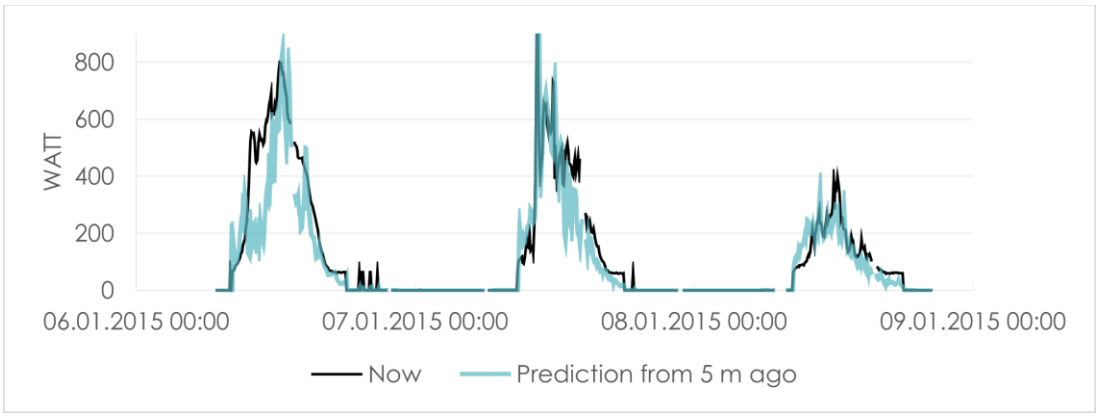


Figure V.2-3 PV 1 with predictions from 60 minutes ago

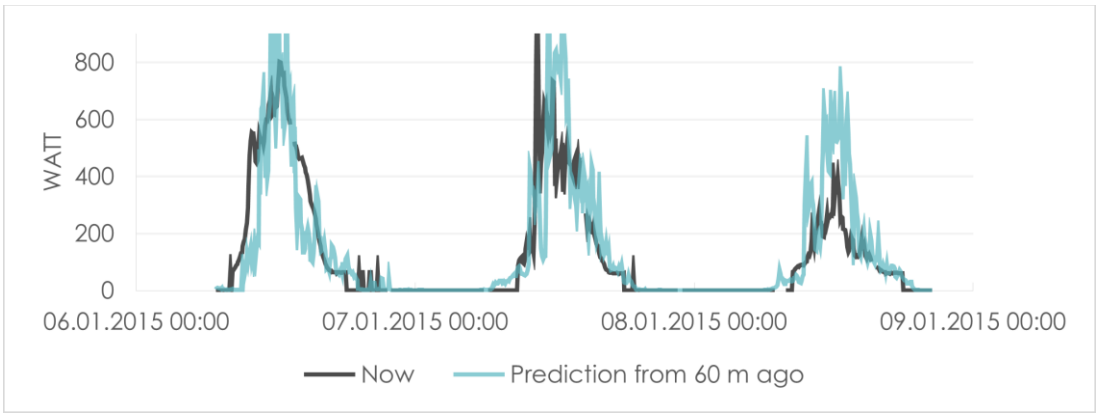


Figure V.2-4 PV 1 with predictions from 60 minutes ago

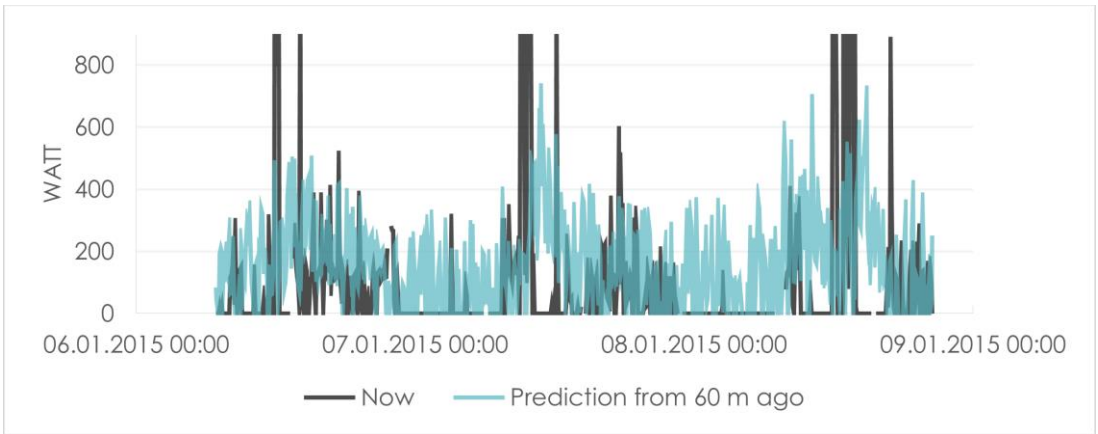


Figure V.2-5 Phase 1 with predictions from 60 minutes ago

V.2.2 Scheduler Agent

The output of the Scheduler Agent is the Schedule. Figure V.2-6 shows the approach of creating a Schedule at the timestamp "07.01.2015 11:00" for actions to be executed after five minutes in the future.

Time / Agent	07.01.2015 11:05
PV1 Prediction Agent	631,11 W
PV2 Prediciton Agent	321,89 W
Category PV	**953 W**
Grid Phase 1 Prediction Agent	∞ W
Grid Phase 2 Prediction Agent	∞ W
Grid Phase 3 Prediction Agent	∞ W
Category Grid	**∞ W**
Phase 1 Prediction Agent	380,98 W
Phase 2 Prediction Agent	75,84 W
Phase 3 Prediction Agent	399,12 W
Total Consumption Prediction	**855,94 W**
Heating Device Load Agent	
Consumption	500 W
Added Value	> Category PV
Scheduler Agent	
Supplied by PV	855,94 W
Remaining from PV	97,06 W
Supplied by Grid	0 W
Remaining from Grid	∞ W
Schedule	
Heating Device Agent	97,06 W < 500 W ⇒ No scheduling

Figure V.2-6 Creation of a Schedule for a specific timestamp

The same calculation happens for every future timestamp where a prediction exists. Figure V.2-7 shows a resulting "Schedule for the upcoming 20 minutes" according to the current value and the predictions at the timestamp "07.01.2015 11:00".

Time / Agent	11:00 (now)	11:05 (in 5 m)	11:10 (in 10 m)	11:15 (in 15 m)	11:20 (in 20 m)
Category PV [W]	1167,00	953,00	1013,75	1095,70	851,63
Category Grid [W]	∞	∞	∞	∞	∞
Total Consumption Prediction	0,00	855,94	1064,97	1148,49	963,45
Scheduler Agent					
Supplied by PV	0,00	855,94	1013,75	1095,70	851,63
Remaining from PV	1167,00	97,06	0,00	0,00	0,00
Supplied by Grid	0,00	0	51,22	52,79	111,82
Remaining from Grid	∞	∞	∞	∞	∞
Schedule					
Heating Device Agent	✓	✕	✕	✕	✕

Figure V.2-7 Schedule for the upcoming 20 minutes

V.3 Management of real loads

The actual implementation of the proposed architecture into the FEMS binding for openHAB is not in the scope of this thesis. The effect of an energy management system can still be visualized by existing examples, where FEMS was already applied. Figure V.3-1 shows the effect of a simple SOC based load management[1] to control a heating device.

In the morning hours, when the sun rises and the photovoltaic system starts to produce electricity, the battery of the energy storage system charges. When it is full, the SOC based rule activates and deactivates the heating device in order to consume the excess electricity that would be lost without an energy management system. Because of this, the curves for energy consumption and electricity production are nearly congruent throughout the day.

With sunset in the evening, the rule is inactive and the remaining consumptions are supplied by the full battery.

[1] See Chapter IV.3.2.3 "Controlling external devices" on page 46

The brown color shows, that in a few moments electricity needs to be bought from the power grid. This is for electrical reasons, as supplying very high loads per single phase from the battery is limited by the storage system.

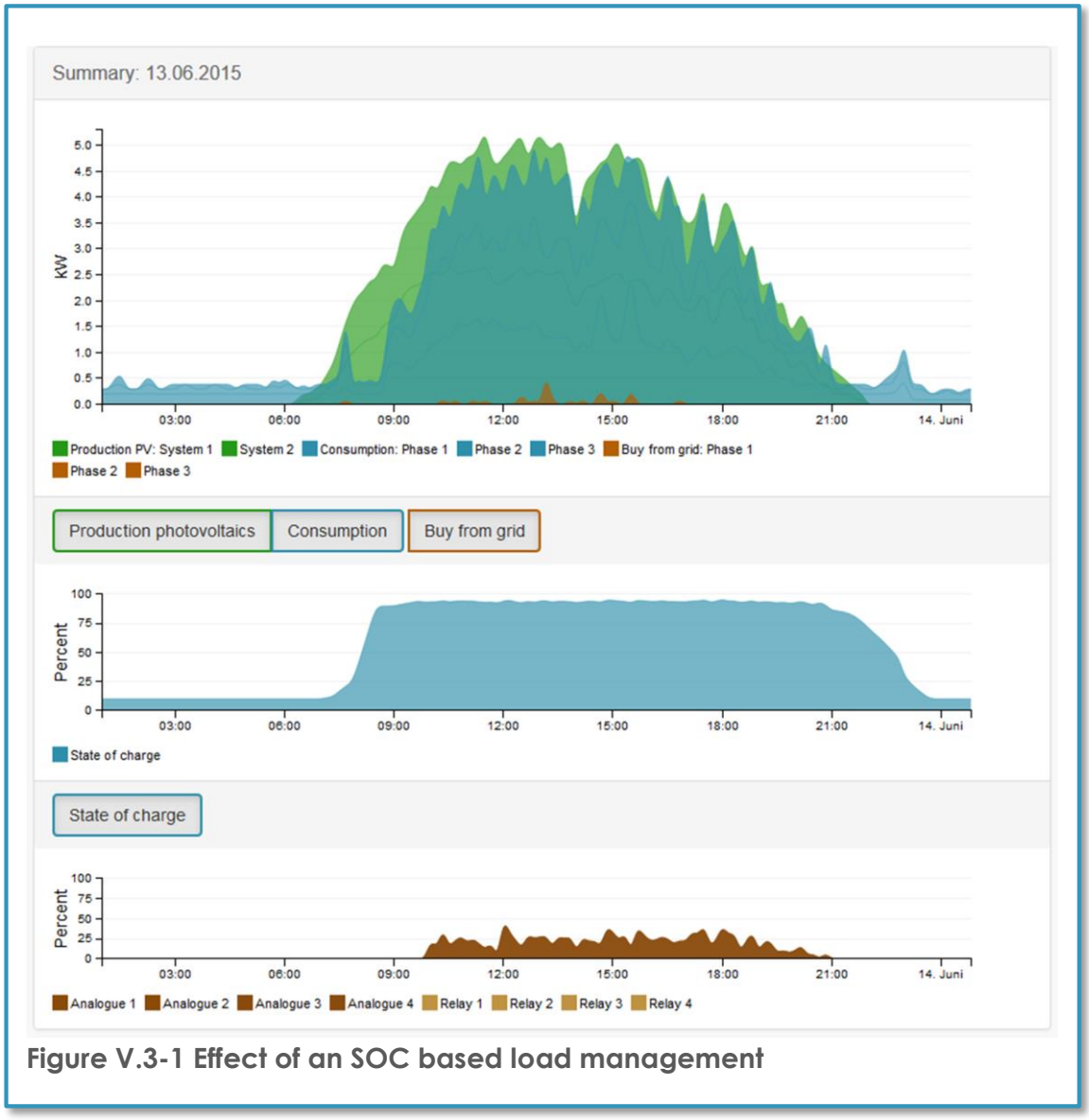


Figure V.3-1 Effect of an SOC based load management

V.4 Next steps

The architecture and toolchain were purposely designed in a way to enable further studies and improvements in a flexible, modular manner. Being a complex system, there are possible improvements of many different kinds.

An important necessity, to make the concept usable for real energy management systems, is to connect it with the FEMS binding[1]. It is advisable to

[1] See Chapter IV.3.2.4 “Attaching the proposed architecture” on page 47

implement the agents as OSGi containers, to fit seamlessly into the openHAB ecosystem and to be easily adjustable to different energy storage system configurations.

To be usable by end customers, a user-friendly interface to monitor the Schedule and to configure manageable loads needs to be implemented. The method of visualizing the Schedule as a Gantt diagram[1] seems to be sensible.

With regard to the Predictors, it became clear, that further preprocessing and cleaning of the input data is required to improve the prediction performance. Also the introduction of additional input variables like the moment of the year, the moment of the day, temperature, holiday settings and more, as suggested in the "State of the art"[2] should be evaluated.

The development toolchain with FADSE and Encog was reduced to evaluating Multi-Layer perceptron networks. It is feasible to try and evaluate further machine learning methods using this flexible and effective toolchain, in order to improve the overall prediction performance.

At the moment, the optimized machine learning method is generated only once and injected into the energy management system. To adjust itself to changing patterns, it is required to constantly learn from new data. This "online learning" approach is well supported by Encog and should be implemented at the Predictors level.

[1] See Figure IV.2-1 "Mockup: Visualized schedule using Gantt chart" on page 43 and Figure V.2-7 "Schedule for the upcoming 20 minutes" on page 57
[2] See Chapter III.1 "State of the art", especially the "Conclusion" chapter on page 28

VI. Conclusion

The scope of this paper was defined in the "Purpose of the thesis"[1] chapter in the "Introduction". With a first implementation of the proposed architecture for an energy management system, the development of a basic supporting toolchain as well as the successful evaluation of the approach, the set targets are accomplished.

While simulating the scheduling, it became clear, that the agent-based architecture is working well and is flexible enough for complex management scenarios. In addition, the external preparation and injection of optimized machine learning methods proved to be practicable.

As discussed in the "Next steps" chapter there are still a number of tasks to accomplish, before the system can be properly used in a real energy management system. The development of a user-friendly interface has a high priority, to enable a wide adaptation and implementation of the system. Further investigation is also required into the prediction accuracy of the underlying predictors, as they highly influence the overall performance of the system.

Even though, the results achieved by this first implementation of the proposed architecture are promising and already prove, that performance and flexibility are far superior to the previous approach with simple SOC based rules.

Decentralized electricity production using photovoltaics is a feasible vision for an ecologically and economically sustainable energy supply. In combination with an energy storage system the fluctuating nature of solar power can be overcome. Improving the production and consumption of electricity with an intelligent energy management system is the purpose of this dissertation – and my contribution to an "Urgent and concrete action […] to address climate change"[2].

[1] See Chapter I.2 "Purpose of the thesis" on page 9
[2] As demanded by the G7, see Chapter I "Introduction" on page 7

References

Adel Mellit, Alessandro Massi Pavan. "Performance prediction of 20kWp grid-connected photovoltaic plant at Trieste (Italy) using artificial neural network." *Energy Conversion and Management, Volume 51, Issue 12*, 2010.

Almonacid, F. , P. J. Pérez-Higueras, Eduardo F. Fernández, and L. Hontoria. "A methodology based on dynamic artificial neural network for short-term forecasting of the power output of a PV generator." *Energy Conversion and Management, Volume 85*, 11 February 2014: 389-398.

Bouzerdoum, M., A. Mellit, and A. Massi Pavan. "A hybrid model (SARIMA–SVM) for short-term power forecasting of a small-scale grid-connected photovoltaic plant." *Solar Energy*, 23 September 2013: 226–235.

Bundesanzeiger. "Gesetz für den Vorrang Erneuerbarer Energien (Erneuerbare-Energien-Gesetz – EEG)." 29 March 2000. https://www.bgbl.de/banzxaver/bgbl/start.xav?startbk=Bundesanzeiger_BGBl&jumpTo=bgbl100s0305.pdf (accessed June 18, 2015).

—. "Gesetz über die Einspeisung von Strom aus erneuerbaren Energien in das öffentliche Netz (Stromeinspeisungsgesetz)." 07 December 1990. http://archiv.jura.uni-saarland.de/BGBl/TEIL1/1990/19902633.A10.HTML (accessed June 18, 2015).

Calborean, Horia. "Multi-Objective Optimization of Advanced Computer Architectures using Domain-Knowledge." *PhD Thesis, "L. Blaga" University of Sibiu, PhD Supervisor Prof. Univ. Dr. Ing. Lucian Vintan*, 25 November 2011.

Changsong Chen, Shanxu Duan,Tao Cai,Bangyin Liu. "Online 24-h solar power forecasting based on weather type classification using artificial neural network." *Solar Energy*, November 2011: 2856–2870.

Chen, Changsong, Shanxu Duan, Tao Cai, and Bangyin Liu. "Online 24-h solar power forecasting based on weather type classification using artificial neural network." *Solar Energy*, 05 January 2011: 2856–2870.

Chow, Stanley K. H., Eric W. M. Lee, and Danny H. W. Li. "Short-term prediction of photovoltaic energy generation by intelligent approach." *Energy and Buildings*, 09 July 2012: 660-667.

Curry, Andrew. *Can You Have Too Much Solar Energy?* March 2013. http://www.slate.com/articles/health_and_science/alternative_energy/2013/

03/solar_power_in_germany_how_a_cloudy_country_became_the_world_leader_in_solar.html (accessed July 05, 2015).

Eckl, Fabian. "Demand side management based on a distributed energy storage system." 2014.

Escrivá-Escrivá, Guillermo, Carlos Álvarez-Bel, Carlos Roldán-Blay, and Manuel Alcázar-Ortega. "New artificial neural network prediction method for electrical consumption forecasting based on building end-uses." *Energy and Buildings, Volume 43*, 123 December 2010: 3112–3119.

European Commission. *Institute for Energy and Transport - PV potential estimation utility.* n.d. http://re.jrc.ec.europa.eu/pvgis/apps4/pvest.php (accessed April 05, 2015).

—. *Intelligent Energy Europe: Definition of grid-parity for photovoltaics and development of measures to accompany PV applications to the grid parity and beyond (PV PARITY).* December 2013. https://ec.europa.eu/energy/intelligent/projects/en/projects/pv-parity (accessed July 05, 2015).

Eurostat. *Electricity prices by type of user (EUR per kWh), Code: ten00117.* 15 June 2015. http://ec.europa.eu/eurostat/tgm/table.do?tab=table&plugin=1&language=en&pcode=ten00117 (accessed June 19, 2015).

—. *Electricity prices for domestic consumers - bi-annual data, Code: nrg_pc_204.* 07 May 2015. http://ec.europa.eu/eurostat/statistics-explained/index.php?title=Energy_price_statistics (accessed June 19, 2015).

FENECON. *Entwicklungspartnerschaft mit BYD.* May 2015. https://fenecon.de/page/unternehmen#BYD (accessed July 05, 2015).

Fernandez-Jimenez, L. Alfredo, et al. "Short-term power forecasting system for photovoltaic plants." *Renewable Energy, Volume 44*, 10 May 2011: 311–317.

Fraunhofer Institute for Solar Energy Systems ISE. "Recent Facts about Photovoltaics in Germany." 19 May 2015. http://www.ise.fraunhofer.de/en/publications/veroeffentlichungen-pdf-dateien-en/studien-und-konzeptpapiere/recent-facts-about-photovoltaics-in-germany.pdf (accessed June 19, 2015).

G7. "Summit Declaration." 08 June 2015. http://www.bundesregierung.de/Content/EN/Artikel/2015/06_en/g7-gipfel-dokumente_en.html (accessed June 18, 2015).

Gajowniczeka, Krzysztof, and Tomasz Ząbkowski. "Short term electricity forecasting using individual smart meter data." *Procedia Computer Science, Volume 35*, 2014: 589-597.

Hai-xiang Zhao, Frédéric Magoulès. "A review on the prediction of building energy consumption." *Renewable and Sustainable Energy Reviews, Volume 16*, August 2012: 3586–3592.

Heaton, Jeff. "Encog: Library of Interchangeable Machine Learning Models for Java and C#." *arXiv:1506.04776 [cs.MS]*, June 2015.

Heinrich Böll Foundation. *Energy Transition - The German Energiewende.* October 2012. http://energytransition.de/2012/10/renewable-energy-act-with-feed-in-tariffs/ (accessed June 19, 2015).

Hernandez, Luis, Carlos Baladron, Javier M. Aguiar, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. "Artificial neural networks for short-term load forecasting in microgrids environment." *Energy, Volume 75*, 03 June 2013: 252–264.

Izgi, Ercan, Ahmet Öztopal, Bihter Yerli, Mustafa Kemal Kaymak, and Ahmet Duran Şahin. "Short-mid-term solar power prediction by using artificial neural networks." *Solar Energy, Volume 86*, 24 June 2011: 725-733.

Jacobson, Lee. *Introduction to Artificial Neural Networks - Part 1.* December 05, 2013. http://www.theprojectspot.com/tutorial-post/introduction-to-artificial-neural-networks-part-1/7 (accessed June 21, 2015).

Klare, Michael. *Resource Wars: The New Landscape of Global Conflict.* Henry Holt and Company, 2002.

Mellit, Adel, and Alessandro Massi Pavan. "Performance prediction of 20kWp grid-connected photovoltaic plant at Trieste (Italy) using artificial neural network." *Energy Conversion and Management, Volume 51*, 03 July 2009: 2431-2441.

openHAB UG. *openHAB Architecture.* n.d. http://www.openhab.org/features-architecture.html (accessed 06 24, 2015).

Quaschning, Volker, Johannes Weniger, and Tjarko Tjaden. "Der unterschätze Markt." *BWK Das Energie-Fachmagazin, Bd. 64, Nr. 7/8*, 2012: 28-28.

Renusol GmbH. *East/west solar installations generate around 30 percent higher yields.* 29 October 2014. http://www.renusol.com/unternehmen/news/news-imgvid/article/973.html (accessed July 05, 2015).

Riedmiller, M., and H. Braun. "Rprop - A Fast Adaptive Learning Algorithm." *Proceedings of the International Symposium on Computer and Information Science VII*, 1992.

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." *Nature, Volume 323*, October 09, 1986: 533-536.

Russell, Stuart, and Peter Norvig. *Artificial Intelligence: A Modern Approach, 3rd Edition*. Pearson, 2010.

SolarServer. *Basiswissen: Photovoltaik-Speicher.* n.d. http://www.solarserver.de/wissen/basiswissen/photovoltaik-speicher.html (accessed June 19, 2015).

The Economist. *Daily chart: Pricing sunshine.* 28 December 2012. http://www.economist.com/blogs/graphicdetail/2012/12/daily-chart-19 (accessed July 05, 2015).

TÜV SÜD. "Basic Understanding of IEC Standard Testing For Photovoltaic Panels." 2009. https://www.tuvamerica.com/services/photovoltaics/ArticleBasicUnderstandingPV.pdf (accessed July 05, 2015).

Ward Systems Group Inc. "Neuroshell 2 Manual." 1996. http://www.wardsystems.com/manuals/neuroshell2/index.html?idxtutorialtwo.htm (accessed July 02, 2015).

List of illustrations

Figure I.1-1 Development of electricity generation from renewable souces in Germany (Heinrich Böll Foundation 2012) 8
Figure I.1-2 Household-prices per kWh of electricity including taxes in a selection of European countries in 2014 (Eurostat 2015) 8
Figure I.1-3 Prive of PV cells over time in $ per Watt (The Economist 2012) 9
Figure I.2-1 Comparing PV systems directing East/West and South (Renusol GmbH 2014) 10
Figure I.3-1 FENECON Logo 11
Figure I.4-1 Example installation: FENECON by BYD PRO Hybrid with FEMS 12
Figure I.4-2 Schema Energy Storage System 12
Figure I.4-3 Example data recorded by FEMS 13
Figure II.1-1 PV system, power grid and loads 14
Figure II.1-2 Typical PV production curve recorded by FEMS 15
Figure II.1-3 PV potential estimation utility 15
Figure II.1-4 PV system, energy storage system, power grid and loads 17
Figure II.1-5 PV system and additional energy sources, energy storage system, power grid and loads, in combination with an energy management system 19
Figure II.1-6 Energy source prioritization 19
Figure II.2-1 Energy consumption curve 20
Figure II.2-2 "Peak-Load Shifting" approach (Eckl 2014) 21
Figure II.2-3 Added value per kWh for electric car 22
Figure II.2-4 Added value per kWh for water heating device 22
Figure II.3-1 PV system and additional energy sources, energy storage system, power grid and manageable loads, in combination with an energy management system 23
Figure II.3-2 Proposed architecture 24
Figure II.3-3 Scheduling example 25
Figure III.2-1 Schema of a perceptron 30
Figure III.2-2 Schema of a Multi-Layer Perceptron network 31
Figure III.2-3 Overfitting 32
Figure III.2-4 Sliding Window technique 33
Figure III.2-5 General approach: One Predictor per specific problem 34
Figure III.2-6 Simplified predictor design space 36
Figure III.2-7 Toolchain for creation and injection of an Encog ML method 37
Figure IV.1-1 FADSE XML definition 39
Figure IV.2-1 Mockup: Visualized schedule using Gantt chart 43
Figure IV.3-1 FEMS device 44
Figure IV.3-2 Eclipse SmartHome and openHAB logos 45
Figure IV.3-3 openHAB Architecture Overview (openHAB UG) 46

Figure IV.3-4 openHAB rule "Water heater" 47
Figure IV.3-5 FENECON Online-Monitoring 48
Figure V.1-1 Individuals generated by FADSE 50
Figure V.1-2 Training and validation error per training iteration 51
Figure V.1-3 Evaluation of individuals according to the training and validation error 52
Figure V.1-4 Optimal MLP networks per benchmark (table) 52
Figure V.1-5 Optimal MLP networks per benchmark (chart) 53
Figure V.2-1 Direct Predictor output for PV 1 per prediction timestamp 54
Figure V.2-2 Transformed Predictor output for PV 1 per predicted timestamp 54
Figure V.2-3 PV 1 with predictions from 60 minutes ago 55
Figure V.2-4 PV 1 with predictions from 60 minutes ago 55
Figure V.2-5 Phase 1 with predictions from 60 minutes ago 55
Figure V.2-6 Creation of a Schedule for a specific timestamp 56
Figure V.2-7 Schedule for the upcoming 20 minutes 57
Figure V.3-1 Effect of an SOC based load management 58

List of equations

Equation II.1-1 Cost per produced kWh from photovoltaics 16
Equation III.1-1 Rule of thumb for the number of hidden neurons 27
Equation III.2-1 Perceptron forward-propagation 30
Equation III.2-2 Calculation of the error during back-propagation 31
Equation III.2-3 Adjustment of the perceptron's weights 31

List of abbreviations

AC *Alternating Current*
AI *Artificial Intelligence*
ANN *Artificial Neural Network*
BMS *Battery Management System*
DC *Direct Current*
EEG *Erneuerbare-Energien-Gesetz (Renewable Energy Act)*
ESS *Energy Storage System, Energy Storage System*
FADSE *Framework for Automatic Design Space Exploration*
FEMS *FENECON Energy Management System*
kWh *kilowatt hour*
kWp kilowatt-peak
LCC *Life-cycle cost*
LCoE *Levelized Cost Of Electricity*
LMA *Levenberg Marquardt algorithm*
ML *Machine Learning*
MLP *Multi-Layer Perceptron*
MW *Megawatt*
PV *Photovoltaics*
PVGIS *Photovoltaic Geographical Information System*
RBFN *Radial Basis Function Networks*
RMSE *Root Mean Square Error*
SOC *State of charge*
STC *Standard Test Conditions*
SVM State Vector Machines
W *Watt*
Wp *Watt-peak*

Annexes

A. Datasheet: FENECON by BYD PRO Hybrid 70
B. Source code: development toolchain 72
B.1 FemsPredictorSimulator.java 72
B.2 fems.FemsConstants.java 75
B.3 fems.FemsSimulatorParameters.java 76
B.4 fems.FemsIndividualParameters.java 78
B.5 fems.FemsPredictor.xml 80
B.6 fems.FemsTimeSeriesSimulator.java 81
B.7 fems.Normalizer.java 86
C. Source code: implementation of the proposed architecture 88
C.1 de.fenecon.fems 88
C.1.1 FemsApp.java 88
C.1.2 FemsConstants.java 90
C.1.3 FemsTools.java 92
C.2 de.fenecon.fems.agent.consumption 94
C.2.1 ConsumptionAgent.java 94
C.2.2 ConsumptionAgentFactory.java 95
C.2.3 ConsumptionAgentImpl.java 96
C.2.4 ConsumptionField.java 97
C.3 de.fenecon.fems.agent.load 98
C.3.1 HeatingDeviceLoadAgent 98
C.3.2 LoadAction.java 99
C.3.3 LoadAgent.java 100
C.4 de.fenecon.fems.agent.scheduler 102
C.4.1 SchedulerAgent.java 102
C.5 de.fenecon.fems.agent.source 111
C.5.1 SourceAgent.java 111
C.5.2 SourceCategory.java 112
C.5.3 grid.GridAgent.java 113

C.5.4 grid.GridAgentFactory.java 115
C.5.5 grid.GridField.java 116
C.5.6 pv.PvAgent.java 117
C.5.7 pv.PvAgentFactory.java 118
C.5.8 pv.PvField.java 119
C.6 de.fenecon.fems.ess 120
C.6.1 EssListener.java 120
C.6.2 ess.prohybrid.ProHybridSimulator.java 121
C.6.3 ess.prohybrid.ProHybridSimulatorFactory.java 123
C.7 de.fenecon.fems.helper 124
C.7.1 DownloadDataApp.java 124
C.7.2 Field.java 127
C.7.3 Normalizer.java 129
C.7.4 Prediction.java 131
C.7.5 PredictionAgent.java 132
C.7.6 PredictionAgentFactory.java 133
C.7.7 PredictionAgentImpl.java 135
C.7.8 Predictor.java 138
C.7.9 RingCache.java 140
D. Eclipse Public License - v 1.0 141

A. Datasheet: FENECON by BYD PRO Hybrid

Technical features

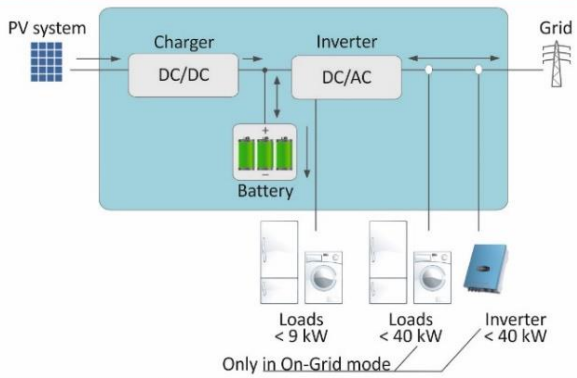
- **$LiFePO_4$** battery technology – offers maximum security with usable capacity of **8,5 kWh**, **17 kWh** or **25,5 kWh**
- **Hybrid** (both-sided) **charging** of the battery from AC and DC sources
- Load-supply via **three integrated bidirectional inverters**
- **Three-phase alternating current** in On/Off-Grid mode
- Designed for **parallel generation** as well as for **island-mode** (Recharging of battery using DC-PV-system in the event of a power failure)
- Max. generator connected power: **40 kW AC connection** and **2 x 4 kW DC connection**

The FENECON by BYD **PRO Hybrid** is a compact and intelligent system for storage of electrical energy in order to increase the personal electricity **self-consumption** and **self-coverage**.

Hybrid stands for the possibility to use AC and DC generators to charge the **lithium iron phosphate battery** as well as the possibility to supply two separate, independent load outputs.

The storage system offers highest flexibility in connecting **PV installations**, **windmills** and **block heating stations**.

System structure



Product designation

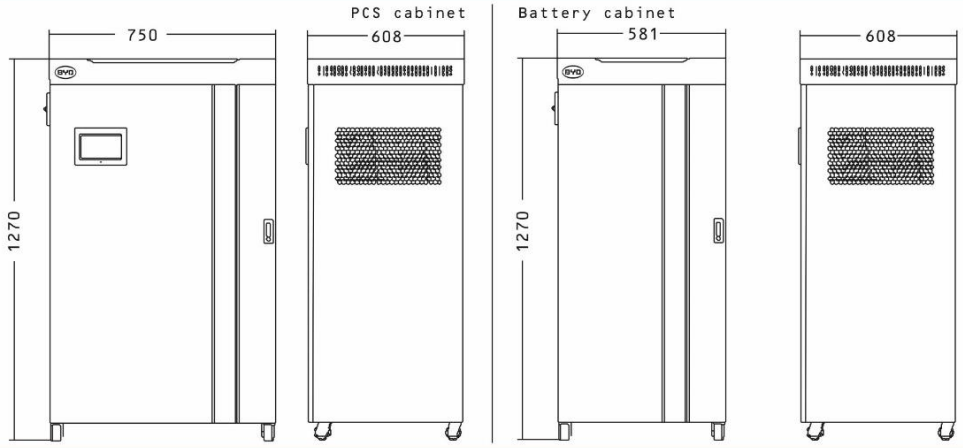
DESS P09 B10 - HC08

- HC08: Hybrid; 8 kW charger performance for DC connection (PV module)
- B10: 10 kWh battery capacity
- P09: 3 x 3 kW charge/discharge capacity (three-phase)
- DESS: Distributed Energy Storage System

Your energy storage expert:

FENECON GmbH & Co. KG
Brunnwiesenstraße 4
94469 Deggendorf, Germany
Tel: +49 991 648800-00
Fax: +49 991 648800-09
E-Mail: info@fenecon.de
www.fenecon.de

Dimensions



Technical parameters

		PRO Hybrid 9-10	PRO Hybrid 9-20	PRO Hybrid 9-30
System type		P09B10-HC08	P09B20-HC08	P09B30-HC08
Max. output current		3 x 30 A (small load) / 3 x 63 A (big load)		
Max. AC generator power		40 kW (charger power max. 9 kW)		
Inverter	Nominal output	3 x 3 kVA		
	Nominal voltage	230 V / 400 V		
	Frequency	47,5 - 51,5 Hz		
	Max. output current	3 x 13,1 A		
	Power factor (cos ϕ)	0,9 (inductive) - 0,9 (capacitive)		
	Efficiency	93 %		
	Switching time UPS	< 200 ms		
	THD	< 4 % (current in grid mode), < 2 % (voltage in island mode)		
PV Charger	PV output	2 x 4 kWp		
	Open-circuit voltage	65 - 145 V_{DC}		
	MPP voltage	70 - 120 V_{DC}		
	MPP tracker	2		
	Max. input current	2 x 64 A		
	Efficiency	97,3 %		
Battery	Nominal voltage	51,2 V_{DC}		
	Cell type	$LiFePO_4$		
	No. of cycles	6.000 (until 80 % residual capacity)		
	Capacity	10 kWh	20 kWh	30 kWh
	Depth of discharge DOD	85 %		
	Battery management	Yes		
	Active balancing	Yes		
Warranty		5 years complete product warranty, 7 years time value replacement warranty according to KfW-terms		
Certifications		CE, low voltage directive (VDE-AR-N 4105), type test (IEC 62109-1:2010), EMC (EN 61000-3-3:2008, EN 61000-3-2:2009, EN 55022:2010), dangerous goods (UN 38.3)		
Interfaces		RS485, Ethernet		
IP protection class		IP 20		
Temperature range		0 - 45 °C		
Humidity		10 - 90 %		
Altitude above mean sea level		< 2.000 m		
Max. sound level		65 dB		
Dimensions (W/D/H)	PCS	750 x 608 x 1.270 mm		
	Battery	581 x 608 x 1.270 mm	2 * 581 x 608 x 1.270 mm	3 * 581 x 608 x 1.270 mm
Weight	PCS + Battery	210 kg + 206 kg	210 kg + 2 * 206 kg	210 kg + 3 * 206 kg

FENECON-06/2015

B. Source code: development toolchain

B.1 FemsPredictorSimulator.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.Reader;
import java.nio.file.Paths;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;
import java.util.Properties;
import java.util.TreeMap;

import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVRecord;
import org.encog.Encog;

import ro.ulbsibiu.fadse.environment.Environment;
import ro.ulbsibiu.fadse.environment.Individual;
import ro.ulbsibiu.fadse.environment.Objective;
import ro.ulbsibiu.fadse.extended.problems.simulators.fems.FemsConstants;
import ro.ulbsibiu.fadse.extended.problems.simulators.fems.FemsIndividualParameters;
import ro.ulbsibiu.fadse.extended.problems.simulators.fems.FemsSimulatorParameters;
import ro.ulbsibiu.fadse.extended.problems.simulators.fems.FemsTimeSeriesSimulator;
import ro.ulbsibiu.fadse.extended.problems.simulators.fems.Normalizer;

/**
 * Implements a Simulator for FADSE to provide a development toolchain for FEMS (FENECON
 * Energy Management System).
 *
 * @author Stefan Feilmeier
 */
public class FemsPredictorSimulator extends SimulatorBase {
  private static final long serialVersionUID = 5241265466817140328L;

  private final HashSet<String> fields = new HashSet<String>();
  private final TreeMap<Long, HashMap<String, Double>> fieldsPerTimestamp =
      new TreeMap<Long, HashMap<String, Double>>();
  private final FemsSimulatorParameters simulatorParameters;

  public FemsPredictorSimulator(Environment environment) throws ClassNotFoundException {
    super(environment);

    // Set simulator parameters
    FemsSimulatorParameters simulatorParameters;
    try {
      simulatorParameters =
          FemsSimulatorParameters.factory(environment.getInputDocument());
    } catch (Exception e) {
      e.printStackTrace();
      // unfortunately we can only throw ClassNotFoundException
      throw new ClassNotFoundException(e.getMessage());
    }
    this.simulatorParameters = simulatorParameters;

    // Initialize output path
    for (File file : simulatorParameters.getOutputPath().listFiles()) {
      file.delete();
    }
```

```
HashMap<String, Double> maxValuesPerField = new HashMap<String, Double>();
HashMap<String, Double> minValuesPerField = new HashMap<String, Double>();

// Read CSV file
try {
  Reader in =
      new FileReader(Paths.get(FemsConstants.FILESPATH,
          simulatorParameters.getDevice() + ".csv").toFile());
  final Iterable<CSVRecord> records = CSVFormat.DEFAULT.withHeader().parse(in);
  for (CSVRecord record : records) {
    HashMap<String, Double> valuesPerField = new HashMap<String, Double>();
    for (Map.Entry<String, String> valuePerField : record.toMap().entrySet()) {
      String field = valuePerField.getKey();
      if (field.equals("timestamp"))
        continue; // ignore timestamp
      Double value = Double.parseDouble(valuePerField.getValue());
      valuesPerField.put(field, value);
      fields.add(field); // Get list of fields

      // Get max/min value per field for normalizer
      Double previousMaxValuePerField = maxValuesPerField.get(field);
      if (previousMaxValuePerField == null || previousMaxValuePerField < value) {
        maxValuesPerField.put(field, value);
      }
      Double previousMinValuePerField = minValuesPerField.get(field);
      if (previousMinValuePerField == null || previousMinValuePerField > value) {
        minValuesPerField.put(field, value);
      }
    }
    Long timestamp = Long.parseLong(record.get("timestamp"));
    fieldsPerTimestamp.put(timestamp, valuesPerField);
  }
} catch (Exception e) {
  e.printStackTrace();
  // unfortunately we can only throw ClassNotFoundException
  throw new ClassNotFoundException(e.getMessage());
}

// Prepare Normalizer
HashMap<String, Normalizer> normalizers = new HashMap<String, Normalizer>();
for (String field : fields) {
  Normalizer normalizer =
      new Normalizer(minValuesPerField.get(field), maxValuesPerField.get(field), 0,
          1);
  normalizers.put(field, normalizer);
}

// Write Properties file
Properties properties = new Properties();
properties.put("Device", simulatorParameters.getDevice());
properties.put("LeadWindowSize",
    Integer.toString(simulatorParameters.getLeadWindowSize()));
for (Map.Entry<String, Normalizer> normalizerPerField : normalizers.entrySet()) {
  String prefix = "Normalizer_" + normalizerPerField.getKey() + "_";
  Normalizer norm = normalizerPerField.getValue();
  properties.setProperty(prefix + "DataLow", Double.toString(norm.getDataLow()));
  properties.setProperty(prefix + "DataHigh", Double.toString(norm.getDataHigh()));
  properties.setProperty(prefix + "NormalizedLow",
      Double.toString(norm.getNormalizedLow()));
  properties.setProperty(prefix + "NormalizedHigh",
      Double.toString(norm.getNormalizedHigh()));
}
FileWriter writer = null;
try {
  writer =
      new FileWriter(new File(simulatorParameters.getOutputPath(),
          "properties.prop"), false);
  properties.store(writer, "Generated by FADSE for FEMS");
```

```
    } catch (Exception e) {
      e.printStackTrace();
      // unfortunately we can only throw ClassNotFoundException
      throw new ClassNotFoundException(e.getMessage());
    }

    // Normalize dataCache
    for (Map.Entry<Long, HashMap<String, Double>> fieldPerTimestamp : fieldsPerTimestamp
        .entrySet()) {
      HashMap<String, Double> valuesPerField = fieldPerTimestamp.getValue();
      for (Map.Entry<String, Double> valuePerField : valuesPerField.entrySet()) {
        String field = valuePerField.getKey();
        Double value = valuePerField.getValue();
        Double normValue = normalizers.get(field).normalize(value);
        valuePerField.setValue(normValue);
      }
    }
  }

  /**
   * Executes the simulation and return its system metrics
   *
   * @param individual
   *          the specific individual for this run
   */
  @Override
  public void performSimulation(Individual individual) {
    // Set individual parameters
    FemsIndividualParameters individualParameters;
    try {
      individualParameters = FemsIndividualParameters.factory(individual);
    } catch (Exception e1) {
      e1.printStackTrace();
      return;
    }
    System.out.println();
    System.out.println("Start FEMS Predictor Simulation:");
    System.out.println("     " + simulatorParameters.toString());
    System.out.println("     " + individualParameters.toString());

    // Run simulation
    FemsTimeSeriesSimulator simulator = new FemsTimeSeriesSimulator();
    Map<String, Double> results =
        simulator.performSimulation(simulatorParameters, individualParameters,
            fieldsPerTimestamp);
    for (Map.Entry<String, Double> result : results.entrySet()) {
      System.out.println("  " + result.getKey() + ": " + result.getValue());
    }

    // Return error objective
    List<Objective> objectives = individual.getObjectives();
    for (Objective objective : objectives) {
      objective.setValue(results.get(objective.getName()));
    }

    try {
      Thread.sleep(1000);
    } catch (InterruptedException e) {
      e.printStackTrace();
    }

    // Shutdown encog
    Encog.getInstance().shutdown();

    System.out.println("Finished FEMS PV Predictor Simulation: "
        + individualParameters.toString());
    System.out.println();
  }
}
```

B.2 fems.FemsConstants.java

```java
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators.fems;

import org.apache.commons.csv.CSVFormat;

/**
 * Defines some commonly used constants.
 *
 * @author Stefan Feilmeier
 */
public class FemsConstants {
	public final static int SLICE_SECONDS = 5*60; // length of one data slice

	public final static CSVFormat CSV_FORMAT = CSVFormat.DEFAULT;

	public final static String FILESPATH = "D:/fems/files";
}
```

B.3 fems.FemsSimulatorParameters.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators.fems;

import java.io.File;
import java.nio.file.Paths;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

import ro.ulbsibiu.fadse.environment.document.InputDocument;

/**
 * Holds the global simulator parameters as defined in the optimization description XML
 * file.
 *
 * @author Stefan Feilmeier
 */
public class FemsSimulatorParameters {
  private final File outputPath;
  private final String device;
  private final Integer leadWindowSize;

  /**
   * Factory method to create a new FemsSimulatorParameters object from an input
   * document XML file
   *
   * @param inputDocument
   *          the InputDocument from FADSE
   * @return a new FemsSimulatorParameters object
   * @throws Exception
   *           if the object cannot be created from the inputdocument
   */
  public static FemsSimulatorParameters factory(InputDocument inputDocument)
      throws Exception {

    String device = inputDocument.getSimulatorParameter("Device");
    Integer leadWindowSize =
        Integer.parseInt(inputDocument.getSimulatorParameter("LeadWindowSize"));
    // Set output path
    File outputPath = Paths.get(inputDocument.getOutputPath()).toFile();
    if (!outputPath.exists()) {
      throw new Exception("Simulator output path does not exist: "
          + outputPath.toString());
    }
    return new FemsSimulatorParameters(outputPath, device, leadWindowSize);
  }

  @Override
  public String toString() {
    return "FemsSimulatorParameters [device=" + device + ", leadWindowSize="
        + leadWindowSize + "]";
  }

  /**
   * Creates a new object with the defined simulator parameters.
   *
   * @param outputPath  the output directory
   * @param device  the device identifier
   * @param leadWindowSize  the size of the lead window
   */
  public FemsSimulatorParameters(File outputPath, String device, int leadWindowSize) {
    this.outputPath = outputPath;
    this.device = device;
    this.leadWindowSize = leadWindowSize;
  }
```

```
  /**
   * Get the output path
   *
   * @return the output path
   */
  public File getOutputPath() {
    return outputPath;
  }

  /**
   * Get the device identifier
   *
   * @return the device identifier
   */
  public String getDevice() {
    return device;
  }

  /**
   * Get the size of the lead window
   *
   * @return the lead window size
   */
  public int getLeadWindowSize() {
    return leadWindowSize;
  }

  /**
   * Return the parameter names, suitable for writing a CSV file.
   *
   * @return a List of the parameter names
   */
  public static List<String> getCsvHeaders() {
    return new LinkedList<String>(Arrays.asList("Device", "LeadWindowSize"));
  }

  /**
   * Return the current parameters, suitable for writing a CSV file.
   *
   * @return a list of the parameters
   */
  public List<String> getCsvLine() {
    return new LinkedList<String>(Arrays.asList(device, leadWindowSize.toString()));
  }
}
```

B.4 fems.FemsIndividualParameters.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators.fems;

import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;

import ro.ulbsibiu.fadse.environment.Individual;
import ro.ulbsibiu.fadse.environment.parameters.Parameter;

/**
 * Holds the parameters of one individual as created by FADSE.
 *
 * @author Stefan Feilmeier
 */
public class FemsIndividualParameters {
  public static FemsIndividualParameters factory(Individual individual)
      throws Exception {
    // Parse benchmark
    String field = individual.getBenchmark();

    // Parse parameters
    Integer noOfHiddenNeurons = null;
    Integer lagWindowSize = null;
    for (Parameter parameter : individual.getParameters()) {
      String name = parameter.getName();
      if (name.equals("NoOfHiddenNeurons")) {
        noOfHiddenNeurons = (Integer) parameter.getValue();
      } else if (name.equals("LagWindowSize")) {
        lagWindowSize = (Integer) parameter.getValue();
      }
    }
    if (noOfHiddenNeurons == null) {
      throw new Exception("Missing Parameter: NoOfHiddenNeurons");
    } else if (lagWindowSize == null) {
      throw new Exception("Missing Parameter: LagWindowSize");
    }
    return new FemsIndividualParameters(field, noOfHiddenNeurons, lagWindowSize);
  }

  final private String field;
  final private int lagWindowSize;
  final private int noOfHiddenNeurons;

  /**
   * Creates a new object with the defined parameters of the individual.
   *
   * @param field
   *          the field identifier
   * @param noOfHiddenNeurons
   *          the number of hidden neurons
   * @param lagWindowSize
   *          the size of the lag window
   */
  public FemsIndividualParameters(String field, int noOfHiddenNeurons, int lagWindowSize) {
    this.field = field;
    this.noOfHiddenNeurons = noOfHiddenNeurons;
    this.lagWindowSize = lagWindowSize;
  }

  @Override
  public String toString() {
    return "FemsIndividualParameters [field=" + field + ", lagWindowSize="
        + lagWindowSize + ", noOfHiddenNeurons=" + noOfHiddenNeurons + "]";
  }
```

```
  /**
   * Return the parameter names, suitable for writing a CSV file.
   *
   * @return a List of the parameter names
   */
  public static List<String> getCsvHeaders() {
    return new LinkedList<String>(Arrays.asList("Field", "NoOfHiddenNeurons",
        "LagWindowSize"));
  }

  /**
   * Return the current parameters, suitable for writing a CSV file.
   *
   * @return a list of the parameters
   */
  public List<String> getCsvLine() {
    return new LinkedList<String>(Arrays.asList(field,
        Integer.toString(noOfHiddenNeurons), Integer.toString(lagWindowSize)));
  }

  /**
   * Get the field identifier.
   *
   * @return the field
   */
  public String getField() {
    return field;
  }

  /**
   * Get the lag window size.
   *
   * @return the lag window size
   */
  public int getLagWindowSize() {
    return lagWindowSize;
  }

  /**
   * Get the number of hidden neurons
   *
   * @return the number of hidden neurons
   */
  public int getNoOfHiddenNeurons() {
    return noOfHiddenNeurons;
  }
}
```

B.5 fems.FemsPredictor.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<!-- Document: FemsPredictor.xml -->
<!-- Created on: June 11, 2015 -->
<!-- Author: Stefan Feilmeier -->
<!-- Description: Input file for FemsPvPredictorSimulator -->

<design_space>
  <simulator name="FemsPredictorSimulator" type="simulator">
    <parameter name="Device" value="fems20" />
    <parameter name="LeadWindowSize" value="36" /> <!-- 3 hours @ 5 minutes: 3*(60/5) -->
  </simulator>
  <database ip="localhost" port="3306" name="FemsPredictor"
    user="fadse" password="password" />
  <benchmarks> <!-- which field to predict -->
    <item name="PV1" />
    <item name="PV2" />
    <item name="Ph1" />
    <item name="Ph2" />
    <item name="Ph3" />
  </benchmarks>
  <metaheuristic name="NSGAII"
       config_path="D:\fems\fadse\trunk\src\main\java\ro\ulbsibiu\fadse\extended\problems\
         simulators\fems\nsgaii.properties" />
  <parameters>
    <parameter name="NoOfHiddenNeurons"
        description="Number of neurons on the hidden layer of the neural network"
        type="integer" min="1" max="270" />
        <!-- Applying rule of thumb: Nh = (Ni+No)/2+sqrt(Np) -->
    <parameter name="LagWindowSize" description="Lag window size for time series"
        type="integer" min="1" max="900" /> <!-- up to about 3 days -->
  </parameters>
  <virtual_parameters></virtual_parameters>
  <system_metrics>
    <system_metric name="TrainingError" type="float" unit=""
        desired="small" />
    <system_metric name="ValidationError" type="float" unit=""
        desired="small" />
  </system_metrics>
  <rules></rules>
  <relations>
  </relations>
  <output output_path="D:\fems\fadse\output" />
</design_space>
```

B.6 fems.FemsTimeSeriesSimulator.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators.fems;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;
import java.util.TreeMap;

import org.apache.commons.csv.CSVPrinter;
import org.encog.engine.network.activation.ActivationSigmoid;
import org.encog.ml.MLRegression;
import org.encog.ml.data.MLData;
import org.encog.ml.data.MLDataSet;
import org.encog.ml.data.basic.BasicMLData;
import org.encog.ml.data.basic.BasicMLDataPair;
import org.encog.ml.data.basic.BasicMLDataSet;
import org.encog.ml.train.strategy.end.SimpleEarlyStoppingStrategy;
import org.encog.neural.networks.BasicNetwork;
import org.encog.neural.networks.layers.BasicLayer;
import org.encog.neural.networks.training.propagation.resilient.ResilientPropagation;
import org.encog.persist.EncogDirectoryPersistence;

/**
 * Provides the actual logic for carrying out a simulation.
 *
 * @author Stefan Feilmeier
 */
public class FemsTimeSeriesSimulator {
  public static final String RESULTS_FILENAME = "results.csv";
  public static final double TRAINING_RATIO = 0.7;

  /**
   * This method is executed by the FADSE simulator to perform one specific simulation.
   *
   * @param simulatorParameters
   *          the global parameters of the simulator
   * @param individualParameters
   *          the parameters of the individual as created by FADSE
   * @param fieldsPerTimestamp
   *          the example dataset
   * @return
   */
  public Map<String, Double> performSimulation(
      FemsSimulatorParameters simulatorParameters,
      FemsIndividualParameters individualParameters,
      TreeMap<Long, HashMap<String, Double>> fieldsPerTimestamp) {
    // Create training & validation set
    System.out.println("  Create training & validation set");
    Long firstTimestamp = fieldsPerTimestamp.firstKey();
    Long lastTimestamp = fieldsPerTimestamp.lastKey();
    Long splitTimestamp =
        (long) ((lastTimestamp - firstTimestamp) * TRAINING_RATIO + firstTimestamp);
    TreeMap<Long, HashMap<String, Double>> trainingMap =
        new TreeMap<Long, HashMap<String, Double>>(
            fieldsPerTimestamp.headMap(splitTimestamp));
    MLDataSet mlTrainingSet =
        getMLDataSet(trainingMap, individualParameters.getField(),
            simulatorParameters.getLeadWindowSize(),
            individualParameters.getLagWindowSize());
    TreeMap<Long, HashMap<String, Double>> validationMap =
        new TreeMap<Long, HashMap<String, Double>>(
            fieldsPerTimestamp.tailMap(splitTimestamp));
```

```
MLDataSet mlValidationSet =
    getMLDataSet(validationMap, individualParameters.getField(),
        simulatorParameters.getLeadWindowSize(),
        individualParameters.getLagWindowSize());

// Create network
System.out.println("  Create network");
BasicNetwork network = new BasicNetwork();
network
    .addLayer(new BasicLayer(null, true, individualParameters.getLagWindowSize()));
// Sigmoid: range 0..1
network.addLayer(new BasicLayer(new ActivationSigmoid(), true, individualParameters
    .getNoOfHiddenNeurons()));
network.addLayer(new BasicLayer(new ActivationSigmoid(), false, simulatorParameters
    .getLeadWindowSize()));
network.getStructure().finalizeStructure();
network.reset(1001);

// Write iteration to CSV file
FileWriter iterationFileWriter = null;
CSVPrinter iterationFilePrinter = null;
try {
  File iterationCsvfile =
      new File(simulatorParameters.getOutputPath(), String.format(
          "%s_H%d_LAG%d.csv", individualParameters.getField(),
          individualParameters.getNoOfHiddenNeurons(),
          individualParameters.getLagWindowSize()));
  iterationFileWriter = new FileWriter(iterationCsvfile, false);
  iterationFilePrinter =
      new CSVPrinter(iterationFileWriter, FemsConstants.CSV_FORMAT);
  iterationFilePrinter.print("Iteration");
  iterationFilePrinter.print("TrainingError");
  iterationFilePrinter.print("ValidationError");
  // TODO: measure time per iteration and write to csv
  iterationFilePrinter.println();

  // Train network
  System.out.println("  Train network");
  final ResilientPropagation train =
      new ResilientPropagation(network, mlTrainingSet);
  // LevenbergMarquardtTraining is effective, but very slow
  // final LevenbergMarquardtTraining train = new
  // LevenbergMarquardtTraining(network, mlTrainingSet);
  SimpleEarlyStoppingStrategy earlyStop =
      new SimpleEarlyStoppingStrategy(mlValidationSet);
  train.addStrategy(earlyStop);
  while (!train.isTrainingDone()) {
    train.iteration(1);
    String iteration = Integer.toString(train.getIteration());
    String trainingError = String.format("%.5f", earlyStop.getTrainingError());
    String validationError = String.format("%.5f", earlyStop.getValidationError());
    System.out.println(String.format(
        "    Iteration: %s, Training Error: %s, Validation Error: %s", iteration,
        trainingError, validationError));
    iterationFilePrinter.print(iteration);
    iterationFilePrinter.print(trainingError);
    iterationFilePrinter.print(validationError);
    // TODO: Fix validationError
    iterationFilePrinter.println();
  }
  train.finishTraining();
  // Clear iteration CSV file
} catch (Exception e) {
  e.printStackTrace();
} finally {
  try {
    iterationFilePrinter.flush();
    iterationFilePrinter.close();
    iterationFileWriter.close();
```

```
    } catch (IOException e) { }
  }

  // Prepare results
  double trainingError = network.calculateError(mlTrainingSet);
  double validationError = network.calculateError(mlValidationSet);
  TreeMap<String, Double> returnMap = new TreeMap<String, Double>();
  returnMap.put("TrainingError", trainingError);
  returnMap.put("ValidationError", validationError);

  // Serialize network object for later use and write results csv file
  System.out.println("  Serialize network object");
  serializeMethod(simulatorParameters, individualParameters, network);
  try {
    writeResultToCsv(simulatorParameters, individualParameters, returnMap);
  } catch (IOException e) {
    e.printStackTrace();
  }

  return returnMap;
}

/**
 * Serialize an Encog method to a file in the output path.
 *
 * @param simulatorParameters
 *          the global parameters of the simulator
 * @param individualParameters
 *          the parameters of the individual as created by FADSE
 * @param method
 *          the Encog method
 */
private void serializeMethod(FemsSimulatorParameters simulatorParameters,
    FemsIndividualParameters individualParameters, MLRegression method) {
  String filename =
      String.format("%s_H%d_LAG%d.method", individualParameters.getField(),
          individualParameters.getNoOfHiddenNeurons(),
          individualParameters.getLagWindowSize());
  EncogDirectoryPersistence.saveObject(new File(simulatorParameters.getOutputPath(),
      filename), method);
}

/**
 * Write the results of the simulation to a CSV file.
 *
 * @param simulatorParameters
 *          the global parameters of the simulator
 * @param individualParameters
 *          the parameters of the individual as created by FADSE
 * @param resultMap
 *          the results
 * @throws IOException
 *           if writing is not possible
 */
private void writeResultToCsv(FemsSimulatorParameters simulatorParameters,
    FemsIndividualParameters individualParameters, TreeMap<String, Double> resultMap)
    throws IOException {
  File csvfile = new File(simulatorParameters.getOutputPath(), RESULTS_FILENAME);
  boolean appendToFile = false;
  if (csvfile.exists()) {
    appendToFile = true;
  }
  FileWriter csvFileWriter = null;
  CSVPrinter csvFilePrinter = null;
  try {
    csvFileWriter = new FileWriter(csvfile, appendToFile);
    csvFilePrinter = new CSVPrinter(csvFileWriter, FemsConstants.CSV_FORMAT);
    if (!appendToFile) {
      for (String header : FemsSimulatorParameters.getCsvHeaders()) {
        csvFilePrinter.print(header);
```

```
      }
      for (String header : FemsIndividualParameters.getCsvHeaders()) {
        csvFilePrinter.print(header);
      }
      for (String header : resultMap.keySet()) {
        csvFilePrinter.print(header);
      }
      csvFilePrinter.println();
    }
    for (String value : simulatorParameters.getCsvLine()) {
      csvFilePrinter.print(value);
    }
    for (String value : individualParameters.getCsvLine()) {
      csvFilePrinter.print(value);
    }
    for (Double value : resultMap.values()) {
      csvFilePrinter.print(String.format("%.8f", value));
    }
    csvFilePrinter.println();
  } finally {
    csvFilePrinter.flush();
    csvFilePrinter.close();
    csvFileWriter.close();
  }
}

/**
 * Prepare an example dataset for usage with Encog.
 *
 * @param fieldsPerTimestamp
 *          the example dataset
 * @param field
 *          the target field identifier
 * @param leadWindowSize
 *          the lead window size
 * @param lagWindowSize
 *          the lag window size
 * @return an MLDataSet suitable for Encog
 */
private MLDataSet getMLDataSet(
    TreeMap<Long, HashMap<String, Double>> fieldsPerTimestamp, String field,
    int leadWindowSize, int lagWindowSize) {
  MLDataSet mlDataSet = new BasicMLDataSet();

  for (Map.Entry<Long, HashMap<String, Double>> fieldPerTimestamp : fieldsPerTimestamp
      .entrySet()) {
    // timestamp of first ideal value
    Long timestamp = fieldPerTimestamp.getKey();
    boolean windowComplete = true;

    // Set ideal data
    MLData mlIdealData = new BasicMLData(leadWindowSize);
    {
      Map.Entry<Long, HashMap<String, Double>> previousHigherEntry =
          fieldPerTimestamp;
      for (int i = 0; i < leadWindowSize; i++) {
        // add the next leadWindowSize-number of values
        if (previousHigherEntry == null) {
          windowComplete = false;
          break;
        }
        mlIdealData.setData(i, previousHigherEntry.getValue().get(field));
        previousHigherEntry =
            fieldsPerTimestamp.higherEntry(previousHigherEntry.getKey());
      }
    }

    // Set input data
    MLData mlInputData = new BasicMLData(lagWindowSize);
    {
```

```
        Map.Entry<Long, HashMap<String, Double>> previousLowerEntry =
            fieldsPerTimestamp.lowerEntry(timestamp);
        for (int i = lagWindowSize - 1; i > -1; i--) {
          // add the strictly previous lagWindowSize-number of values
          if (previousLowerEntry == null) {
            windowComplete = false;
            break;
          }
          double value = previousLowerEntry.getValue().get(field);
          mlInputData.setData(i, value);
          previousLowerEntry =
              fieldsPerTimestamp.lowerEntry(previousLowerEntry.getKey());
        }
      }

      // Set data pair
      if (windowComplete) {
        BasicMLDataPair mlDataPair = new BasicMLDataPair(mlInputData, mlIdealData);
        mlDataSet.add(mlDataPair);
      }
    }
    return mlDataSet;
  }
}
```

B.7 fems.Normalizer.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 */
package ro.ulbsibiu.fadse.extended.problems.simulators.fems;

/**
 * Helper object to carry out pre- and postprocessing of data as required by a machine
 * learning method normalization.
 *
 * @author Stefan Feilmeier
 */
public class Normalizer {

  private final double dataHigh;
  private final double dataLow;
  private final double normalizedHigh;
  private final double normalizedLow;

  public double getDataHigh() {
    return dataHigh;
  }

  public double getDataLow() {
    return dataLow;
  }

  public double getNormalizedHigh() {
    return normalizedHigh;
  }

  public double getNormalizedLow() {
    return normalizedLow;
  }

  /**
   * Constructs the normalization utility for a given normalization range
   *
   * @param dataHigh
   *          the high value for the input data.
   * @param dataLow
   *          the low value for the input data.
   * @param dataHigh
   *          the high value for the normalized data.
   * @param dataLow
   *          the low value for the normalized data.
   */
  public Normalizer(double dataLow, double dataHigh, double normalizedLow,
      double normalizedHigh) {
    this.dataHigh = dataHigh;
    this.dataLow = dataLow;
    this.normalizedHigh = normalizedHigh;
    this.normalizedLow = normalizedLow;
  }


  /**
   * Normalizes a value
   *
   * @param value
   *          the value to normalize
   * @return the normalized value
   */
  public double normalize(double value) {
    if (value > dataHigh)
      return normalizedHigh;
    if (value < dataLow)
      return normalizedLow;
```

```
    return ((value - dataLow) / (dataHigh - dataLow))
        * (normalizedHigh - normalizedLow) + normalizedLow;
  }

  @Override
  public String toString() {
    return "Normalizer [dataHigh=" + dataHigh + ", dataLow=" + dataLow
        + ", normalizedHigh=" + normalizedHigh + ", normalizedLow=" + normalizedLow
        + "]";
  }

  /**
   * Denormalize a value
   *
   * @param value
   *          the normalized value
   * @return the real value.
   */
  public double denormalize(double value) {
    if (value < normalizedLow)
      return dataLow;
    if (value > normalizedHigh)
      return dataHigh;
    return ((dataLow - dataHigh) * value - normalizedHigh * dataLow + dataHigh
        * normalizedLow)
        / (normalizedLow - normalizedHigh);
  }
}
```

C. Source code: implementation of the proposed architecture

C.1 de.fenecon.fems

C.1.1 FemsApp.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems;

import de.fenecon.fems.agent.consumption.ConsumptionAgent;
import de.fenecon.fems.agent.consumption.ConsumptionAgentFactory;
import de.fenecon.fems.agent.load.HeatingDeviceLoadAgent;
import de.fenecon.fems.agent.load.LoadAgent;
import de.fenecon.fems.agent.scheduler.SchedulerAgent;
import de.fenecon.fems.agent.source.SourceAgent;
import de.fenecon.fems.agent.source.grid.GridAgent;
import de.fenecon.fems.agent.source.grid.GridAgentFactory;
import de.fenecon.fems.agent.source.pv.PvAgent;
import de.fenecon.fems.agent.source.pv.PvAgentFactory;
import de.fenecon.fems.ess.prohybrid.ProHybridSimulator;
import de.fenecon.fems.ess.prohybrid.ProHybridSimulatorFactory;

/**
 * Creates a complete environment for load scheduling.
 *
 * The {@link FemsApp} instantiates all {@link SourceAgent}s, {@link ConsumptionAgent}s
 * and {@link LoadAgent}s and starts a complete simulation.
 *
 * @author Stefan Feilmeier
 *
 */
public class FemsApp {

  public static void main(String[] args) throws Exception {
    // Create Source Agents
    PvAgent pv1Agent = PvAgentFactory.create(FemsConstants.PV1);
    PvAgent pv2Agent = PvAgentFactory.create(FemsConstants.PV2);
    GridAgent gridPh1Agent = GridAgentFactory.create(FemsConstants.GRID_PHASE1);
    GridAgent gridPh2Agent = GridAgentFactory.create(FemsConstants.GRID_PHASE2);
    GridAgent gridPh3Agent = GridAgentFactory.create(FemsConstants.GRID_PHASE3);

    // Create Consumption Agents
    ConsumptionAgent consumptionPh1Agent =
        ConsumptionAgentFactory.create(FemsConstants.CONSUMPTION_PHASE1);
    ConsumptionAgent consumptionPh2Agent =
        ConsumptionAgentFactory.create(FemsConstants.CONSUMPTION_PHASE2);
    ConsumptionAgent consumptionPh3Agent =
        ConsumptionAgentFactory.create(FemsConstants.CONSUMPTION_PHASE3);

    // Create Load Agents
    LoadAgent heatingDeviceLoadAgent = new HeatingDeviceLoadAgent();

    // Create Scheduler Agent
    SchedulerAgent scheduler = new SchedulerAgent();
    scheduler.addSourceAgent(pv1Agent);
    scheduler.addSourceAgent(pv2Agent);
    scheduler.addSourceAgent(gridPh1Agent);
    scheduler.addSourceAgent(gridPh2Agent);
    scheduler.addSourceAgent(gridPh3Agent);
    scheduler.addConsumptionAgent(consumptionPh1Agent);
    scheduler.addConsumptionAgent(consumptionPh2Agent);
```

```
    scheduler.addConsumptionAgent(consumptionPh3Agent);
    scheduler.addLoadAgent(heatingDeviceLoadAgent);

    // Initialize Simulator
    ProHybridSimulator proHybridSim = ProHybridSimulatorFactory.create("fems20");
    proHybridSim.addListener(pv1Agent);
    proHybridSim.addListener(pv2Agent);
    proHybridSim.addListener(consumptionPh1Agent);
    proHybridSim.addListener(consumptionPh2Agent);
    proHybridSim.addListener(consumptionPh3Agent);
  }
}
```

C.1.2 FemsConstants.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems;

import java.util.TimeZone;

import org.apache.commons.csv.CSVFormat;
import org.joda.time.DateTimeZone;

import de.fenecon.fems.agent.consumption.ConsumptionField;
import de.fenecon.fems.agent.source.grid.GridField;
import de.fenecon.fems.agent.source.pv.PvField;
import de.fenecon.fems.ess.prohybrid.ProHybridSimulator;

/**
 * Holds important Constants that are used throughout the application.
 *
 * @author Stefan Feilmeier
 */
public class FemsConstants {
  /** The local timezone of the device. */
  public final static DateTimeZone LOCAL_TIMEZONE = DateTimeZone.forTimeZone(TimeZone
      .getTimeZone("Europe/Berlin"));

  /** The length of one data slice (= prediction timespan). */
  public final static int SLICE_SECONDS = 5 * 60;

  /**
   * The maximum number of predictions. (This value multiplied with the SLICE_SECONDS is
   * the total prediction timeframe in seconds.)
   */
  public static final int MAX_PREDICTION_WINDOW = 36;

  /**
   * The simulated polling time for data from the storage system in seconds. In a real
   * application this is equal to SLICE_SECONDS
   */
  public static final int POLLING_TIME_MILLISECONDS = 200;

  /** The format for all CSV-files */
  public final static CSVFormat CSV_FORMAT = CSVFormat.DEFAULT;

  /** The name of the FEMS device to use */
  public final static String FEMS_NAME = "fems20";

  /** The URL of the InfluxDB api endpoint */
  public final static String INFLUXDB_URL = "http://fenecon.de:8086";

  /** The base path for external files */
  public final static String FILESPATH = "D:/fems/files";

  /** The photovoltaic installation on first MPP tracker */
  public final static PvField PV1 = new PvField("PV1", "PV1_Charger1_Output_Power");
  /** The photovoltaic installation on second MPP tracker */
  public final static PvField PV2 = new PvField("PV2", "PV2_Charger2_Output_Power");

  /** The power grid connection on first phase */
  public final static GridField GRID_PHASE1 = new GridField("GRID_PHASE1",
      "PCS1_Grid_Phase1_Active_Power");
  /** The power grid connection on second phase */
  public final static GridField GRID_PHASE2 = new GridField("GRID_PHASE2",
      "PCS2_Grid_Phase2_Active_Power");
```

```
  /** The power grid connection on third phase */
  public final static GridField GRID_PHASE3 = new GridField("GRID_PHASE3",
      "PCS3_Grid_Phase3_Active_Power");

  /** The power consumption on first phase */
  public final static ConsumptionField CONSUMPTION_PHASE1 = new ConsumptionField("Ph1",
      "PCS1_Phase1_Load_Active_Power");
  /** The power consumption on second phase */
  public final static ConsumptionField CONSUMPTION_PHASE2 = new ConsumptionField("Ph2",
      "PCS2_Phase2_Load_Active_Power");
  /** The power consumption on third phase */
  public final static ConsumptionField CONSUMPTION_PHASE3 = new ConsumptionField("Ph3",
      "PCS3_Phase3_Load_Active_Power");

  /**
   * The current timestamp: required for simulation and set by
   * {@link ProHybridSimulator}
   */
  public static volatile long CURRENT_TIMESTAMP = 0;
}
```

C.1.3 FemsTools.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems;

import java.nio.file.Path;
import java.nio.file.Paths;

import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;

/**
 * Hold helper tools that are used throughout the application.
 *
 * @author Stefan Feilmeier
 */
public class FemsTools {
  /**
   * Gets the path of the CSV file for a specific FEMS.
   *
   * @param fems
   *          the name of the fems (e.g. "fems1")
   * @return the path to the CSV file
   */
  public static Path getCsvPath(String fems) {
    return Paths.get(FemsConstants.FILESPATH, fems + ".csv");
  }

  /**
   * Returns the current timestamp, rounded down to a time-slice.
   *
   * @return the rounded current timestamp
   */
  public static long getCurrentRoundedUtcTimestamp() {
    /* DateTime nowInUtc = new
     * DateTime(FemsConstants.LOCAL_TIMEZONE).toDateTime(DateTimeZone.UTC); return
     * roundTimestampToSlice((nowInUtc.getMillis() / 1000));
     */
    return FemsConstants.CURRENT_TIMESTAMP; // only for Simulation
  }

  /**
   * Rounds a given timstamp down to the next time-slice.
   *
   * @param timestamp
   *          the timestamp to round
   * @return the rounded timestamp
   */
  public static long roundTimestampToSlice(long timestamp) {
    return (timestamp / (FemsConstants.SLICE_SECONDS)) * (FemsConstants.SLICE_SECONDS);
  }

  /**
   * Pretty prints a timestamp in human readable form in the local timezone.
   *
   * @param timestamp
   *          the timestamp to format
   * @return the formatted timestamp
   */
  public static String timestampToString(long timestamp) {
    DateTime date = timestampToDateTime(timestamp);
    return date.toString("dd.MM.y HH:mm");
  }
```

```
  /**
   * Converts a timestamp to a DateTime object in the local timezone.
   *
   * @param timestamp
   *          the timestamp
   * @return the DateTime object
   */
  public static DateTime timestampToDateTime(long timestamp) {
    DateTime date =
        new DateTime(timestamp * 1000, DateTimeZone.UTC)
            .toDateTime(FemsConstants.LOCAL_TIMEZONE);
    return date;
  }
}
```

C.2 de.fenecon.fems.agent.consumption

C.2.1 ConsumptionAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.consumption;

import de.fenecon.fems.ess.EssListener;
import de.fenecon.fems.helper.Field;
import de.fenecon.fems.helper.PredictionAgent;

/**
 * General agent for a consumption.
 *
 * @author Stefan Feilmeier
 */
public interface ConsumptionAgent extends PredictionAgent, EssListener {

  /**
   * Gets the {@link Field} of this consumption.
   *
   * @return the Field of this consumption
   */
  public ConsumptionField getField();
}
```

C.2.2 ConsumptionAgentFactory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.consumption;

import de.fenecon.fems.helper.PredictionAgentFactory;

/**
 * Factory for {@link ConsumptionAgent}s.
 *
 * @author Stefan Feilmeier
 */
public class ConsumptionAgentFactory extends PredictionAgentFactory {

  /**
   * Creates a new, valid {@link ConsumptionAgent}.
   *
   * @param field
   *          the field identifier of a consumption
   * @return the new {@link ConsumptionAgent}
   * @throws Exception
   *           if no valid {@link ConsumptionAgent} could be created.
   */
  public static ConsumptionAgent create(ConsumptionField field) throws Exception {
    return new ConsumptionAgentImpl(field, getPredictor(field));
  }
}
```

C.2.3 ConsumptionAgentImpl.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.consumption;

import de.fenecon.fems.helper.PredictionAgentImpl;
import de.fenecon.fems.helper.Predictor;

/**
 * Implementation of a {@link ConsumptionAgent}.
 *
 * @author Stefan Feilmeier
 */
public class ConsumptionAgentImpl extends PredictionAgentImpl implements
    ConsumptionAgent {

  /** The field identifier of this consumption. */
  protected final ConsumptionField field;

  /**
   * Creates a new instance. Preferable use with {@link ConsumptionAgentFactory}.
   *
   * @param field
   *          the field identifier of this consumption
   * @param predictors
   *          the predictor used by this agent
   */
  public ConsumptionAgentImpl(ConsumptionField field, Predictor predictor) {
    super(predictor);
    this.field = field;
  }

  /**
   * Gets the field identifier.
   */
  @Override
  public ConsumptionField getField() {
    return field;
  }
}
```

C.2.4 ConsumptionField.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.consumption;

import de.fenecon.fems.helper.Field;

/**
 * General definition of a field identifier for consumptions.
 *
 * @author Stefan Feilmeier
 */
public class ConsumptionField extends Field {

  /**
   * Creates a specific consumption identifier
   *
   * @param name
   *          the short name of this field
   * @param technicalName
   *          the technical name of this field
   */
  public ConsumptionField(String name, String technicalName) {
    super(name, technicalName);
  }
}
```

C.3 de.fenecon.fems.agent.load

C.3.1 HeatingDeviceLoadAgent

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.load;

import java.util.LinkedList;
import java.util.List;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.agent.consumption.ConsumptionField;
import de.fenecon.fems.agent.source.SourceCategory;

/**
 * Defines a {@link LoadAgent} for a simple heating device, generally used as a simple
 * way to heat water in a central water tank.
 *
 * @author Stefan Feilmeier
 */
public class HeatingDeviceLoadAgent extends LoadAgent {

  /**
   * Always requires at least PV priority.
   */
  @Override
  public float getAddedValue(long timestamp) {
    return SourceCategory.PHOTOVOLTAICS.ordinal();
  }

  /**
   * This device is connected to phase 1.
   */
  @Override
  public List<ConsumptionField> getConsumptionFields() {
    LinkedList<ConsumptionField> fields = new LinkedList<ConsumptionField>();
    fields.add(FemsConstants.CONSUMPTION_PHASE1);
    return fields;
  }

  /**
   * Starts when at least 500 W are available.
   */
  @Override
  public double getRequiredPower(long timestamp) {
    return 500;
  }
}
```

C.3.2 LoadAction.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.load;

import de.fenecon.fems.agent.scheduler.SchedulerAgent;

/**
 * Defines the actions that can be defined for a {@link LoadAgent} by the
 * {@link SchedulerAgent}.
 *
 * @author Stefan Feilmeier
 */
public enum LoadAction {
  START, STOP
}
```

C.3.3 LoadAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.load;

import java.util.List;

import de.fenecon.fems.agent.consumption.ConsumptionField;
import de.fenecon.fems.agent.source.SourceCategory;

/**
 * General agent for a load.
 *
 * @author Stefan Feilmeier
 */
public abstract class LoadAgent implements Comparable<LoadAgent> {
  /**
   * Each {@link LoadAgent} has an internal fallback priority in case of equal Added
   * Values. It is defined on creation of a new object.
   */
  private static Integer nextPriority = 0;
  /** Internal fallback priority of this agent. */
  private final int priority;

  /**
   * Creates a new {@link LoadAgent} with a given fallback priority.
   */
  public LoadAgent() {
    synchronized (nextPriority) {
      this.priority = nextPriority;
      nextPriority++;
    }
  }

  /**
   * Compares this {@link LoadAgent}s fallback priority to another one's.
   */
  @Override
  public int compareTo(LoadAgent o) {
    return priority - o.priority;
  }

  /**
   * Defines the added value of this load with regard to a {@link SourceCategory}
   * .ordinal().
   *
   * @param timestamp
   *          the timestamp to query
   * @return the added value, comparable with {@link SourceCategory}
   */
  public abstract float getAddedValue(long timestamp);

  /**
   * Gets a list of {@link ConsumptionField}s, where this load is attached to.
   *
   * @return the list of {@link ConsumptionField}s
   */
  public abstract List<ConsumptionField> getConsumptionFields();
```

```
  /**
   * Gets the fallback priority.
   *
   * @return the fallback priority
   */
  public int getPriority() {
    return priority;
  }

  /**
   * Gets the required power of this load.
   *
   * @param timestamp
   *          the timestamp to query
   * @return the required power in Watt
   */
  public abstract double getRequiredPower(long timestamp);

  /**
   * Returns the SimpleName of this class.
   */
  @Override
  public String toString() {
    return this.getClass().getSimpleName();
  }
}
```

C.4 de.fenecon.fems.agent.scheduler

C.4.1 SchedulerAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.scheduler;

import java.io.FileWriter;
import java.io.IOException;
import java.nio.file.Paths;
import java.util.Map;
import java.util.Map.Entry;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentSkipListMap;
import java.util.concurrent.ConcurrentSkipListSet;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.TimeUnit;

import org.apache.commons.csv.CSVPrinter;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.FemsTools;
import de.fenecon.fems.agent.consumption.ConsumptionAgent;
import de.fenecon.fems.agent.consumption.ConsumptionField;
import de.fenecon.fems.agent.load.LoadAction;
import de.fenecon.fems.agent.load.LoadAgent;
import de.fenecon.fems.agent.source.SourceAgent;
import de.fenecon.fems.agent.source.SourceCategory;
import de.fenecon.fems.helper.Field;
import de.fenecon.fems.helper.Prediction;
import de.fenecon.fems.helper.PredictionAgent;

/**
 * Creates a schedule for {@link LoadAgent}s.
 *
 * The {@link SchedulerAgent} collects the predictions from all registered
 * {@link SourceAgent}s and {@link ConsumptionAgent}s, creates a plan with the predicted
 * available power per timestamp and uses this informaton to schedule the registered
 * {@link LoadAgent}s.
 *
 * @author Stefan Feilmeier
 */
public class SchedulerAgent {
  /** The list of registered {@link ConsumptionAgent}s */
  private final ConcurrentSkipListSet<ConsumptionAgent> consumptionAgents =
      new ConcurrentSkipListSet<ConsumptionAgent>();
  /** The list of registered {@link LoadAgent}s */
  private final ConcurrentSkipListSet<LoadAgent> loadAgents =
      new ConcurrentSkipListSet<LoadAgent>();
  /** The list of registered {@link SourceAgent}s per {@link SourceCategory} */
  private final ConcurrentSkipListMap<SourceCategory,
      ConcurrentSkipListSet<PredictionAgent>> sourceAgents =
      new ConcurrentSkipListMap<SourceCategory, ConcurrentSkipListSet<PredictionAgent>>();
  /** The Executor service for the actual Scheduler Agent worker */
  private final ScheduledExecutorService scheduler = Executors
      .newScheduledThreadPool(1);
```

```
/**
 * Worker runnable for SchedulerAgent
 */
private Runnable worker = new Runnable() {
  @Override
  public void run() {
    try {
      // Set the current timestamp for simulation purposes
      long currentTimestamp = FemsTools.getCurrentRoundedUtcTimestamp();

      // Get the sourcePredictions from registered SourceAgents
      TreeMap<Long, TreeMap<SourceCategory, Prediction>> sourcePredictions =
          getSourcePredictions(currentTimestamp);
      // printSourcePredictions(sourcePredictions, currentTimestamp);

      // Get the consumptionPredictions from registered
      // ConsumptionAgents
      TreeMap<Long, TreeMap<ConsumptionField, Prediction>> consumptionPredictions =
          getConsumptionPredictions(currentTimestamp);
      // printConsumptionPredictions(consumptionPredictions, currentTimestamp);

      // Copy data to a new Map for debug purposes
      refreshFieldsPerTimestamp(currentTimestamp);

      // Create a plan with the predicted available power per
      // timestamp
      TreeMap<Long, TreeMap<SourceCategory, Double[]>> powerPredictions =
          getPowerPredictions(sourcePredictions, consumptionPredictions,
              currentTimestamp);
      // printPowerPredictions(powerPredictions, currentTimestamp);

      // Create the load schedule
      TreeMap<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>> schedule =
          createSchedule(powerPredictions, currentTimestamp);
      // printSchedule(schedule, currentTimestamp);

    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Creates the schedule for {@link LoadAgent}s.
   *
   * @param powerPredictions
   *          the powerPredictions (Use {@link SchedulerAgent#getPowerPredictions}
   * @param currentTimestamp
   *          the current timestamp
   * @return the schedule per timestamp and {@link LoadAgent}
   */
  private TreeMap<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>> createSchedule(
      TreeMap<Long, TreeMap<SourceCategory, Double[]>> powerPredictions,
      long currentTimestamp) {
    TreeMap<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>> schedule =
        new TreeMap<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>>();
    for (long timestamp = currentTimestamp; timestamp <= currentTimestamp
        + FemsConstants.SLICE_SECONDS * FemsConstants.MAX_PREDICTION_WINDOW; timestamp +=
        FemsConstants.SLICE_SECONDS) {
      schedule.putIfAbsent(timestamp,
          new ConcurrentSkipListMap<LoadAgent, LoadAction>());
      ConcurrentSkipListMap<LoadAgent, LoadAction> schedulePerTimestamp =
          schedule.get(timestamp);
      TreeMap<SourceCategory, Double[]> powerPerTimestamp =
          powerPredictions.get(timestamp);
      LoadAction loadAction = LoadAction.STOP;

      for (LoadAgent loadAgent : loadAgents) {
        double remainingPower = 0.;
        float addedValue = loadAgent.getAddedValue(timestamp);
```

```
      for (Map.Entry<SourceCategory, Double[]> powerPerCategory : powerPerTimestamp
          .entrySet()) {
        if (addedValue >= powerPerCategory.getKey().ordinal()) {
          remainingPower = powerPerCategory.getValue()[1];
        } else {
          break;
        }
      }
      double requiredPower = loadAgent.getRequiredPower(timestamp);
      if (remainingPower > requiredPower) {
        // very basic scheduling!
        loadAction = LoadAction.START;
        remainingPower -= requiredPower;
      }
      schedulePerTimestamp.put(loadAgent, loadAction);
    }

  }

  return schedule;
}

/**
 * Poll all registered {@link ConsumptionAgent1}s for their predictions and sum them
 * up per {@link ConsumptionField}.
 *
 * @param currentTimestamp
 *          the current timestamp
 * @return the list of consumptionPredictions
 */
private TreeMap<Long, TreeMap<ConsumptionField, Prediction>> getConsumptionPredictions(
    long currentTimestamp) {
  final TreeMap<Long, TreeMap<ConsumptionField, Prediction>> predictions =
      new TreeMap<Long, TreeMap<ConsumptionField, Prediction>>();
  for (long timestamp = currentTimestamp; timestamp <= currentTimestamp
      + FemsConstants.SLICE_SECONDS * FemsConstants.MAX_PREDICTION_WINDOW; timestamp +=
      FemsConstants.SLICE_SECONDS) {
    TreeMap<ConsumptionField, Prediction> predictionsPerTimestamp =
        new TreeMap<ConsumptionField, Prediction>();

    for (PredictionAgent agent : consumptionAgents) {
      Prediction prediction = agent.getBestPredictionAtTimestamp(timestamp); // Get
      Field field = agent.getField();
      if (prediction != null && field instanceof ConsumptionField) {
        predictionsPerTimestamp.put((ConsumptionField) field, prediction);
      }
    }

    predictions.put(timestamp, predictionsPerTimestamp);
  }

  return predictions;
}

/**
 * Creates a plan with the predicted available power per timestamp.
 *
 * @param sourcePredictions
 *          the list of sourcePredictions
 * @param consumptionPredictions
 *          the list of consumptionPredictions
 * @param currentTimestamp
 *          the current timestamp
 * @return the plan with predicted available power per timestamp and
 *         {@link SourceCategory}
 */
private TreeMap<Long, TreeMap<SourceCategory, Double[]>> getPowerPredictions(
    TreeMap<Long, TreeMap<SourceCategory, Prediction>> sourcePredictions,
    TreeMap<Long, TreeMap<ConsumptionField, Prediction>> consumptionPredictions,
    long currentTimestamp) {
```

```
  TreeMap<Long, TreeMap<SourceCategory, Double[]>> powerPredictions =
      new TreeMap<Long, TreeMap<SourceCategory, Double[]>>();
  for (long timestamp = currentTimestamp; timestamp <= currentTimestamp
      + FemsConstants.SLICE_SECONDS * FemsConstants.MAX_PREDICTION_WINDOW; timestamp +=
      FemsConstants.SLICE_SECONDS) {
    double totalConsumptionPerTimestamp = 0.;
    TreeMap<ConsumptionField, Prediction> consumptionPredictionsPerTimestamp =
        consumptionPredictions.get(timestamp);

    for (Map.Entry<ConsumptionField, Prediction> consumptionPredictionPerCategory :
        consumptionPredictionsPerTimestamp.entrySet()) {
      totalConsumptionPerTimestamp +=
          consumptionPredictionPerCategory.getValue().getValue();
    }

    double remainingConsumptionPerTimestamp = totalConsumptionPerTimestamp;
    TreeMap<SourceCategory, Prediction> sourcePredictionsPerTimestamp =
        sourcePredictions.get(timestamp);
    TreeMap<SourceCategory, Double[]> powerPerTimestamp =
        new TreeMap<SourceCategory, Double[]>();

    for (Map.Entry<SourceCategory, Prediction> sourcePredictionPerCategory :
        sourcePredictionsPerTimestamp.entrySet()) {
      final Double[] powerPerCategory = new Double[2];
      // [0]: total power; [1]: available power
      powerPerCategory[0] = sourcePredictionPerCategory.getValue().getValue();
      if (remainingConsumptionPerTimestamp > powerPerCategory[0]) {
        powerPerCategory[1] = 0.;
        remainingConsumptionPerTimestamp -= powerPerCategory[0];
      } else if (remainingConsumptionPerTimestamp <= powerPerCategory[0]) {
        remainingConsumptionPerTimestamp = 0;
        powerPerCategory[1] =
            powerPerCategory[0] - remainingConsumptionPerTimestamp;
      }
      powerPerTimestamp.put(sourcePredictionPerCategory.getKey(), powerPerCategory);
    }

    powerPredictions.put(timestamp, powerPerTimestamp);
  }

  return powerPredictions;
}

/**
 * Poll all registered {@link SourceAgent}s for their predictions and sum them up
 * per {@link SourceCategory}.
 *
 * @param currentTimestamp
 *          the current timestamp
 * @return the list of sourcePredictions
 */
private TreeMap<Long, TreeMap<SourceCategory, Prediction>> getSourcePredictions(
    long currentTimestamp) {
  final TreeMap<Long, TreeMap<SourceCategory, Prediction>> predictions =
      new TreeMap<Long, TreeMap<SourceCategory, Prediction>>();
  for (long timestamp = currentTimestamp; timestamp <= currentTimestamp
      + FemsConstants.SLICE_SECONDS * FemsConstants.MAX_PREDICTION_WINDOW; timestamp +=
      FemsConstants.SLICE_SECONDS) {
    TreeMap<SourceCategory, Prediction> predictionsPerTimestamp =
        new TreeMap<SourceCategory, Prediction>();

    for (Map.Entry<SourceCategory, ConcurrentSkipListSet<PredictionAgent>>
        categoryAgents : sourceAgents.entrySet()) {
      float sumLeadWindowSize = 0f;
      double sumValue = 0.;
      for (PredictionAgent agent : categoryAgents.getValue()) {
        Prediction prediction = agent.getBestPredictionAtTimestamp(timestamp);
        if (prediction != null) {
          sumLeadWindowSize += prediction.getLeadWindowSize();
          sumValue += prediction.getValue();
```

```
        }
      }
      predictionsPerTimestamp.put(categoryAgents.getKey(), new Prediction(sumValue,
          sumLeadWindowSize / categoryAgents.getValue().size()));
    }

    predictions.put(timestamp, predictionsPerTimestamp);
  }

  return predictions;
}

/**
 * Pretty prints the consumptionPredictions to standard output.
 *
 * @param consumptionPredictions
 *          the consumptionPredictions
 * @param currentTimestamp
 *          the current timestamp
 */
private void printConsumptionPredictions(
    TreeMap<Long, TreeMap<ConsumptionField, Prediction>> consumptionPredictions,
    long currentTimestamp) {
  System.out.println("ConsumptionPredictions at "
      + FemsTools.timestampToString(currentTimestamp));
  for (Entry<Long, TreeMap<ConsumptionField, Prediction>> consumptionPerTimestamp :
      consumptionPredictions.entrySet()) {
    StringBuilder line = new StringBuilder();
    line.append("@");
    line.append(FemsTools.timestampToString(consumptionPerTimestamp.getKey()));
    line.append(": ");
    String split = "";
    for (Entry<ConsumptionField, Prediction> consumptionPerCategory :
        consumptionPerTimestamp.getValue().entrySet()) {
      line.append(split);
      line.append("{");
      line.append(consumptionPerCategory.getKey());
      line.append(": ");
      line.append(consumptionPerCategory.getValue());
      line.append("}");
      split = "; ";
    }
    System.out.println(line.toString());
  }
}

/**
 * Pretty prints the powerPredictions to standard output.
 *
 * @param powerPredictions
 *          the powerPredictions
 * @param currentTimestamp
 *          the current timestamp
 */
private void printPowerPredictions(
    TreeMap<Long, TreeMap<SourceCategory, Double[]>> powerPredictions,
    long currentTimestamp) {
  System.out.println("PowerPredictions at "
      + FemsTools.timestampToString(currentTimestamp));
  for (Entry<Long, TreeMap<SourceCategory, Double[]>> powerPerTimestamp :
      powerPredictions.entrySet()) {
    StringBuilder line = new StringBuilder();
    line.append("@");
    line.append(FemsTools.timestampToString(powerPerTimestamp.getKey()));
    line.append(": ");
    String split = "";
    for (Entry<SourceCategory, Double[]> powerPerCategory : powerPerTimestamp
        .getValue().entrySet()) {
      line.append(split);
      line.append("{");
```

```
      line.append(powerPerCategory.getKey());
      line.append(": ");
      line.append(String.format("Total: %.2f, ", powerPerCategory.getValue()[0]));
      line.append(String.format("Available: %.2f", powerPerCategory.getValue()[1]));
      line.append("}");
      split = "; ";
    }
    System.out.println(line.toString());
  }
}

/**
 * Pretty prints the schedule to standard output.
 *
 * @param schedule
 *          the schedule
 * @param currentTimestamp
 *          the current timestamp
 */
private void printSchedule(
    TreeMap<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>> schedule,
    long currentTimestamp) {
  System.out
      .println("Schedule at " + FemsTools.timestampToString(currentTimestamp));
  for (Map.Entry<Long, ConcurrentSkipListMap<LoadAgent, LoadAction>>
      schedulePerTimestamp : schedule.entrySet()) {
    StringBuilder line = new StringBuilder();
    line.append("@");
    line.append(FemsTools.timestampToString(schedulePerTimestamp.getKey()));
    line.append(": ");
    String split = "";
    for (Entry<LoadAgent, LoadAction> schedulePerLoadAgent : schedulePerTimestamp
        .getValue().entrySet()) {
      line.append(split);
      line.append("{");
      line.append(schedulePerLoadAgent.getKey());
      line.append(": ");
      line.append(schedulePerLoadAgent.getValue());
      line.append("}");
      split = "; ";
    }
    System.out.println(line.toString());
  }
}

/**
 * Pretty prints the sourcePredictions to standard output.
 *
 * @param sourcePredictions
 *          the sourcePredictions
 * @param currentTimestamp
 *          the currentTimestamp
 */
private void printSourcePredictions(
    TreeMap<Long, TreeMap<SourceCategory, Prediction>> sourcePredictions,
    long currentTimestamp) {
  System.out.println("SourcePredictions at "
      + FemsTools.timestampToString(currentTimestamp));
  for (Entry<Long, TreeMap<SourceCategory, Prediction>> sourcePerTimestamp :
      sourcePredictions.entrySet()) {
    StringBuilder line = new StringBuilder();
    line.append("@");
    line.append(FemsTools.timestampToString(sourcePerTimestamp.getKey()));
    line.append(": ");
    String split = "";
    for (Entry<SourceCategory, Prediction> sourcePerCategory : sourcePerTimestamp
        .getValue().entrySet()) {
      line.append(split);
      line.append("{");
      line.append(sourcePerCategory.getKey());
```

```
        line.append(": ");
        line.append(sourcePerCategory.getValue());
        line.append("}");
        split = "; ";
      }
      System.out.println(line.toString());
    }
  }
};

/**
 * Creates a SchedulerAgent and starts its worker runnable.
 */
public SchedulerAgent() {
  scheduler.scheduleAtFixedRate(worker, FemsConstants.POLLING_TIME_MILLISECONDS / 2,
      FemsConstants.POLLING_TIME_MILLISECONDS, TimeUnit.MILLISECONDS);

  scheduler.scheduleAtFixedRate(writeCsvFile,
      FemsConstants.POLLING_TIME_MILLISECONDS * 2,
      FemsConstants.POLLING_TIME_MILLISECONDS * 100, TimeUnit.MILLISECONDS);
}

/**
 * Adds a new {@link ConsumptionAgent}.
 *
 * @param agent
 *          the ConsumptionAgent to add
 */
public void addConsumptionAgent(ConsumptionAgent agent) {
  if (agent == null)
    return;
  consumptionAgents.add(agent);
}

/**
 * Adds a new {@link LoadAgent}.
 *
 * @param agent
 *          the LoadAgent to add
 */
public void addLoadAgent(LoadAgent agent) {
  loadAgents.add(agent);
}

/**
 * Adds a new {@link SourceAgent}.
 *
 * @param agent
 *          the SourceAgent to add
 */
public void addSourceAgent(SourceAgent agent) {
  if (agent == null)
    return;
  SourceCategory sourceCategory = agent.getSourceCategory();
  sourceAgents.putIfAbsent(sourceCategory,
      new ConcurrentSkipListSet<PredictionAgent>());
  ConcurrentSkipListSet<PredictionAgent> currentSourceAgents =
      sourceAgents.get(sourceCategory);
  currentSourceAgents.add(agent);
}

private ConcurrentSkipListMap<Long, ConcurrentSkipListMap<Field,
    ConcurrentSkipListMap<Integer, Double>>> fieldsPerTimestamp =
    new ConcurrentSkipListMap<Long, ConcurrentSkipListMap<Field,
    ConcurrentSkipListMap<Integer, Double>>>();
```

```
private void refreshFieldsPerTimestamp(long currentTimestamp) {
  for (long timestamp = currentTimestamp; timestamp <= currentTimestamp
      + FemsConstants.SLICE_SECONDS * FemsConstants.MAX_PREDICTION_WINDOW; timestamp +=
      FemsConstants.SLICE_SECONDS) {
    fieldsPerTimestamp.putIfAbsent(timestamp,
        new ConcurrentSkipListMap<Field, ConcurrentSkipListMap<Integer, Double>>());
    ConcurrentSkipListMap<Field, ConcurrentSkipListMap<Integer, Double>> leadsPerField =
        fieldsPerTimestamp.get(timestamp);

    for (ConcurrentSkipListSet<PredictionAgent> agents : sourceAgents.values()) {
      for (PredictionAgent agent : agents) {
        leadsPerField.putIfAbsent(agent.getField(),
            new ConcurrentSkipListMap<Integer, Double>());
        ConcurrentSkipListMap<Integer, Double> valuesPerLead =
            leadsPerField.get(agent.getField());

        ConcurrentSkipListSet<Prediction> predictions =
            agent.getPredictionsAtTimestamp(timestamp);
        for (Prediction prediction : predictions) {
          valuesPerLead.putIfAbsent((int) prediction.getLeadWindowSize(),
              prediction.getValue());
        }
      }
    }
    for (ConsumptionAgent agent : consumptionAgents) {
      leadsPerField.putIfAbsent(agent.getField(),
          new ConcurrentSkipListMap<Integer, Double>());
      ConcurrentSkipListMap<Integer, Double> valuesPerLead =
          leadsPerField.get(agent.getField());

      ConcurrentSkipListSet<Prediction> predictions =
          agent.getPredictionsAtTimestamp(timestamp);
      for (Prediction prediction : predictions) {
        valuesPerLead.putIfAbsent((int) prediction.getLeadWindowSize(),
            prediction.getValue());
      }
    }
  }
}

private Runnable writeCsvFile = new Runnable() {
  public void run() {
    try {
      // Source Agents
      for (Map.Entry<SourceCategory, ConcurrentSkipListSet<PredictionAgent>>
          agentPerCategory : sourceAgents.entrySet()) {
        for (PredictionAgent agent : agentPerCategory.getValue()) {
          Field field = agent.getField();
          printFieldCsvFile(field);
        }
      }
      // Consumption Agents
      for (PredictionAgent agent : consumptionAgents) {
        Field field = agent.getField();
        printFieldCsvFile(field);
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  private void printFieldCsvFile(Field field) throws IOException {
    String time = FemsTools.timestampToString(FemsConstants.CURRENT_TIMESTAMP);
    try (CSVPrinter csvFilePrinter =
        new CSVPrinter(new FileWriter(Paths
            .get(FemsConstants.FILESPATH, "scheduler",
                field + "_" + time.replace(":", ".") + ".csv").toAbsolutePath()
            .toString()), FemsConstants.CSV_FORMAT)) {
```

```
        // Print header
        csvFilePrinter.print("Timestamp");
        csvFilePrinter.print("Ideal");
        for (int lead = 1; lead <= FemsConstants.MAX_PREDICTION_WINDOW; lead++) {
          csvFilePrinter.print((FemsConstants.SLICE_SECONDS * lead / 60) + " Minutes");
        }
        csvFilePrinter.println();

        // Print lines
        for (Map.Entry<Long, ConcurrentSkipListMap<Field, ConcurrentSkipListMap<Integer,
            Double>>> fieldPerTimestamp : fieldsPerTimestamp.entrySet()) {
          csvFilePrinter.print(FemsTools.timestampToString(fieldPerTimestamp.getKey()));
          for (Map.Entry<Field, ConcurrentSkipListMap<Integer, Double>> leadsPerField :
              fieldPerTimestamp.getValue().entrySet()) {
            if (leadsPerField.getKey() != field)
              continue;
            ConcurrentSkipListMap<Integer, Double> valuesPerLead =
                leadsPerField.getValue();
            for (int lead = 0; lead <= FemsConstants.MAX_PREDICTION_WINDOW; lead++) {
              Double value = valuesPerLead.get(lead);
              if (value == null) {
                csvFilePrinter.print("");
              } else {
                csvFilePrinter.print(String.format("%.2f", value));
              }
            }
          }
          csvFilePrinter.println();
        }
      }
    }
  };
}
```

C.5 de.fenecon.fems.agent.source

C.5.1 SourceAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source;

import de.fenecon.fems.helper.PredictionAgent;

/**
 * General agent for a power source.
 *
 * @author Stefan Feilmeier
 */
public interface SourceAgent extends PredictionAgent {

	/**
	 * Gets the {@link SourceCategory} of this agent.
	 *
	 * @return the {@link SourceCategory}
	 */
	public SourceCategory getSourceCategory();

}
```

C.5.2 SourceCategory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source;

import de.fenecon.fems.agent.load.LoadAgent;
import de.fenecon.fems.agent.scheduler.SchedulerAgent;

/**
 * Defines a category of a electricity source. It is used as a group-by parameter for
 * source predictions in {@link SchedulerAgent} and as a measure of priority in the form
 * of "Added Value" of a {@link LoadAgent}.
 *
 * @author Stefan Feilmeier
 */
public enum SourceCategory {
  PHOTOVOLTAICS, ENERGY_STORAGE_SYSTEM, BLOCK_HEATING_DEVICE, POWER_GRID, DIESEL_GENERATOR
}
```

C.5.3 grid.GridAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.grid;

import java.util.concurrent.ConcurrentSkipListSet;

import de.fenecon.fems.agent.source.SourceAgent;
import de.fenecon.fems.agent.source.SourceCategory;
import de.fenecon.fems.helper.Field;
import de.fenecon.fems.helper.Prediction;
import de.fenecon.fems.helper.PredictionAgent;

/**
 * Defines the power grid as a source of electricity.
 *
 * @author Stefan Feilmeier
 */
public class GridAgent implements SourceAgent {
  /** the field descriptor defining the phase of this source */
  private final GridField field;

  /**
   * Creates a new {@link GridAgent} defined by its {@link GridField}.
   *
   * @param field
   *          the connected field
   */
  public GridAgent(GridField field) {
    this.field = field;
  }

  @Override
  public int compareTo(PredictionAgent o) {
    return this.field.compareTo(o.getField());
  }

  /**
   * Always returns an infinity value for prediction.
   *
   * @return the infinite prediction
   */
  @Override
  public Prediction getBestPredictionAtTimestamp(long timestamp) {
    return new Prediction(Double.MAX_VALUE, 0);
  }

  @Override
  public Field getField() {
    return field;
  }

  @Override
  public ConcurrentSkipListSet<Prediction> getPredictionsAtTimestamp(long timestamp) {
    ConcurrentSkipListSet<Prediction> predictions =
        new ConcurrentSkipListSet<Prediction>();
    predictions.add(getBestPredictionAtTimestamp(timestamp));
    return predictions;
  }
```

```
  /**
   * Gets the {@link SourceCategory} "POWER_GRID".
   *
   * @return the {@link SourceCategory}
   */
  @Override
  public SourceCategory getSourceCategory() {
    return SourceCategory.POWER_GRID;
  }

  @Override
  public String toString() {
    return field.getName();
  }
}
```

C.5.4 grid.GridAgentFactory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.grid;

/**
 * Factory for {@link GridAgent}s.
 *
 * @author Stefan Feilmeier
 */
public class GridAgentFactory {

  /**
   * Create a new {@link GridAgent} connected to a {@link GridField}.
   *
   * @param field
   *          the field of the grid phase
   * @return the new {@link GridAgent}
   */
  public static GridAgent create(GridField field) {
    return new GridAgent(field);
  }
}
```

C.5.5 grid.GridField.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.grid;

import de.fenecon.fems.helper.Field;

/**
 * General definition of a field identifier for power grid connection.
 *
 * @author Stefan Feilmeier
 */
public class GridField extends Field {

  /**
   * Creates a specific power grid identifier
   *
   * @param name
   *          the short name of this field
   * @param technicalName
   *          the technical name of this field
   */
  public GridField(String name, String technicalName) {
    super(name, technicalName);
  }

}
```

C.5.6 pv.PvAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.pv;

import de.fenecon.fems.agent.source.SourceAgent;
import de.fenecon.fems.agent.source.SourceCategory;
import de.fenecon.fems.helper.Field;
import de.fenecon.fems.helper.PredictionAgentImpl;
import de.fenecon.fems.helper.Predictor;

/**
 * Defines a {@link SourceAgent} for a photovoltaic installation.
 *
 * @author Stefan Feilmeier
 */
public class PvAgent extends PredictionAgentImpl implements SourceAgent {
  /** the field of this pv installation */
  private final PvField field;

  /**
   * Creates a new PvAgent. Use with {@link PvAgentFactory}.
   *
   * @param field
   *          the field of this pv installation
   * @param predictor
   *          the predictor for this agent
   */
  public PvAgent(PvField field, Predictor predictor) {
    super(predictor);
    this.field = field;
  }

  @Override
  public Field getField() {
    return field;
  }

  /**
   * Gets the {@link SourceCategory} "PHOTOVOLTAICS".
   *
   * @return the {@link SourceCategory}
   */
  @Override
  public SourceCategory getSourceCategory() {
    return SourceCategory.PHOTOVOLTAICS;
  }
}
```

C.5.7 pv.PvAgentFactory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.pv;

import de.fenecon.fems.helper.PredictionAgentFactory;

/**
 * Factory for {@link PvAgent}s.
 *
 * @author Stefan Feilmeier
 */
public class PvAgentFactory extends PredictionAgentFactory {
  /**
   * Creates a new, valid {@link PvAgent}
   *
   * @param field
   *          the field identifier of a photovoltaic installation
   * @return the new {@link PvAgent}
   * @throws Exception
   *           if no valid {@link PvAgent} could be created.
   */
  public static PvAgent create(PvField field) throws Exception {
    return new PvAgent(field, getPredictor(field));
  }
}
```

C.5.8 pv.PvField.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.agent.source.pv;

import de.fenecon.fems.helper.Field;

/**
 * General definition of a field identifier for a photovoltaic installation.
 *
 * @author Stefan Feilmeier
 */
public class PvField extends Field {

  /**
   * Creates a specific photovoltaic installation identifier
   *
   * @param name
   *          the short name of this field
   * @param technicalName
   *          the technical name of this field
   */
  public PvField(String name, String technicalName) {
    super(name, technicalName);
  }

}
```

C.6 de.fenecon.fems.ess

C.6.1 EssListener.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.ess;

import de.fenecon.fems.helper.Field;

/**
 * Defines a Listener for Energy Storage System (ESS) events.
 *
 * @author Stefan Feilmeier
 */
public abstract interface EssListener {
  /**
   * Gets the {@link Field} descriptor.
   *
   * @return the field descriptor
   */
  Field getField();

  /**
   * Notifies this listener of a new value. Implement this method to react on
   * notifications fom Energy Storage System (ESS).
   *
   * @param timestamp
   *          the timestamp of the value
   * @param value
   *          the value
   */
  public void newValue(long timestamp, double value);
}
```

C.6.2 ess.prohybrid.ProHybridSimulator.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.ess.prohybrid;

import java.util.HashMap;
import java.util.Map;
import java.util.NoSuchElementException;
import java.util.TreeMap;
import java.util.concurrent.ConcurrentSkipListSet;
import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import java.util.concurrent.TimeUnit;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.ess.EssListener;
import de.fenecon.fems.helper.Field;

/**
 * Defines a simulator for a FENECON by BYD PRO Hybrid Energy Storage System. It reads
 * the from an internal cache and sends a new event every few seconds, as defined in
 * {@link FemsConstants}.
 *
 * @author Stefan Feilmeier
 */
public class ProHybridSimulator {
  /** the cached CSV file */
  private final TreeMap<Long, HashMap<Field, Double>> cacheMap;
  /** the list of {@link EssListener}s */
  private final ConcurrentSkipListSet<EssListener> listeners =
      new ConcurrentSkipListSet<EssListener>();

  private final ScheduledFuture<?> scheduledFuture;
  private final ScheduledExecutorService scheduler = Executors
      .newScheduledThreadPool(1);

  /**
   * Runnable that sends the next event.
   */
  private Runnable simulator = new Runnable() {
    @Override
    public void run() {
      try {
        Map.Entry<Long, HashMap<Field, Double>> entry = cacheMap.pollFirstEntry();
        long timestamp = entry.getKey();
        // Simulation Timestamp
        FemsConstants.CURRENT_TIMESTAMP = timestamp;
        for (EssListener listener : listeners) {
          Double value = entry.getValue().get(listener.getField());
          if (value != null) {
            listener.newValue(timestamp, value);
          }
        }
      } catch (NoSuchElementException e) {
        System.out.println("HistoryCache is empty: " + e.getMessage());
        scheduledFuture.cancel(false);
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  };
```

```
  /**
   * Creates a new Simulator with a defined cache. Use with
   * {@link ProHybridSimulatorFactory}.
   *
   * @param cacheMap
   */
  public ProHybridSimulator(TreeMap<Long, HashMap<Field, Double>> cacheMap) {
    this.cacheMap = cacheMap;
    scheduledFuture =
        scheduler.scheduleAtFixedRate(simulator, 0,
            FemsConstants.POLLING_TIME_MILLISECONDS, TimeUnit.MILLISECONDS);
  }

  /**
   * Add a new listener.
   *
   * @param listener
   *          the listener object
   */
  public void addListener(EssListener listener) {
    listeners.add(listener);
  }
}
```

C.6.3 ess.prohybrid.ProHybridSimulatorFactory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.ess.prohybrid;

import java.io.FileReader;
import java.io.IOException;
import java.io.Reader;
import java.util.HashMap;
import java.util.TreeMap;

import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVRecord;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.FemsTools;
import de.fenecon.fems.helper.Field;

/**
 * Factory for a {@link ProHybridSimulator}.
 *
 * @author Stefan Feilmeier
 */
public class ProHybridSimulatorFactory {

  /**
   * Creates a new {@link ProHybridSimulator} with data from a CSV file.
   *
   * @return the new {@link ProHybridSimulator}
   * @throws IOException
   *           if no valid {@link ProHybridSimulator} could be created from the CSV file
   */
  public static ProHybridSimulator create(String fems) throws IOException {
    Field[] fields =
        new Field[] { FemsConstants.PV1, FemsConstants.PV2,
            FemsConstants.CONSUMPTION_PHASE1, FemsConstants.CONSUMPTION_PHASE2,
            FemsConstants.CONSUMPTION_PHASE3 };
    final TreeMap<Long, HashMap<Field, Double>> cacheMap =
        new TreeMap<Long, HashMap<Field, Double>>();

    // read CSV file
    final Reader in = new FileReader(FemsTools.getCsvPath(fems).toFile());
    final Iterable<CSVRecord> records = CSVFormat.DEFAULT.withHeader().parse(in);
    for (CSVRecord record : records) {
      HashMap<Field, Double> values = new HashMap<Field, Double>();
      for (Field field : fields) {
        values.put(field, Double.parseDouble(record.get(field.getName())));
      }
      cacheMap.put(Long.parseLong(record.get("timestamp")), values);
    }
    return new ProHybridSimulator(cacheMap);
  }
}
```

C.7 de.fenecon.fems.helper

C.7.1 DownloadDataApp.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.SortedMap;
import java.util.TreeMap;
import java.util.concurrent.TimeUnit;

import org.apache.commons.csv.CSVPrinter;
import org.influxdb.InfluxDB;
import org.influxdb.InfluxDBFactory;
import org.influxdb.dto.Serie;
import org.joda.time.DateTime;
import org.joda.time.DateTimeZone;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.FemsTools;
import de.fenecon.fems.agent.source.pv.PvField;

/**
 * Helper App to generate a CSV file from the FENECON Online-Monitoring InfluxDB
 * database.
 *
 * @author Stefan Feilmeier
 */
public class DownloadDataApp {
  /** parameterized, SQL-like query string */
  private final static String QUERY =
      "SELECT MEAN(value) FROM %s GROUP BY time(%ds) fill(null) "
      + "WHERE time > %ds AND time < %ds";

  /**
   * Defines from- and to-date, and the fields that should be received from the InfluxDB
   * database and starts the download.
   *
   * @param args
   * @throws IOException
   */
  public static void main(String[] args) throws IOException {
    DownloadDataApp d = new DownloadDataApp();
    DateTime fromDate = new DateTime(2015, 1, 1, 0, 0, FemsConstants.LOCAL_TIMEZONE);
    DateTime toDate = new DateTime(2015, 6, 1, 0, 0, FemsConstants.LOCAL_TIMEZONE);
    Field[] fields =
        new Field[] { FemsConstants.PV1, FemsConstants.PV2,
            FemsConstants.CONSUMPTION_PHASE1, FemsConstants.CONSUMPTION_PHASE2,
            FemsConstants.CONSUMPTION_PHASE3 };
    String fems = FemsConstants.FEMS_NAME;
    String influxdbUser = args[0];
    String influxdbPassword = args[1];

    d.start(fems, influxdbUser, influxdbPassword, fields, fromDate, toDate);
  }
```

```
/**
 * Starts the download and saves the result to the CSV file as defined in
 * {@link FemsTools#getCsvPath()}
 *
 * @param fems
 *          the name of the FEMS to query
 * @param influxdbUser
 *          the InfluxDB username
 * @param influxdbPassword
 *          the InfluxDB password
 * @param fields
 *          the fields to query
 * @param fromDate
 *          the from-date of the query
 * @param toDate
 *          the to-date of the query
 * @throws IOException
 *           if anything goes wrong and it was not able to create the CSV file.
 */
private void start(String fems, String influxdbUser, String influxdbPassword,
    Field[] fields, DateTime fromDate, DateTime toDate) throws IOException {
  // Preparations (timestamps, query)
  System.out.println("From: " + fromDate);
  System.out.println("To : " + toDate);
  long fromTimestamp = fromDate.toDateTime(DateTimeZone.UTC).getMillis() / 1000;
  long toTimestamp = toDate.toDateTime(DateTimeZone.UTC).getMillis() / 1000;
  StringBuilder queryFromString = new StringBuilder();
  String split = "";
  for (Field field : fields) {
    queryFromString.append(split);
    queryFromString.append(field.getTechnicalName());
    split = ", ";
  }

  String query =
      String.format(QUERY, queryFromString, FemsConstants.SLICE_SECONDS,
          fromTimestamp, toTimestamp);
  System.out.println("Query: " + query);

  // Get data from influxdb
  InfluxDB influxDB =
      InfluxDBFactory.connect(FemsConstants.INFLUXDB_URL, influxdbUser,
          influxdbPassword);
  List<Serie> series = influxDB.query(fems, query, TimeUnit.SECONDS);

  // Write to internal HashMap to get rid of double entries by influxdb
  SortedMap<Long, HashMap<Field, Double>> data =
      new TreeMap<Long, HashMap<Field, Double>>();
  for (Serie serie : series) {
    Field thisField = null;
    for (Field field : fields) {
      if (field.getTechnicalName().equals(serie.getName())) {
        thisField = field;
      }
    }
    assert thisField != null;

    List<Map<String, Object>> rows = serie.getRows();
    for (Map<String, Object> row : rows) {
      Long timestamp = ((Double) row.get("time")).longValue();
      data.putIfAbsent(timestamp, new HashMap<Field, Double>());
      HashMap<Field, Double> valuePerField = data.get(timestamp);
      if (valuePerField.get(thisField) == null) {
        Double value = (Double) row.get("mean");
        value = processData(timestamp, thisField, value);
        valuePerField.put(thisField, value);
      }
    }
  }
```

```
    // Open CSV handler
    try (CSVPrinter csvFilePrinter =
        new CSVPrinter(new FileWriter(FemsTools.getCsvPath(fems).toFile()),
            FemsConstants.CSV_FORMAT)) {
      csvFilePrinter.print("timestamp");
      for (Field field : fields) {
        csvFilePrinter.print(field.getName());
      }
      csvFilePrinter.println();
      for (Long timestamp : data.keySet()) {
        csvFilePrinter.print(timestamp);
        HashMap<Field, Double> valuePerField = data.get(timestamp);
        for (Field field : fields) {
          Double value = valuePerField.get(field);
          if (value != null) {
            csvFilePrinter.print(value);
          } else {
            // TODO: here we simply replace null with 0.; better: interpolation
            csvFilePrinter.print(0.);
          }
        }
        csvFilePrinter.println();
      }
    }
    ;
  }

  /**
   * Process data values, e.g. to clean it
   *
   * @param timestamp
   *          the timestamp
   * @param field
   *          the field of this value
   * @param value
   *          the value
   * @return the cleaned value or null
   */
  private Double processData(Long timestamp, Field field, Double value) {
    if (value == null)
      return null; // replace null with 0.
    if (field instanceof PvField) {
      DateTime date = FemsTools.timestampToDateTime(timestamp);
      int hourOfDay = date.getHourOfDay(); // from 0 to 23
      if (hourOfDay < 6 || hourOfDay > 20)
        return 0.; // return 0 for PV data in the night
      if ((hourOfDay < 9 || hourOfDay > 17) && value < 65)
        return 0.; // return 0 for wrong data in morning/evening
    }
    return value;
  }
}
```

C.7.2 Field.java

```java
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

/**
 * General definition of a field identifier.
 *
 * @author Stefan Feilmeier
 */
public class Field implements Comparable<Field> {
  /** the short name of this field */
  private final String name;
  /** the technical name of this field */
  private final String technicalName;

  /**
   * Creates a specific identifier for this field.
   *
   * @param name
   *          the short name of this field
   * @param technicalName
   *          the technical name of this field
   */
  public Field(String name, String technicalName) {
    this.name = name;
    this.technicalName = technicalName;
  }

  /**
   * Compare this fields short name to the other fields short name.
   */
  @Override
  public int compareTo(Field field) {
    return name.compareTo(field.name);
  }

  /**
   * If the other object is a "Field", tests if this fields short name is equel to the
   * other fields short name. Otherwise call general {@link Object#equals()} function.
   */
  @Override
  public boolean equals(Object o) {
    if (o instanceof Field) {
      return name.equals(((Field) o).name);
    } else {
      return super.equals(o);
    }
  }

  /**
   * Gets the short name.
   *
   * @return the short name
   */
  public String getName() {
    return name;
  }
```

```
  /**
   * Gets the technical name.
   *
   * @return the technical name
   */
  public String getTechnicalName() {
    return technicalName;
  }

  public String toString() {
    return name;
  }
}
```

C.7.3 Normalizer.java

```
package de.fenecon.fems.helper;

public class Normalizer {

  private final double dataHigh;
  private final double dataLow;
  private final double normalizedHigh;
  private final double normalizedLow;

  public double getDataHigh() {
    return dataHigh;
  }

  public double getDataLow() {
    return dataLow;
  }

  public double getNormalizedHigh() {
    return normalizedHigh;
  }

  public double getNormalizedLow() {
    return normalizedLow;
  }

  /**
   * Constructs the normalization utility for a given normalization range
   *
   * @param dataHigh
   *          the high value for the input data.
   * @param dataLow
   *          the low value for the input data.
   * @param dataHigh
   *          the high value for the normalized data.
   * @param dataLow
   *          the low value for the normalized data.
   */
  public Normalizer(double dataLow, double dataHigh, double normalizedLow,
      double normalizedHigh) {
    this.dataHigh = dataHigh;
    this.dataLow = dataLow;
    this.normalizedHigh = normalizedHigh;
    this.normalizedLow = normalizedLow;
  }

  /**
   * Normalizes a value
   *
   * @param value
   *          the value to normalize
   * @return the normalized value
   */
  public double normalize(double value) {
    if (value > dataHigh)
      return normalizedHigh;
    if (value < dataLow)
      return normalizedLow;
    return ((value - dataLow) / (dataHigh - dataLow))
        * (normalizedHigh - normalizedLow) + normalizedLow;
  }

  @Override
  public String toString() {
    return "Normalizer [dataHigh=" + dataHigh + ", dataLow=" + dataLow
        + ", normalizedHigh=" + normalizedHigh + ", normalizedLow=" + normalizedLow
        + "]";
  }
```

```
  /**
   * Denormalize a value
   *
   * @param value
   *          the normalized value
   * @return the real value.
   */
  public double denormalize(double value) {
    if (value < normalizedLow)
      return dataLow;
    if (value > normalizedHigh)
      return dataHigh;
    return ((dataLow - dataHigh) * value - normalizedHigh * dataLow + dataHigh
        * normalizedLow)
        / (normalizedLow - normalizedHigh);
  }
}
```

C.7.4 Prediction.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

/**
 * Holds a prediction value and its lead window size, giving an idea about the accuracy
 * of the prediction. The higher the lead, the worse the prediction.
 *
 * @author Stefan Feilmeier
 */
public class Prediction implements Comparable<Prediction> {
  /**
   * The lead window size of this prediction. If the prediction was summed up, this is
   * an average of all base predictions.
   */
  private final float leadWindowSize;
  /** The value of this prediction */
  private final double value;

  /**
   * Creates a new Prediction object.
   *
   * @param value
   *          the value
   * @param leadWindowSize
   *          the lead window size
   */
  public Prediction(double value, float leadWindowSize) {
    this.value = value;
    this.leadWindowSize = leadWindowSize;
  }

  /**
   * Compare the accuracy (lead window size) of two {@link Prediction}s.
   */
  @Override
  public int compareTo(Prediction o) {
    return (int) (leadWindowSize - o.leadWindowSize);
  }

  /**
   * Gets the lead window size. Smaller is more accurate.
   *
   * @return the lead window size
   */
  public float getLeadWindowSize() {
    return leadWindowSize;
  }

  /**
   * Gets the value.
   *
   * @return the value
   */
  public double getValue() {
    return value;
  }

  @Override
  public String toString() {
    return String.format("[Value=%.2f, Lead=%.1f]", value, leadWindowSize);
  } }
```

C.7.5 PredictionAgent.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

import java.util.concurrent.ConcurrentSkipListSet;

/**
 * General definition of a {@link PredictionAgent}
 *
 * @author Stefan Feilmeier
 */
public interface PredictionAgent extends Comparable<PredictionAgent> {

  /**
   * Gets the best prediction for a given timestamp.
   *
   * @param timestamp
   *          the timestamp
   * @return the best prediction
   */
  public Prediction getBestPredictionAtTimestamp(long timestamp);

  /**
   * Get the field descriptor of this agent.
   *
   * @return the field descriptor
   */
  public Field getField();

  /**
   * Gets all available predictions for a given Timestamp.
   *
   * @param timestamp
   *          the timestamp
   * @return the set of predictions or empty set
   */
  public ConcurrentSkipListSet<Prediction> getPredictionsAtTimestamp(long timestamp);

}
```

C.7.6 PredictionAgentFactory.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

import java.io.FileReader;
import java.nio.file.DirectoryStream;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Properties;

import org.encog.ml.MLRegression;
import org.encog.persist.EncogDirectoryPersistence;

import de.fenecon.fems.FemsConstants;

/**
 * Abstract class to create a new {@link PredictionAgent}.
 *
 * @author Stefan Feilmeier
 */
public abstract class PredictionAgentFactory {

  /**
   * Loads the Encog machine learning network file from "{field}_*.method" and the
   * properties file from "properties.prop" inside the FILESPATH folder defined in
   * {@link FemsConstants}. Those files should have been created by a FADSE simulation.
   * If anything goes wrong, an exception is thrown.
   *
   * @param field
   *          the field descriptor
   * @return a {@link Predictor} for this field
   * @throws Exception
   *           if {@link Predictor} could not be loaded
   */
  protected static Predictor getPredictor(Field field) throws Exception {
    // Import Properties file
    final Normalizer normalizer;
    {
      Properties properties = new Properties();
      FileReader reader =
          new FileReader(Paths.get(FemsConstants.FILESPATH, "properties.prop").toFile());
      properties.load(reader);
      int leadWindowSize = Integer.parseInt(properties.getProperty("LeadWindowSize"));
      if (leadWindowSize != FemsConstants.MAX_PREDICTION_WINDOW) {
        throw new Exception(
            "Provided properties and MLMethods are not fitting for Prediction Window!");
      }
      // Create Normalizer
      String prefix = "Normalizer_" + field + "_";
      Double dataLow = Double.parseDouble(properties.getProperty(prefix + "DataLow"));
      Double dataHigh = Double.parseDouble(properties.getProperty(prefix + "DataHigh"));
      Double normalizedLow =
          Double.parseDouble(properties.getProperty(prefix + "NormalizedLow"));
      Double normalizedHigh =
          Double.parseDouble(properties.getProperty(prefix + "NormalizedHigh"));
      normalizer = new Normalizer(dataLow, dataHigh, normalizedLow, normalizedHigh);
    }

    // Import Encog file
    Integer lagWindowSize = null;
    MLRegression method = null;
```

```
    {
      try (DirectoryStream<Path> paths =
          Files.newDirectoryStream(Paths.get(FemsConstants.FILESPATH),
              String.format("%s_*.method", field))) {
        for (Path path : paths) {
          String filename =
              path.getName(path.getNameCount() - 1).toString().split("\\.")[0];
          for (String part : filename.split("_")) {
            if (part.startsWith("LAG")) {
              lagWindowSize = Integer.parseInt(part.substring(3));
            }
          }
          method =
              (MLRegression) EncogDirectoryPersistence.loadObject(path.toAbsolutePath()
                  .resolveSibling(filename + ".method").toFile());
        }
      } catch (Exception e) {
        throw e;
      }
    }
    if (method == null || lagWindowSize == null) {
      throw new Exception("No MLMethod found!");
    }

    // Create Predictor
    final Predictor predictor = new Predictor(method, lagWindowSize, normalizer);
    return predictor;
  }
}
```

C.7.7 PredictionAgentImpl.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

import java.util.HashMap;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;
import java.util.concurrent.ConcurrentSkipListSet;

import de.fenecon.fems.FemsConstants;
import de.fenecon.fems.agent.consumption.ConsumptionAgentFactory;
import de.fenecon.fems.agent.source.pv.PvAgentFactory;
import de.fenecon.fems.ess.EssListener;

/**
 * General implementation of a {@link PredictionAgent}. It manages a {@link Predictor},
 * which is carrying out the actual prediction as soon as a new value is received via
 * {@link EssListener}..
 *
 * @author Stefan Feilmeier
 */
public abstract class PredictionAgentImpl implements EssListener, PredictionAgent {
  /** Collection of prediction results per timestamp */
  private final ConcurrentHashMap<Long, ConcurrentSkipListSet<Prediction>>
predictionsPerTimestamp =
      new ConcurrentHashMap<Long, ConcurrentSkipListSet<Prediction>>();

  /**
   * The Predictor for this field
   */
  private final Predictor predictor;

  /**
   * Creates a new {@link PredictionAgentImpl}. Use with an appropriate implementation
   * of {@link PredictionAgentFactory}, like {@link ConsumptionAgentFactory} or
   * {@link PvAgentFactory}.
   *
   * @param predictor
   *          the predictor for this agent
   */
  public PredictionAgentImpl(Predictor predictor) {
    this.predictor = predictor;
  }

  /**
   * Add a new prediction; overwrite existing prediction only if the lead of the new
   * {@link Prediction} is smaller (more accurate).
   *
   * @param timestamp
   *          the timestamp of the prediction
   * @param prediction
   *          the prediction
   */
  public void addPrediction(long timestamp, Prediction prediction) {
    if (prediction == null)
      return; // not accepting invalid predictions
    predictionsPerTimestamp.putIfAbsent(timestamp,
        new ConcurrentSkipListSet<Prediction>());
    ConcurrentSkipListSet<Prediction> predictions =
        predictionsPerTimestamp.get(timestamp);
    predictions.add(prediction);
  }
```

```
/**
 * Gets the best prediction (= the prediction with the highest accuracy, the smallest
 * lead window size) from the list.
 *
 * @return
 */
@Override
public Prediction getBestPredictionAtTimestamp(long timestamp) {
  // use internal sorting of ConcurrentSkipListSet to return the most
  // accurate prediction
  ConcurrentSkipListSet<Prediction> predictions =
      predictionsPerTimestamp.get(timestamp);
  try {
    return predictions.first();
  } catch (Exception e) {
    return null;
  }
}

@Override
public ConcurrentSkipListSet<Prediction> getPredictionsAtTimestamp(long timestamp) {
  ConcurrentSkipListSet<Prediction> predictions =
      predictionsPerTimestamp.get(timestamp);
  if (predictions == null) {
    return new ConcurrentSkipListSet<Prediction>();
  }
  return predictions;
}

/**
 * Remove old predictions (elder than current timestamp) from the cache.
 *
 * @param timestamp
 *          the current timestamp
 */
private void clearOldPredictions(long timestamp) {
  for (Long oldTimestamp : predictionsPerTimestamp.keySet()) {
    if (oldTimestamp < timestamp) {
      try {
        predictionsPerTimestamp.remove(oldTimestamp);
      } catch (NullPointerException e) {
        ;
      }
    }
  }
}

/**
 * Compare the {@link Field} descriptor of two {@link PredictionAgent}s.
 */
@Override
public int compareTo(PredictionAgent o) {
  return getField().compareTo(o.getField());
}

/**
 * On arrival of a new value from {@link EssListener}, add the new value as the final
 * prediction (lead = 0) and execute the predictor with this new value.
 *
 * @param timestamp
 *          the timestamp of the value
 * @param value
 *          the value
 */
@Override
public void newValue(long timestamp, double value) {
  // add the current value to predictions
  addPrediction(timestamp, new Prediction(value, 0));
```

```
    HashMap<Integer, Prediction> predictionsPerLead =
        predictor.addValueAndPredict(value);
    for (Map.Entry<Integer, Prediction> predictionPerLead : predictionsPerLead
        .entrySet()) {
      long futureTimestamp =
          timestamp + FemsConstants.SLICE_SECONDS * predictionPerLead.getKey();
      addPrediction(futureTimestamp, predictionPerLead.getValue());
    }
    clearOldPredictions(timestamp);
  }

  @Override
  public String toString() {
    return getField().getName();
  }
}
```

C.7.8 Predictor.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

import java.util.HashMap;

import org.encog.ml.MLRegression;
import org.encog.ml.data.MLData;
import org.encog.ml.data.basic.BasicMLData;

/**
 * A wrapper around the Encog objects and methods to calculate the result of a machine
 * learning network.
 *
 * @author Stefan Feilmeier
 */
public class Predictor {
  /**
   * The lag window size of this prediction method. It represents the number of past
   * values in the input vector for the machine learning method.
   */
  private final int lagWindowSize;
  /**
   * The Encog method used for this network. The object is loaded from a serialized file
   * using {@link PredictionAgentFactory}
   */
  private final MLRegression method;
  /**
   * The Normalizer used for this network. The object is loaded from a serialized file
   * using {@link PredictionAgentFactory}
   */
  private final Normalizer normalizer;
  /**
   * The time-series window cache for the input vector of this method
   */
  private final RingCache windowCache;

  /**
   * Creates a new Predictor. Use via {@link PredictionAgentFactory}.
   *
   * @param method
   *          the Encog prediction method
   * @param lagWindowSize
   *          the lag window size for the time-series window
   * @param normalizer
   *          the Normalizer
   */
  public Predictor(MLRegression method, int lagWindowSize, Normalizer normalizer) {
    this.method = method;
    this.lagWindowSize = lagWindowSize;
    this.normalizer = normalizer;
    this.windowCache = new RingCache(lagWindowSize);
  }
```

```
  /**
   * Adds a new value to the time-series window and calculates the new
   * {@link Prediction}s.
   *
   * @param value
   *          the new value
   * @return the calculated predictions per leadWindow
   */
  public synchronized HashMap<Integer, Prediction> addValueAndPredict(double newValue) {
    HashMap<Integer, Prediction> predictionsPerLead =
        new HashMap<Integer, Prediction>();
    Double newNormValue = normalizer.normalize(newValue);
    windowCache.push(newNormValue);
    /*
     * TODO while (!window.isReady()) { window.add(slice); // if window not full, just
     * interpolate current // value => better a bad prediction than none }
     */
    if (windowCache.isWindowReady()) {
      MLData mlInputData = new BasicMLData(windowCache.getWindow());
      MLData mlOutputData = method.compute(mlInputData);
      for (int lead = 1; lead < mlOutputData.size() + 1; lead++) {
        double normValue = mlOutputData.getData(lead - 1);
        double value = normalizer.denormalize(normValue);
        predictionsPerLead.put(lead, new Prediction(value, lead));
      }
    }
    return predictionsPerLead;
  }

  /**
   * Gets the Encog machine learning network method.
   *
   * @return the Encog method.
   */
  public MLRegression getMethod() {
    return method;
  }

  /**
   * Gets the Normalizer.
   *
   * @return the Normalizer
   */
  public Normalizer getNormalizer() {
    return normalizer;
  }

  /**
   * Gets the lag window size. It represents the number of past values in the input
   * vector for the machine learning method.
   *
   * @return the lag window size.
   */
  public int getLagWindowSize() {
    return lagWindowSize;
  }
}
```

C.7.9 RingCache.java

```
/**
 * Copyright (c) 2015 Stefan Feilmeier.
 * All rights reserved. This program and the accompanying materials
 * are made available under the terms of the Eclipse Public License v1.0
 * which accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 */
package de.fenecon.fems.helper;

/**
 * A fixed size array that implements a ring cache.
 *
 * Part of the Source:
 * http://stackoverflow.com/questions/13157675/looking-for-a-circular
 * -fixed-size-array-based-deque
 *
 * @author Stefan Feilmeier
 */
public class RingCache {
  private final int size;
  private final Double[] data;
  private int n = 0;
  /**
   * Creates a new RingCache with the defined size
   *
   * @param size
   *          the size of the cache
   */
  public RingCache(int size) {
    this.size = size;
    data = new Double[size];
  }

  /**
   * Pushes a value to the end of the array
   *
   * @param value
   *          the new value
   */
  public void push(Double value) {
    data[n] = value;
    n = (n + 1) % data.length;
  }

  public void shift(Double value) {
    data[n = (n - 1) % data.length] = value;
  }

  public Double get(int index) {
    return data[(n + index) % data.length];
  }

  public int getSize() {
    return size;
  }

  public boolean isWindowReady() {
    return get(0) != null;
  }

  public final double[] getWindow() {
    double[] window = new double[size];
    for (int i = 0; i < size; i++) {
      window[i] = get(i).doubleValue();
    }
    return window;
  } }
```

D. Eclipse Public License - v 1.0

THE ACCOMPANYING PROGRAM IS PROVIDED UNDER THE TERMS OF THIS ECLIPSE PUBLIC LICENSE ("AGREEMENT"). ANY USE, REPRODUCTION OR DISTRIBUTION OF THE PROGRAM CONSTITUTES RECIPIENT'S ACCEPTANCE OF THIS AGREEMENT.

1. DEFINITIONS

"Contribution" means:

a) in the case of the initial Contributor, the initial code and documentation distributed under this Agreement, and

b) in the case of each subsequent Contributor:

i) changes to the Program, and

ii) additions to the Program;

where such changes and/or additions to the Program originate from and are distributed by that particular Contributor. A Contribution 'originates' from a Contributor if it was added to the Program by such Contributor itself or anyone acting on such Contributor's behalf. Contributions do not include additions to the Program which: (i) are separate modules of software distributed in conjunction with the Program under their own license agreement, and (ii) are not derivative works of the Program.

"Contributor" means any person or entity that distributes the Program.

"Licensed Patents" mean patent claims licensable by a Contributor which are necessarily infringed by the use or sale of its Contribution alone or when combined with the Program.

"Program" means the Contributions distributed in accordance with this Agreement.

"Recipient" means anyone who receives the Program under this Agreement, including all Contributors.

2. GRANT OF RIGHTS

a) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free copyright license to reproduce, prepare derivative works of, publicly display, publicly perform, distribute and sublicense the Contribution of such Contributor, if any, and such derivative works, in source code and object code form.

b) Subject to the terms of this Agreement, each Contributor hereby grants Recipient a non-exclusive, worldwide, royalty-free patent license under Licensed Patents to make, use, sell, offer to sell, import and otherwise transfer the Contribution of such Contributor, if any, in source code and object code form. This patent license shall apply to the combination of the Contribution and the Program if, at the time the

Contribution is added by the Contributor, such addition of the Contribution causes such combination to be covered by the Licensed Patents. The patent license shall not apply to any other combinations which include the Contribution. No hardware per se is licensed hereunder.

c) Recipient understands that although each Contributor grants the licenses to its Contributions set forth herein, no assurances are provided by any Contributor that the Program does not infringe the patent or other intellectual property rights of any other entity. Each Contributor disclaims any liability to Recipient for claims brought by any other entity based on infringement of intellectual property rights or otherwise. As a condition to exercising the rights and licenses granted hereunder, each Recipient hereby assumes sole responsibility to secure any other intellectual property rights needed, if any. For example, if a third party patent license is required to allow Recipient to distribute the Program, it is Recipient's responsibility to acquire that license before distributing the Program.

d) Each Contributor represents that to its knowledge it has sufficient copyright rights in its Contribution, if any, to grant the copyright license set forth in this Agreement.

3. REQUIREMENTS

A Contributor may choose to distribute the Program in object code form under its own license agreement, provided that:

a) it complies with the terms and conditions of this Agreement; and

b) its license agreement:

i) effectively disclaims on behalf of all Contributors all warranties and conditions, express and implied, including warranties or conditions of title and non-infringement, and implied warranties or conditions of merchantability and fitness for a particular purpose;

ii) effectively excludes on behalf of all Contributors all liability for damages, including direct, indirect, special, incidental and consequential damages, such as lost profits;

iii) states that any provisions which differ from this Agreement are offered by that Contributor alone and not by any other party; and

iv) states that source code for the Program is available from such Contributor, and informs licensees how to obtain it in a reasonable manner on or through a medium customarily used for software exchange.

When the Program is made available in source code form:

a) it must be made available under this Agreement; and

b) a copy of this Agreement must be included with each copy of the Program.

Contributors may not remove or alter any copyright notices contained within the Program.

Each Contributor must identify itself as the originator of its Contribution, if any, in a manner that reasonably allows subsequent Recipients to identify the originator of the Contribution.

4. COMMERCIAL DISTRIBUTION

Commercial distributors of software may accept certain responsibilities with respect to end users, business partners and the like. While this license is intended to facilitate the commercial use of the Program, the Contributor who includes the Program in a commercial product offering should do so in a manner which does not create potential liability for other Contributors. Therefore, if a Contributor includes the Program in a commercial product offering, such Contributor ("Commercial Contributor") hereby agrees to defend and indemnify every other Contributor ("Indemnified Contributor") against any losses, damages and costs (collectively "Losses") arising from claims, lawsuits and other legal actions brought by a third party against the Indemnified Contributor to the extent caused by the acts or omissions of such Commercial Contributor in connection with its distribution of the Program in a commercial product offering. The obligations in this section do not apply to any claims or Losses relating to any actual or alleged intellectual property infringement. In order to qualify, an Indemnified Contributor must: a) promptly notify the Commercial Contributor in writing of such claim, and b) allow the Commercial Contributor to control, and cooperate with the Commercial Contributor in, the defense and any related settlement negotiations. The Indemnified Contributor may participate in any such claim at its own expense.

For example, a Contributor might include the Program in a commercial product offering, Product X. That Contributor is then a Commercial Contributor. If that Commercial Contributor then makes performance claims, or offers warranties related to Product X, those performance claims and warranties are such Commercial Contributor's responsibility alone. Under this section, the Commercial Contributor would have to defend claims against the other Contributors related to those performance claims and warranties, and if a court requires any other Contributor to pay any damages as a result, the Commercial Contributor must pay those damages.

5. NO WARRANTY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, THE PROGRAM IS PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Each Recipient is solely responsible for determining the appropriateness of using and distributing the Program and assumes all risks associated with its exercise of rights under this Agreement , including but not limited to the risks and costs of program errors, compliance with applicable laws, damage to or loss of data, programs or equipment, and unavailability or interruption of operations.

6. DISCLAIMER OF LIABILITY

EXCEPT AS EXPRESSLY SET FORTH IN THIS AGREEMENT, NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAM OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. GENERAL

If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement, and without further action by the parties hereto, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.

If Recipient institutes patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Program itself (excluding combinations of the Program with other software or hardware) infringes such Recipient's patent(s), then such Recipient's rights granted under Section 2(b) shall terminate as of the date such litigation is filed.

All Recipient's rights under this Agreement shall terminate if it fails to comply with any of the material terms or conditions of this Agreement and does not cure such failure in a reasonable period of time after becoming aware of such noncompliance. If all Recipient's rights under this Agreement terminate, Recipient agrees to cease use and distribution of the Program as soon as reasonably practicable. However, Recipient's obligations under this Agreement and any licenses granted by Recipient relating to the Program shall continue and survive.

Everyone is permitted to copy and distribute copies of this Agreement, but in order to avoid inconsistency the Agreement is copyrighted and may only be modified in the following manner. The Agreement Steward reserves the right to publish new versions (including revisions) of this Agreement from time to time. No one other than the Agreement Steward has the right to modify this Agreement. The Eclipse Foundation is the initial Agreement Steward. The Eclipse Foundation may assign the responsibility to serve as the Agreement Steward to a suitable separate entity. Each new version of the Agreement will be given a distinguishing version number. The Program (including Contributions) may always be distributed subject to the version of the Agreement under which it was received. In addition, after a new version of the Agreement is published, Contributor may elect to distribute the Program (including its Contributions) under the new version. Except as expressly stated in Sections 2(a) and 2(b) above, Recipient receives no rights or licenses to the intellectual property of any

Contributor under this Agreement, whether expressly, by implication, estoppel or otherwise. All rights in the Program not expressly granted under this Agreement are reserved.

This Agreement is governed by the laws of the State of New York and the intellectual property laws of the United States of America. No party to this Agreement will bring a legal action under this Agreement more than one year after the cause of action arose. Each party waives its rights to a jury trial in any resulting litigation.