Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Master Thesis

Farid Naimi

Java Based Load Forecasting for a Small Number of Public Properties

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Farid Naimi

Java Based Load Forecasting for a Small Number of Public Properties

Master thesis based on the examination and study regulations for the
Master of Engineering degree programme
Information Engineering
at the Department of Information and Electrical Engineering
of the Faculty of Engineering and Computer Science
of the University of Applied Sciences Hamburg

Supervising examiner : Prof. Dr. –Ing. Gustav Vaupel
Second examiner : Prof. Dr. –Ing. Franz Schubert

Day of delivery March 20th 2008

Farid Naimi

Title of the Master Thesis

Java Based Load Forecasting of a Small Number of Public Properties

Keywords

Load Management, Load Forecasting, Load Forecasting Methods, Neural Network, UML, Java

Abstract

This master thesis studies different approaches of load forecasting. The Method Neural Network is used as Load forecasting method. A graphical user interface is designed using Java to forecast the load shape. This program gets the historical data from the database and makes use of the method Neural Network to calculate a typical day. Testing is carried out on the real load data of a small number of public properties placed in Germany. Furthermore, the real load data is compared with the program generated values.

Farid Naimi

Thema der Masterarbeit

Java-basierte Lastprognose für eine geringe Anzahl von Liegenschaften

Stichworte

Lastmanagement, Lastprognose, Lastprognose Methoden, Neuronales Netz, UML, Java

Kurzzusammenfassung

Diese Masterarbeit beschreibt verschiedene Ansätze der Lastprognose. Die Methode Neuronales Netz wird zur Lastprognose benutzt. Eine grafische Benutzeroberfläche wird mit Java entwickelt, um den Lastgang vorhersagen zu können. Dieses Programm holt sich den historischen Daten aus der Datenbank und nutzt die Methode Neuronales Netz zur Berechnung eines typischen Tages. Die Analyse befasst sich mit tatsächlichen Lastgangdaten von einer kleinen Anzahl von Liegenschaften in Deutschland. Darüber hinaus werden die tatsächlichen Daten mit den vom Program generierten Werten verglichen.

Table of Content

List of Figures

List of Tables

Chapter 1

Introduction

This master thesis is one part of the E-island R&D project. E-island stands for expandable internet sustained load and demand side management for the integration into virtual power plants and is a R&D project financed by the program aif/FH of the German ministry of education and research BMBF. The project is carried out by a public private partnership consortium led by the University of Applied Sciences Hamburg (HAW, Prof. Dr.-Ing. F. Schubert and Prof. Dr.-Ing. G. Vaupel). Its other members are

- the department of urban development and environment of the city of Hamburg,
- Steag Saarenergie - a German electricity utility operating a virtual power plant marketing reserve capacity
- SUmBi and Envidatec, two German engineering companies.

Aims of the project E-island

The project's central aim is to create a network (internet based) of 30 public properties equipped with load management hard- and software at low and medium voltage level. This network will be the basis for the examination of two research tasks:

- Find out how and to what degree 30 independent load management systems can be synchronized with the aim of harmonizing the resulting load curve rather than shaping the individual ones of the properties involved.
- Find out how much load can be cut off for how long with the aim of selling it as reserve capacity to the virtual power plant (VPP) of Steag Saarenergie.

Figure 1.1 shows some of the important public properties which are involved in this project.

Figure 1.1: GSM/GPRS Network of 35 load intensive public properties at medium voltage level [1]

Modeling and day ahead simulation based on the real-time data of the public properties are used to examine the reserve capacity and load curve smoothing. A equipment called Vida 84 communication modules is installed in the public properties to get accurate real time meter readings and status data of the load management systems. The following figure shows the intended system structure [1].

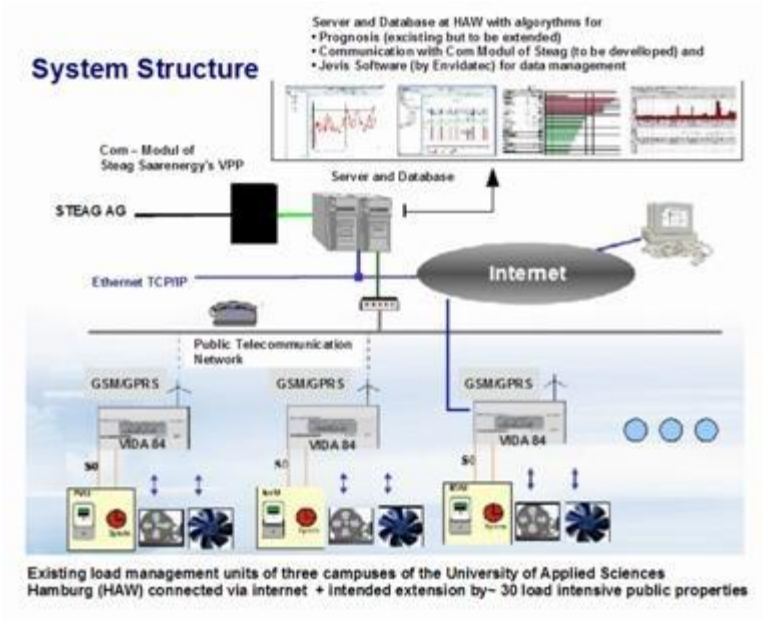


Figure 1.2: System Structure [1]

1.1 State of the Art

In addition to the work (kWh) the city of Hamburg paid a prize for the highest consumed power (kW) in a year for each of the its approximately 200 properties at medium-voltage level.), also. For the majority of these properties, this peak lies below 400 kW. Figure 1.3 shows that in 2003 only 37 properties were responsible for higher peaks.

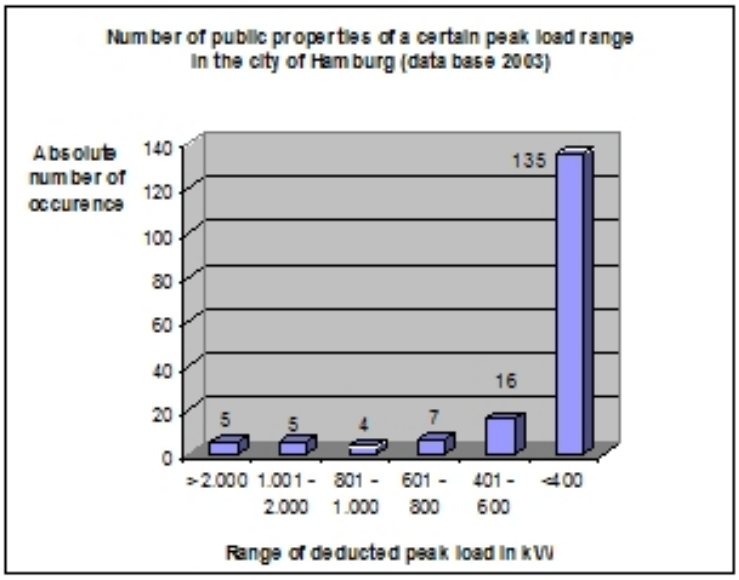


Figure 1.3: Number of properties per peak load [1]

In 2003, for all 200 properties at medium voltage level the sum of all peak loads was 58.560 kW. As Figure 1.4 shows, approximately, 65% of this was caused by the 37 most load intensive properties of the city.

The load curves of these properties were well above 400 kW in their annual peak. Therefore, for the project all the properties were selected, whose peak exceeded the 400 kW limit significantly in years 2003 to 2005. This applies to about 40 properties. The university has the highest load profile with an annual peak of about 5.3 MW followed by the properties „METHA III“ and „Großmarkt“ each with about 3 MW. The Campus Berliner Tor of HAW is ranked at sixth place with an annual peak of about 1.75 MW.

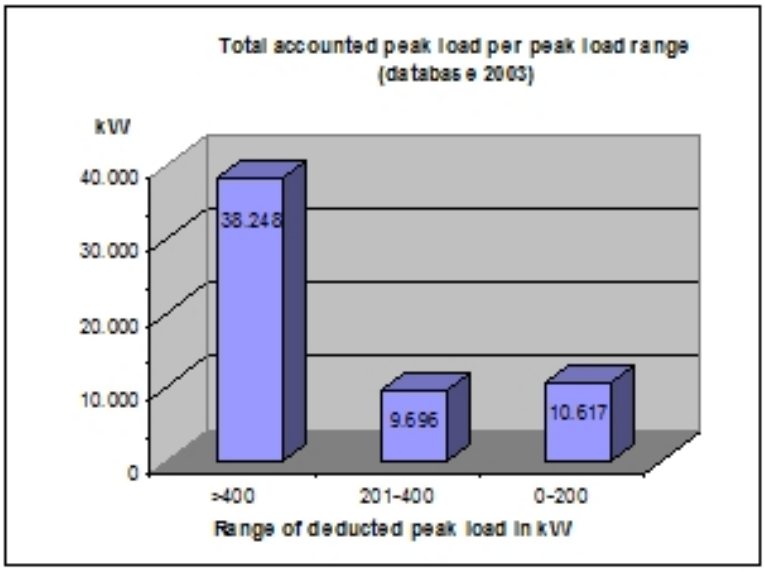


Figure 1.4: Total accounted peak load per load range [1]

Further investigation of the daily load profiles of the 40 properties also showed that the resulting sum load curve in 2005 does not have a value above 25,300 kW. However, due to the current legal situation the network usage costs for each property is based on the full usage hours (annual work divided by annual power peak). As a result in 2005 the city was billed for power of over 35,000 kW for the 40 surveyed properties. Figure 1.5 below shows the summarization of the daily load curves of the properties. Some unexpected load peaks occur during the day so the profile is far from being a simple smooth bell curve (ideal case). It shows the resulting stacked load profile of January 5 2005 of the 40 properties with a resolution of one step every 15 minutes. The power consumed by all of the properties together does not exceed 25000 kW. The curve shows about 15 single load peaks, each being about two MW high.

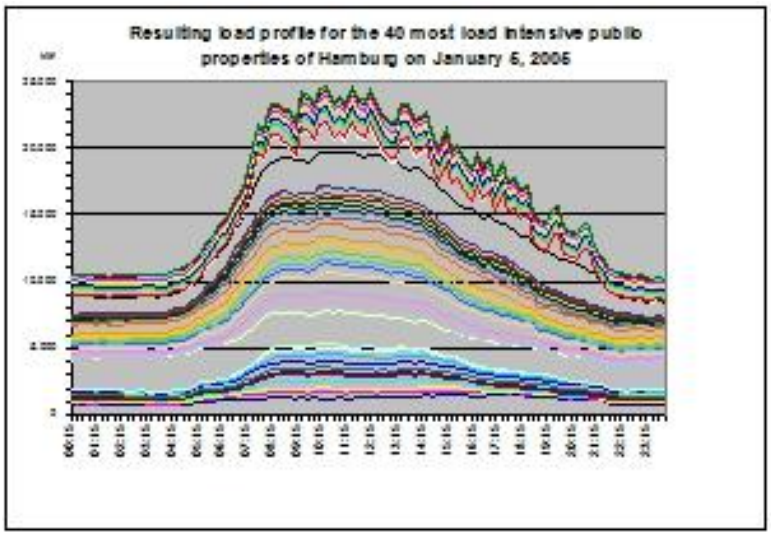


Figure 1.5: Resulting load profile [1]

However, it is expected that the existing possibilities for load management in the properties would be enough to smooth those peaks. Applying a rough guess of these possibilities to the sum curve in figure 1.5 results in a load profile shown in figure 1.6. Note that this figure does not show the simulation results, but it is hand calculated. Nevertheless the figure illustrates that without the influence of that load fluctuation, the load curve shows a quite harmonious performance. This effect has been found for all studied daily profiles. Now, the resulting sum load curve only shows a peak which is about 200 to 300 kW high. In addition, it has a distinctive bell-curve shape with a high base-load share of about 10 MW. Such a curve should be well predictable.

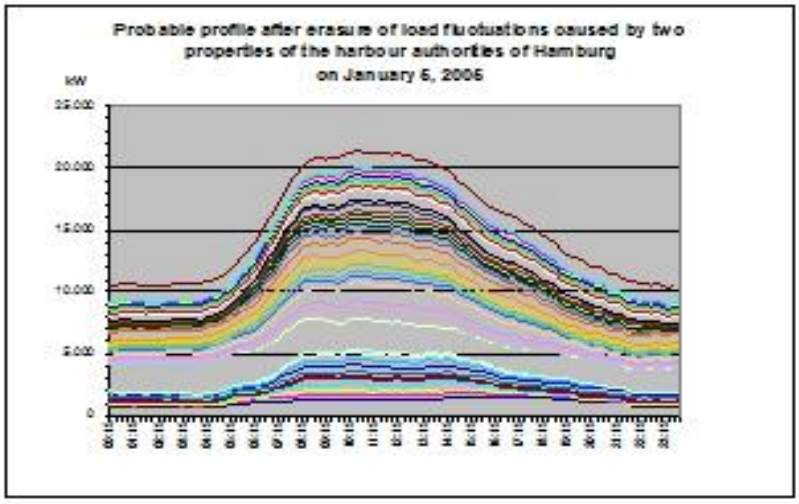


Figure 1.6: Probable profile after erasure of load fluctuation [1]

1.2 Purpose of this Master Thesis

The task of the Master Thesis was to design a program which could calculate the load consumption for a typical day of the individual buildings which are investigated in the R&D "E-Island" , i.e. to make load forecasting for the selected properties and as output create comma separated value file (CSV file) containing this information. To make the calculation of the typical day as accurate as possible, the following specifications have to be entered by the user using a graphical user interface (GUI):

- Public properties of which the load profiles should be calculated,
- Weekday,
- Season,
- Holiday restriction

Using this input, the program connects itself with a server placed at the Envidatec company and gets the collected historical data from the database on that server. Historical data is taken order to make an appropriate load forecasting. The database includes timestamps and for each public property and the load consumed in one sample per minute.

This master thesis studies different approaches for load forecasting. As it may be seen from literature, many methods have been developed for load forecasting. From the experimental results the conclusion can be drawn that different methods might outperform others in different situations, i.e. one method might gain the lowest prediction error for one time point, and another might for another time point. The method Neural Network has been chosen which is well suited for this thesis because this method has a very simple input structure and is most used method in load forecasting The selected method was tested on the real load data of a small number of public properties in Hamburg.

The forecasted values of the typical day load is then used as an input of the Load Estimation Project (different master thesis in the E-Island project) which simulates the behavior of building devices (ventilation, air conditioning etc.) which a load management system in these buildings would make use of. The Load Estimation Project simulates a new ("managed") load chart via Matlab Simulink. The Load Estimation Project interfaces with yet another thesis called "Load Control Management". The Load Control Management Project simulates the algorithmic behavior of a load management system. It decides which building devices at which substations shall be switched off or on. After deciding this, it sends a matrix back which informs the Load Estimation Project to switch off the respective devices. Figure 1.7 shows a block diagram how these three projects interact with each other.

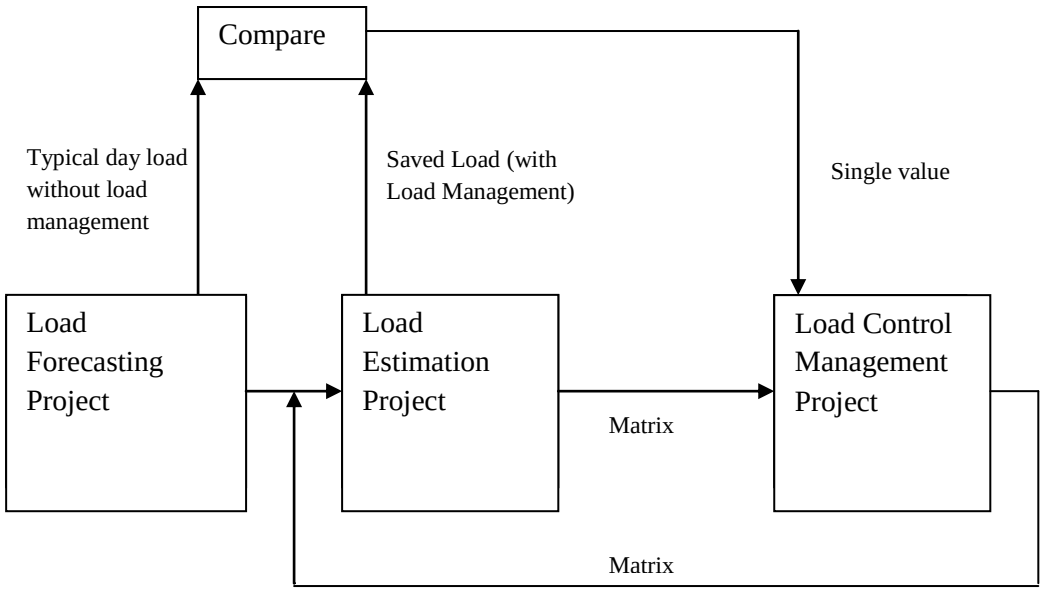


Figure 1.7 Blockdiagram of the three projects

Since, the Load Forecasting Project (this master thesis) contains different elements such as GUI, Client-Server Connection and some specific calculation, it was decided to use the object oriented programming language JAVA.

1.3 Structure of the Thesis

Chapter two describes the basics of Load Management. Chapter three discusses the properties of the objected oriented programming language JAVA. Furthermore, it lists the main features of this programming language and why it performs better as compared to other programming languages.

Chapter four concentrates on the subject of load forecasting in general. First the different factors affecting the load are discussed. Then the most popular conventional methods are shortly introduced. Furthermore, it gives a short introduction to neural network. The most popular network type, the Multi-Layer Perceptron network (MLP) is discussed. The basic idea in applying MLP based methods to the problem is described. According to that, chapter five describes the algorithm used to forecast the daily load profile.

From chapter six to chapter eight, the programming requirement and realization are described. First, some Unified Modeling Language (UML) diagrams like Use Case diagram, Class diagram and Sequence diagram are presented which graphically show the way of solutions. Then the programming part is described. Chapter nine discusses all the different test results achieved with the algorithm used for load forecasting. In addition, it also compares the really measured values with the program generated values.

Chapter 2

Load Management

Load Management is needed as the name states to manage load. Its task is to bring the load down from a high load value to a predefined maximum load allowance. It may be understood as follows: if a building tries to exceed the maximum load allowance, the load management system switches off some of the devices inside that specific building. Thus, the task of the load management system is to keep the load near the maximum load allowance value. Furthermore, each device in a building has a priority telling the load management system about its minimum and maximum shut down time. If the minimum shut down is over, the device may want to be switched on, again. The purpose of load management systems is to automate these procedures, and to run them as fast and efficiently as possible, depending on the circumstances. Keeping the load below the maximum load allowance may not be difficult; the only problem could be the flattering of the load around the maximum load allowance.

2.1 One-Way Load Management

"*Load management is called one-way when devices used to control loads are unable to report back to the controller whether the respective loads are running at the time of management*" [2].

In ideal case, if all devices are switched on, the sum load could be brought down to the maximum load allowance in a single load management cycle between two polls for demand data [2]. This is shown in figure 2.1 graphically. Vertical dashed lines are poll times, the red line is the maximum load allowance and the blue line the real time measured load values.

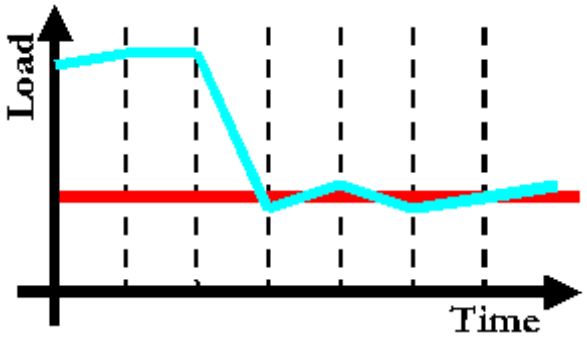


Figure 2.1: One-Way Load Management (ideally)

In reality, some devices are connected which are not consuming any power at that time. Since in the one-way load management system, it is not possible of knowing that while switching load, the situation shown in figure 2.2 is achieved. It may be seen that after the first load management cycle that not enough loads was shed, thus, another cycle is executed. As a result, the system gets very slow which means the load stays above the maximum load allowance during that time.

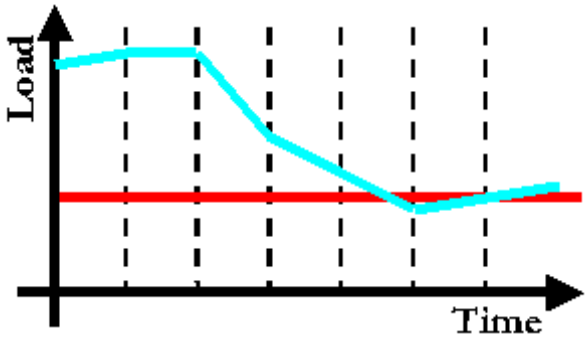


Figure 2.2: One-Way Load Management (reality)

2.2 Two-Way Load Management

"Load management is called two-way when devices used to control loads are able to report back to the controller whether the respective loads are running at the time of management" [2].

When a two-way load management system enters into a shed cycle, every outgoing control request is not only acknowledged, but also followed by the kW number of the target load at the time of the disconnect. In this case, the system continues to turn devices off as long as it takes to bring total load down to the desired level in one management cycle (situation depicted in Figure 2.1).

Chapter 3

Java

Java is a programming language like C or C++. As compared to other programming language, Java is a high level language. It has many features which are not part of the other languages. Another feature of Java is the Java bytecode. Java bytecode is a compiler that transforms the Java source code to bytecode that runs in the Java Virtual Machine (JVM). JVM is a program that runs on any operating system and it takes the compile bytecode as input and interprets them like if it were a physical processor executing machine code. Figure 3.1 visualizes this situation. After a Java program is written, it is compiled to Java bytecode which is then interpreted to machine code.

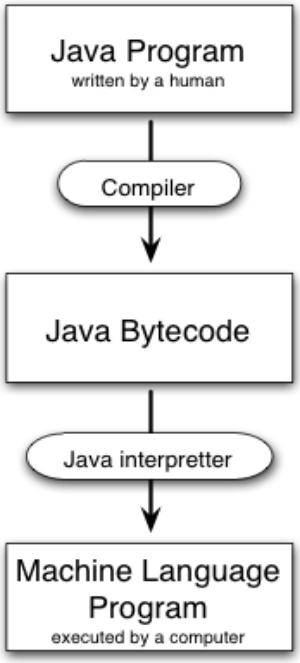


Figure 3.1: Java Program Hierarchy

3.1 Features

Java has additional features as compared to other languages which make Java so powerful and popular programming language. Java is platform independent and object oriented which does not allowed coding outside of class definitions, including the main() method. As told before, the code is compiled to bytecodes which are interpreted by JVM. The advantage of doing this is that it provides portability to any machine for which a virtual machine has been written. This compilation and interpretation steps allow for extensive code checking and improved security.

Another additional feature of Java is being robust like it does exception handling, type checking and assures that local variable are initialized. The programming languages like C and C++ have some dangerous features which are removed in Java. Dangerous features include memory pointers, preprocessor and the problem of defining array. Thus, in Java is no memory pointers, no preprocessor and it does array index limit checking. Java also has the ability of doing memory management automatically. The so called garbage collection is responsible for that.

Java is a very secure programming language. As mentioned before, there are no memory pointers, the program runs inside the virtual machine environment, it does array limit checking. The security manager checks which resources a class can access such as reading or writing to the hard disk. As compared to C++, Java does dynamic binding whereas C++ uses static binding. Java makes use of dynamic binding like the linking of data and methods to where they are located, is done at runtime. New classes can be loaded while the program is running. Even if libraries are recompiled, it is not needed to recompile the code that uses classes in those libraries. This differs from C++, which uses static binding. This can result that the linked code is changed and memory pointers then point to the wrong addresses. The performance of Java is very good. Java is up to 50% to 100% faster as compared to speed of C++ programs.

Threading is also one of the important features of Java which is great for multimedia. Java also supports networking. The built in networking allows developing internet communication applications.

Features like eliminating memory pointers and checking array limits greatly help the programmer to remove program errors. The garbage collector relieves programmers of the big project of memory management. As a result, all of these features can lead to make the program development fast compared to C/C++ programming.

There are two ways of executing Java programs. An applet runs under the control of a browser and an application runs like other programs alone, with the support of a virtual machine.

3.2 The Java Virtual Machine (JVM)

Java source code is compiled to low-level machine instruction which is known as bytecode. Java source code is compiled to standard and platform independent bytecodes which runs with JVM. The JVM is a separate program that is optimized for the specific platform on which Java code is executed. Figure 3.2 shows shows the Java Component Structue which may be explained as follows: Java source is compiled into bytecodes, which are platform independent. Each platform has installed a JVM that is specific to its operating system. The Java bytecodes from the source get interpreted through the JVM into appropriate platform dependent actions.

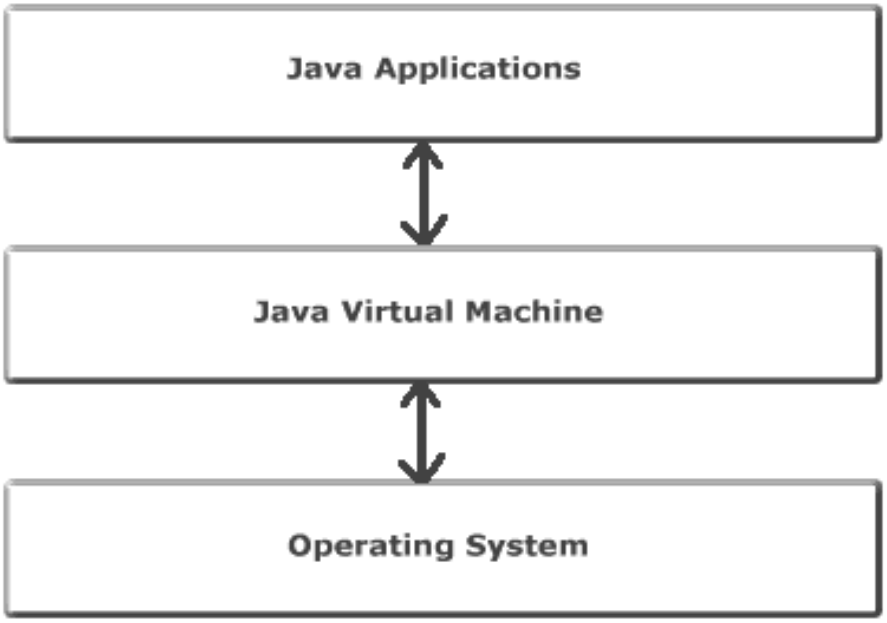


Figure 3.2: Java Component Structure

When a Java program is developed, predefined core class libraries written in the Java language is used. The JVM and core class libraries together provide a platform on which Java programmers can develop and be sure that any hardware and operating system that supports Java will execute their program because as said before, Java is platform independent. This is what makes Java so popular because once a program is implemented, it may run anywhere.

3.3 Oracle Database

The oracle database prefers Java. The only reason that it is allowed to write and load Java applications within the database is because it is a safe language. Languages like C can introduce security problems within the database. Thus, Java is because of its powerful features a safe language and therefore, it is used within the database.

Figure 3.3 shows that Oracle Java applications are on top of the Java core class libraries, which is on top of the JVM. Because the Oracle Java support system is located within the database, the JVM interacts with the Oracle database libraries, instead of directly with the operating system.

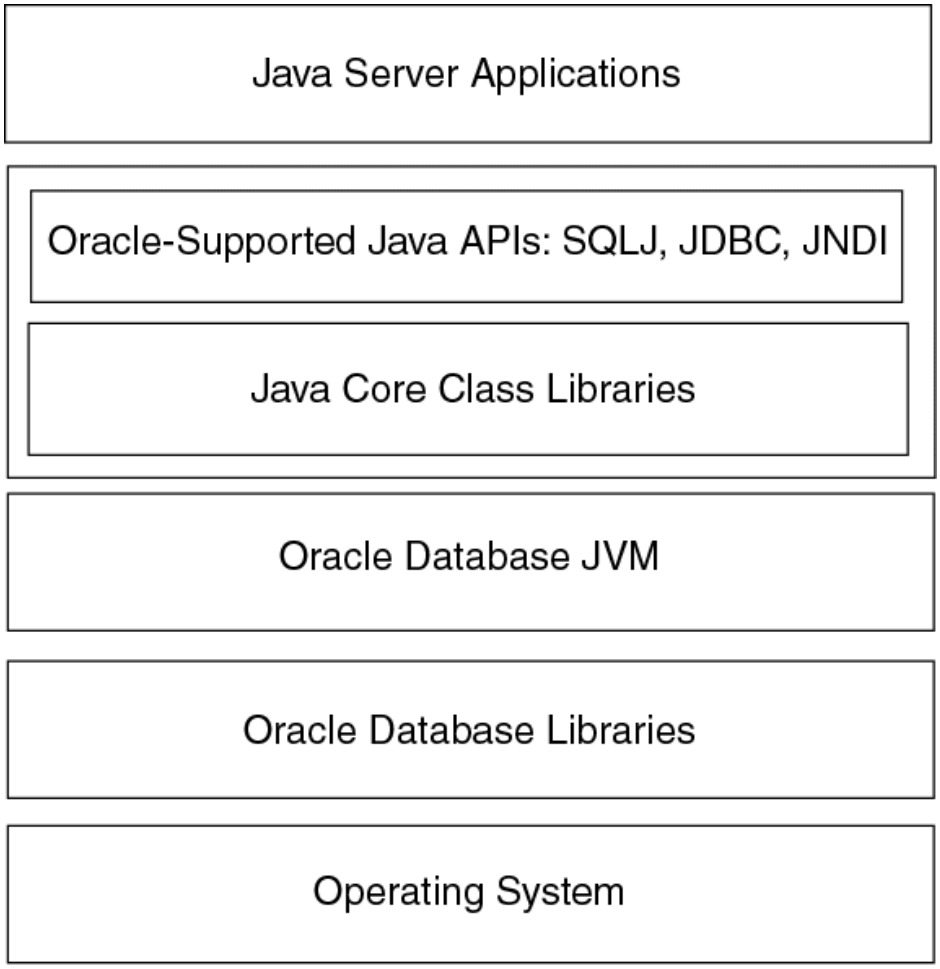


Figure 3.3: Oracle Database Java Component Structure

Chapter 4

Load Forecasting

4.1 Introduction

Precise models for load forecasting are important for designing, operating and planning a Load Management System. Load forecasting is needed in Load Management Systems to make the decisions correctly, including decisions on switching on and off the devices. Load forecasts are important for energy management projects like E-Island and other participants in electric energy management. There are three parts of load forecasting, short-term forecasting which are usually from one hour to one week, medium forecasting which are usually from a week to a year, and long-term forecasting which are longer than a year. The features of these forecasting types are different. For example, in some cases, it is possible to forecast the next day load with an accuracy of approximately fewer than 5 percent. While in some other cases, it is impossible to forecast the next year peak load with the similar accuracy, because precise long-term weather forecasts are not available.

Almost all forecasting methods use statistical techniques or artificial intelligence algorithms like regression, neural networks, fuzzy logic, and expert systems. Two of the methods, so-called end-use and econometric approach are broadly used for medium- and long-term forecasting. Lots of methods, like the so-called similar day approach, various regression models, time series, neural networks, statistical learning algorithms, fuzzy logic, and expert systems, have been developed for short-term forecasting.

None of these methods are said to be a general method because all forecasting methods have been tested and proven successful in lots of literature. Therefore, choosing a method depends on the project, circumstances and requirements of a particular situation.

4.2 Important Factors for Forecasts

For short-term load forecasting several factors must be taken into account, like time factors and historical load data. As compared to short-term load forecasting, "*the medium- and long-term forecasts take into account the historical load and weather data, the number of customers in different categories, the appliances in the area and their characteristics including age, the economic and demographic data and their forecasts, the appliance sales data, and other factors*" [26]. Since, in this research project, a typical day has to be calculated, this Master Thesis is based on the short-term load forecasting.

The time factors include the time of the year, the day of the week, and the hour of the day. There are important differences in load between weekdays and weekends. The load on different weekdays also can behave differently. For example, Mondays and Fridays could have different load values as compared to Tuesdays or Thursdays because Mondays and Fridays are adjacent to weekends. This case may be true during the summer time because of the holidays. Holidays are more difficult to forecast than non-holidays because of their relative infrequent occurrence.

The typical load curve of the normal year of the HAW Berliner Tor is shown in figure 4.1, with the sample interval of 15 minutes. The weekend or holiday load curve is lower than the weekday curve, due to the decrease of working load.

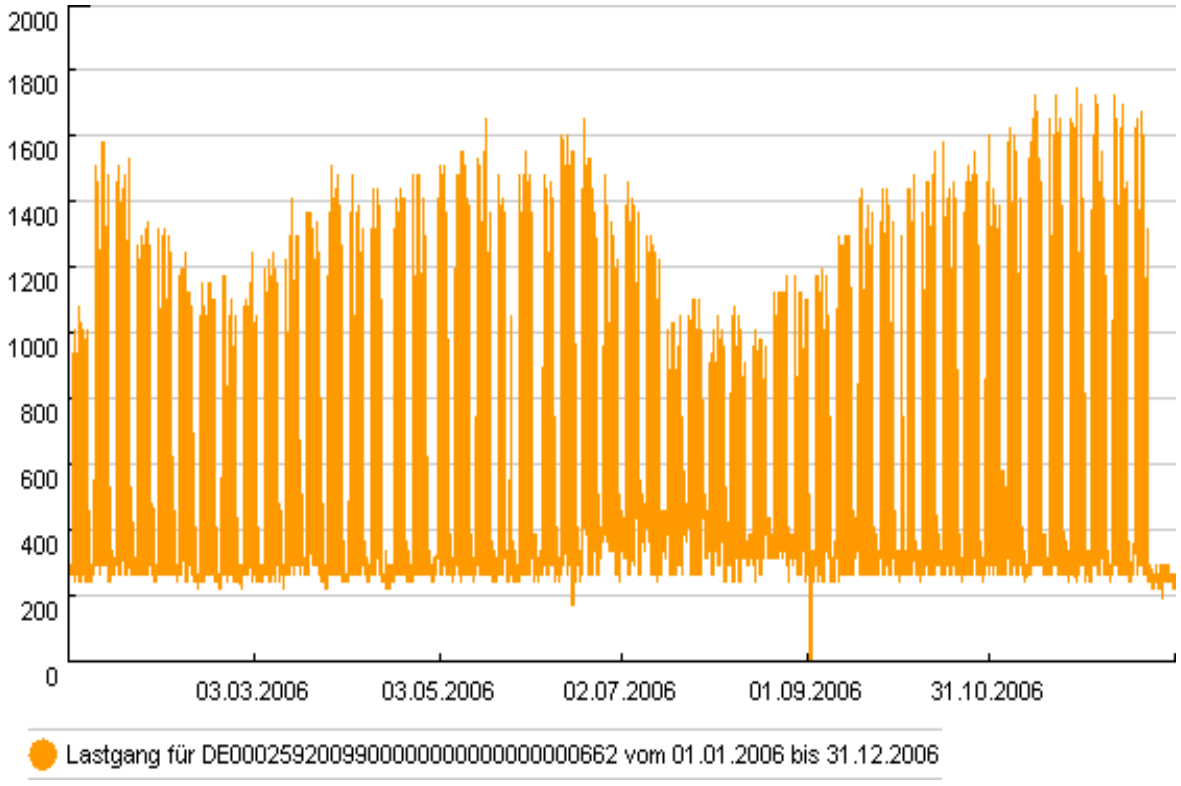


Figure 4.1: Typical load curve of the normal year of the HAW Berliner Tor

Factors influencing the load depend on the specific public property. For example, the industrial load is based on the level of the production. The load value is almost steady during the day and it is possible to forecast load curve of a public property being industrial. *"The issues which may disturb the forecasting, is the possibility of unexpected events, like machine breakdowns or workers strikes, which affects the load level"* [22]. In case of private people, the factors influencing the load forecasting are more difficult to select. The reason is that each person behaves in his own individual way. In case of forecasting the load curve of universities or schools may also become difficult in some point in time because of the school and university holidays. Thus, holidays also effect the load forecasting. The weather is also one of the important factors because if in summer the temperature drops down, the heating systems may be switched on.

Random factors are also one of the factors affecting the load forecasting. There is a way to overcome this factor. Although it may be difficult to forecast how each public property consumes the energy, the amount of the total sum loads of all public properties shows good historical data and it leads to smooth load curves. This is the first step of the load forecasting. But the startup and shutdown of the large loads, such as the wind tunnels in harbor in Hamburg, always lead to an obvious impulse to the load curve. This is a random disturbance, since for the dispatchers, the startup and shutdown time of these users is quite random, i.e. there is no obvious rule of when and how they get power from the grid. When the data from such a load curve are used in load forecasting, the curve of forecasted load may have some unwanted peaks.

Special events like the world cup football are also random disturbance because their effect on load is not quite certain. Lots of people may watch the match on a television, but that is not for sure. Strikes in companies are also another source of random disturbance.

4.3 Forecasting Methods

In terms of lead time, load forecasting is divided into four categories:

- Long-term forecasting with the lead time of more than one year
- Mid-term forecasting with the lead time of one week to one year
- Short-term load forecasting (STLF) with the lead time of one day to one week
- Very short-term load forecasting with the lead time shorter than one day

Different categories of forecasting serve for different purposes. In this master thesis short-term load forecasting is focused because in this project a calculation of a typical day is done.

The research approaches of short-term load forecasting can be mainly divided into two categories: statistical methods and artificial intelligence methods. In statistical methods, equations can be obtained showing the relationship between load and its relative factors after training the historical data, while artificial intelligence methods try to imitate human beings'

way of thinking and reasoning to get knowledge from the past experience and forecast the future load.

Some main STLF methods are introduced as follows.

Regression Methods

Regression is one of most widely used statistical techniques. For load forecasting regression methods are usually employed to model the relationship of load consumption and other factors such as weather, day type and customer class.

Time Series

Time series methods are based on the assumption that the data have an internal structure, such as autocorrelation, trend or seasonal variation. The methods detect and explore such a structure. Time series have been used for decades in such fields as economics, digital signal processing, as well as electric load forecasting. In particular, ARMA (autoregressive moving average), ARIMA (autoregressive integrated moving average) and ARIMAX (autoregressive integrated moving average with exogenous variables) are the most often used classical time series methods. ARMA models are usually used for stationary processes while ARIMA is an extension of ARMA to nonstationary processes. ARMA and ARIMA use the time and load as the only input parameters. Since load generally depends on the weather and time of the day, ARIMAX is the most natural tool for load forecasting among the classical time series models.

Similar Day Approach

This approach is based on searching historical data for days within one, two or three years with similar characteristics to the forecast day. Similar characteristics include weather, day of the week and the date. The load of a similar day is considered as a forecast. Instead of a single similar day load, the forecast can be a linear combination or regression procedure that can include several similar days. The trend coefficients can be used for similar days in the previous years.

Expert systems

Expert systems are heuristic models, which are usually able to take both quantitative and qualitative factors into account. A typical approach is to try to imitate the reasoning of a human operator. The idea is then to reduce the analogical thinking behind the intuitive forecasting to formal steps of logic. A possible method for a human expert to create the forecast is to search in history database for a day that corresponds to the target day with regard to the day type, social factors and weather factors. Then the load values of this similar day are taken as the basis for the forecast.

Fuzzy Logic

Fuzzy logic is a generalization of the usual Boolean logic used for digital circuit design. An input under Boolean logic takes on a value of “True” or “False”. Under fuzzy logic an input is associated with certain qualitative ranges. For instance the temperature of a day may be “low”, “medium” or “high”. Fuzzy logic allows one to logically deduce outputs from fuzzy

inputs. In this sense fuzzy logic is one of a number of techniques for mapping inputs to outputs. Among the advantages of the use of fuzzy logic are the absence of a need for a mathematical model mapping inputs to outputs and the absence of a need for precise inputs. With such generic conditioning rules, properly designed fuzzy logic systems can be very robust when used for forecasting.

Integration of Different Algorithms

As there are many presented methods for STLF, it is natural to combine the results of several methods. One simple way is to get the average value of them, which can lower the risk of individual unsatisfactory prediction. A more complicated and reasonable way is to get the weight coefficient of every forecasting method by reviewing the historical prediction results. The comprehensive result is deduced by weighted average method.

4.4 Forecasting Requirements

This subsection lists and describes the requirements to develop a user friendly and a good load forecasting tool. A good load forecasting tool should fulfill the requirement of accuracy, fast speed, friendly interface and automatic data access.

Accuracy

The most important requirement of designing a load forecasting tool is its prediction accuracy. As mentioned before, good accuracy is the basis of economic dispatch, system reliability and electricity markets. The main goal of this thesis is to make the forecasting result as accurate as possible.

Fast Speed

Employment of the latest historical data helps to increase the accuracy. When the deadline of the forecasted result is fixed, the longer the runtime of the forecasting program is, the earlier historical data can be employed by the program. Therefore the speed of the forecasting is a basic requirement of the forecasting program. Programs with too long training time should be abandoned and new techniques shortening the training time should be employed.

Friendly Interface

The graphical user interface of the load forecasting tool should be easy, convenient and practical. The users can easily define what they want to forecast, whether through graphics or tables. The output should also be with the graphical and numerical format, in order that the users can access it easily.

Automatic Data Access

The historical data are stored in the database. The load forecasting tool should be able to access it automatically and get the needed data.

4.5 Neural networks in load forecasting

Several Load forecasting methods have been discussed. Next, one of the most popular methods is discussed and used for load forecasting in this master thesis. Reason for using this method is that the implemented program should calculate a typical day. The method discussed up to now forecast the load curve of next day or next week and not the same weekday. The neural networks give fulfills this requirement. This subsection and the algorithm to forecast a typical day is taken from [21].

An Artificial Neural Network (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information. The key element of this paradigm is the novel structure of the information processing system. It is composed of a large number of highly interconnected processing elements (neurons) working in unison to solve specific problems. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons.

There are two ways of neural network, supervised or unsupervised learning. In supervised learning, the network is provided with example cases and desired responses. The network weights are then adapted in order to minimize the difference between network outputs and desired outputs. In unsupervised learning the network is given only input signals, and the network weights change through a predefined mechanism, which usually groups the data into clusters of similar data.

The most used network type which uses supervised learning is a feed-forward network. The network is given an input signal, which is transferred forward through the network. In the end, an output signal is produced. The network can be understood as a mapping from the input space to the output space. The most popular of all neural networks is the Multi-Layer Perceptron network (MLP).

Most of the load forecasting approaches are based on the use of an MLP network.

4.5.1 Multi-Layer Perceptron network (MLP)

Multi-Layer Perceptron network is the frequently used neural network type and almost every neural network load forecasting models are based on it. The basic unit (neuron) of the network is a perceptron which is a computation unit, and which produces its output by taking a linear combination of the input signals and by transforming this by a function called activity function. The following function shows that the output of the perceptron as a function of the input signals:

$$y = \sigma(\sum_i^n w_i x_i - \theta),$$

where

y is the output
xi are the input signals
wi are the neuron weights
θ is the bias term (another neuron weight)
σ is the activity function

The MLP network is made up of a number of layers of neurons. As the following figure shows, every neuron in a specific layer is connected to every neuron of the next layer. There are no feedback connections. The figure 4.2 shows an example of a three-layer MLP network.

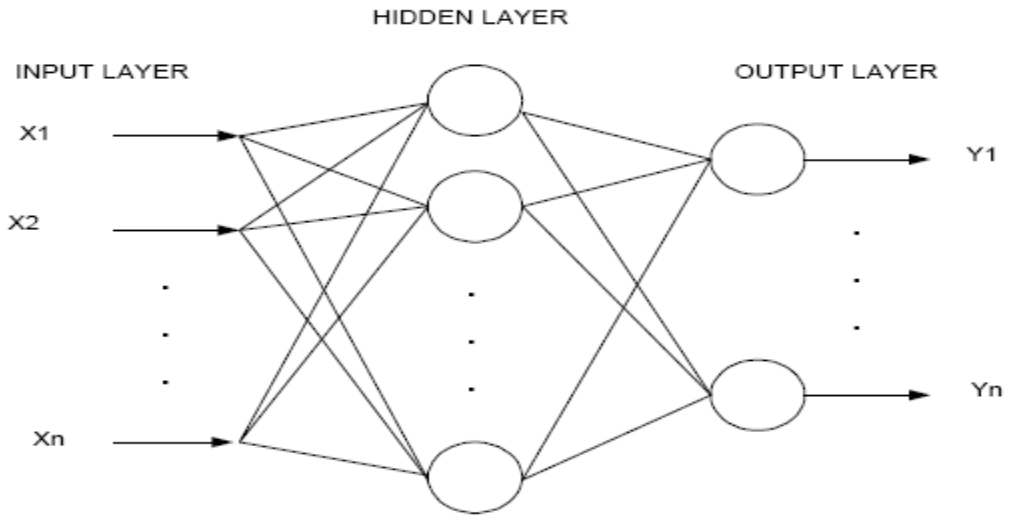


Figure 4.2: A three-layer MLP network

Thus, in general case, the function looks as follows:

$$\mathbf{y} = f(\mathbf{x}; \mathbf{W}) = \sigma(\mathbf{W}_n \sigma(\mathbf{W}_{n-1} \sigma(..\sigma(\mathbf{W}_1 \mathbf{x})...))),$$

where

y is the output vector
x is the input vector
W*i* is a matrix containing the neuron weights of the *i*:th hidden layer. The neuron weights are considered as free parameters.

If the network is fed by an *N*-dimensional input vector, then the output has an *M*-dimensional output vector. Thus, the network may be described as a function from the *N*-dimensional input space to the *M*-dimensional output space.

Why the MLP models are used in load forecasting, may be explained as follows: The forecasted values dependents on historical load data, and the MLP network is used to estimate this dependency. The inputs to the network consist of those historical load values, and the output is the forecasted load values for a typical day.

4.5.2 Basic MLP-models

Load forecasting with MLP neural network models may have three following types. These different model types are intended for:

- forecasting daily peak-, valley- or total load
- forecasting the whole daily load curve at one time
- forecasting the load of the next hour

The first two types are static. The calculation is based on historical data. The third type is dynamic because the forecast can be updated every time new data arrives. This master thesis makes use of static models, i.e. it uses the first two models.

Chapter 5

Forecasting the daily load profile

The idea in using the MLP neural is to identify the assumed dependency of the daily peak-, valley-, and average load on earlier load. In the figure below the seasonal trend on the load can be easily seen. Also, the weekly load structure can be seen in the form of lower load values on weekends than on working days.

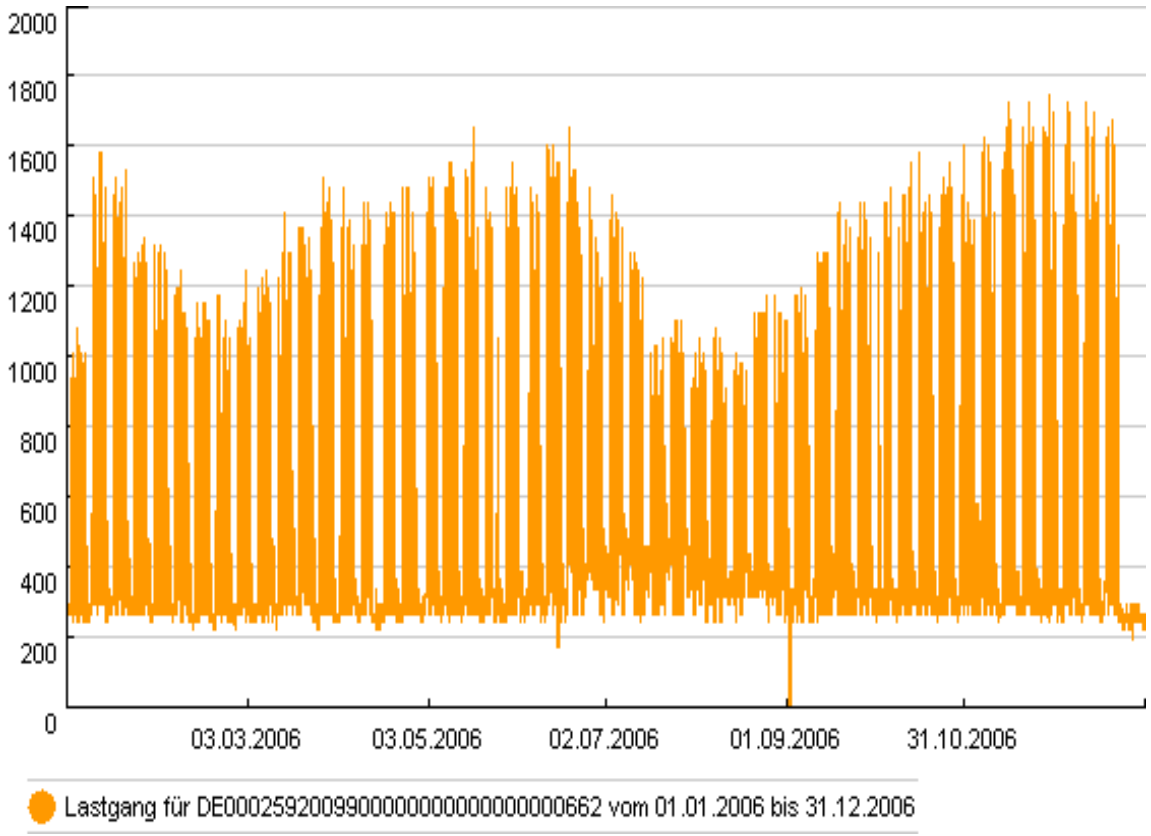


Figure 5.1: Typical load curve of the normal year of the HAW Berliner Tor

To forecast the load curve of a certain day, at least the maximum loads, minimum loads and average loads of the similar day should be considered as input variables. In addition, informing the program about the day type is important, because Saturdays and Sundays have much lower peak loads than working days.

Classifying the days

The load shape is predicted by averaging some load curves of similar days in the load history, first. Therefore, days have to be grouped into classes of different day types, i.e. Mondays, Tuesdays, Wednesdays, etc..

The load shape based on peak- and valley loads

Second step is to normalize the averaged (first step) load values which are used as factors for the calculation of a typical day. The shape of the load curve for a certain day contains 1440 normalized load values (24 h * 60 min). When the load shape is combined with the averaged maximum and minimum load values of similar days, the normalized load values are [21]:

$$L_{nor}(i,j) = \frac{L(i,j) - L_{\min}(i)}{L_{\max}(i) - L_{\min}(i)} \ ,$$

where

$L_{nor}(i,j)$ = normalized load at hour j on day i

$L(i,j)$ = load at hour j on day i

$L_{\min}(i)$ = valley load of day i

$L_{\max}(i)$ = peak load of day i

When the load shape has been predicted, the minutely load forecast can be calculated as third step [21]:

$$\hat{L}(i,j) = \left[\hat{L}_{\max}(i) - \hat{L}_{\min}(i)\right]\hat{L}_{nor}(i,j) + \hat{L}_{\min}(i),$$

where the hats indicate that the load values are forecasts.

The load shape based on average load

When the load shape is combined with the average load, the normalized load values are [21]:

$$L_{nor}(i,j) = L(i,j) - L_{ave}(i),$$

where

$L_{ave}(i)$ = average load of day i

The forecast for the hourly load is then:

$$\hat{L}(i,j) = \hat{L}_{nor}(i,j) + \hat{L}_{ave}(i).$$

Different averaging models

The forecast for the load shape is obtained by averaging some load shapes of the corresponding day type class in the load history. The number of these days has to be decided. Four different models are considered for the classification given on the last page. These are:

Model 1: For all day types all similar days in spring are averaged.

Model 2: For all day types all similar days in summer are averaged.

Model 3: For all day types all similar days in autumn are averaged.

Model 4: For all day types all similar days in winter are averaged.

Chapter 6

Using Unified Modeling Language for Load Forecasting

The Unified Modeling Language (UML) is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems.[22] The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects oriented software and the software development process itself. The UML uses mostly graphical notations to express the design of software projects. Using the UML helps project teams communicate, explore potential designs, and validate the architectural design of the software [22].

There are three types of UML Diagrams which are functional requirement view, static structural view and dynamic behavior view. The functional requirement view shows the interaction of the user and the system without taking the technical requirement into consideration. This type of diagram is called the Use Case Diagram. The static structural view represents the design part of a project. In object oriented programming, it is called the Class Diagram. It shows which methods and attributes are implemented inside each of the classes and how they are connected to each other. The dynamic behavior view shows the program interaction in runtime. The so called sequence diagram illustrates the dynamic behavior of each object taking place in runtime.

6.1 Use Case Diagram (functional requirements view)

The following figure shows the use case scenario of the load forecasting tool. It shows the idea how to implement the code. The user selects some public properties, a weekday, a season and a holiday restriction. Now, the user clicks on a start button which starts the program to

generate a typical day. For that the program gets the historical data and calculates the typical day. Finally, it saves the load values of that day in a CSV file.

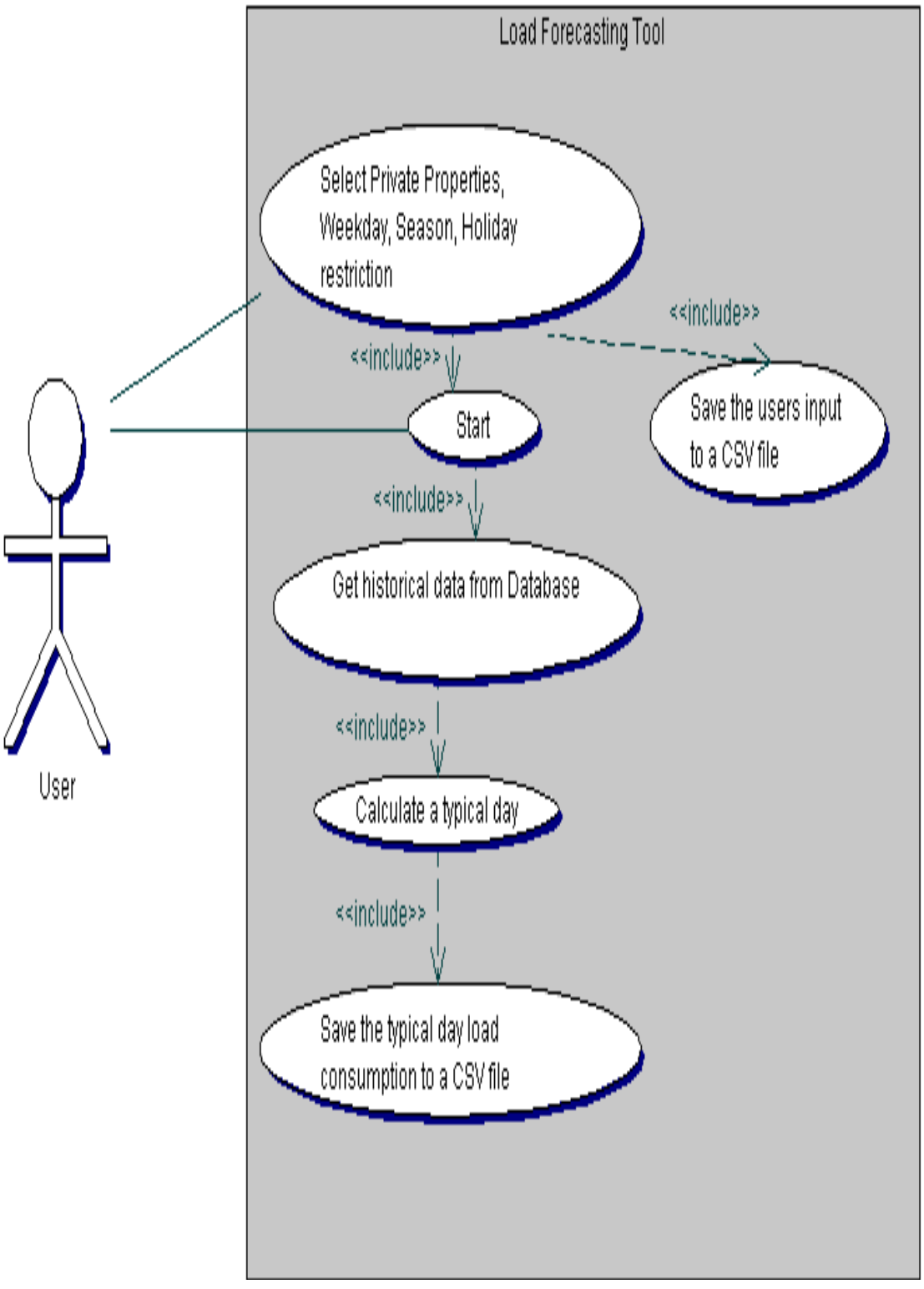


Figure 6.1: Use Case Diagram of Load Forecasting a typical day

6.2 Class Diagram (static structural view)

Following is the class diagram of this project, i.e. load forecasting of a typical day using object oriented programming. It shows that four classes are needed to realize this project. Each class has its own responsibility. JDBCQuery is responsible for database query. HAWCalc is responsible for creating the GUI and doing the calculations. WorkdayCalendar is responsible calculating the German national holidays. And CreateFile is responsible for generating CSV files. The classes are discussed in more details in chapter 8.

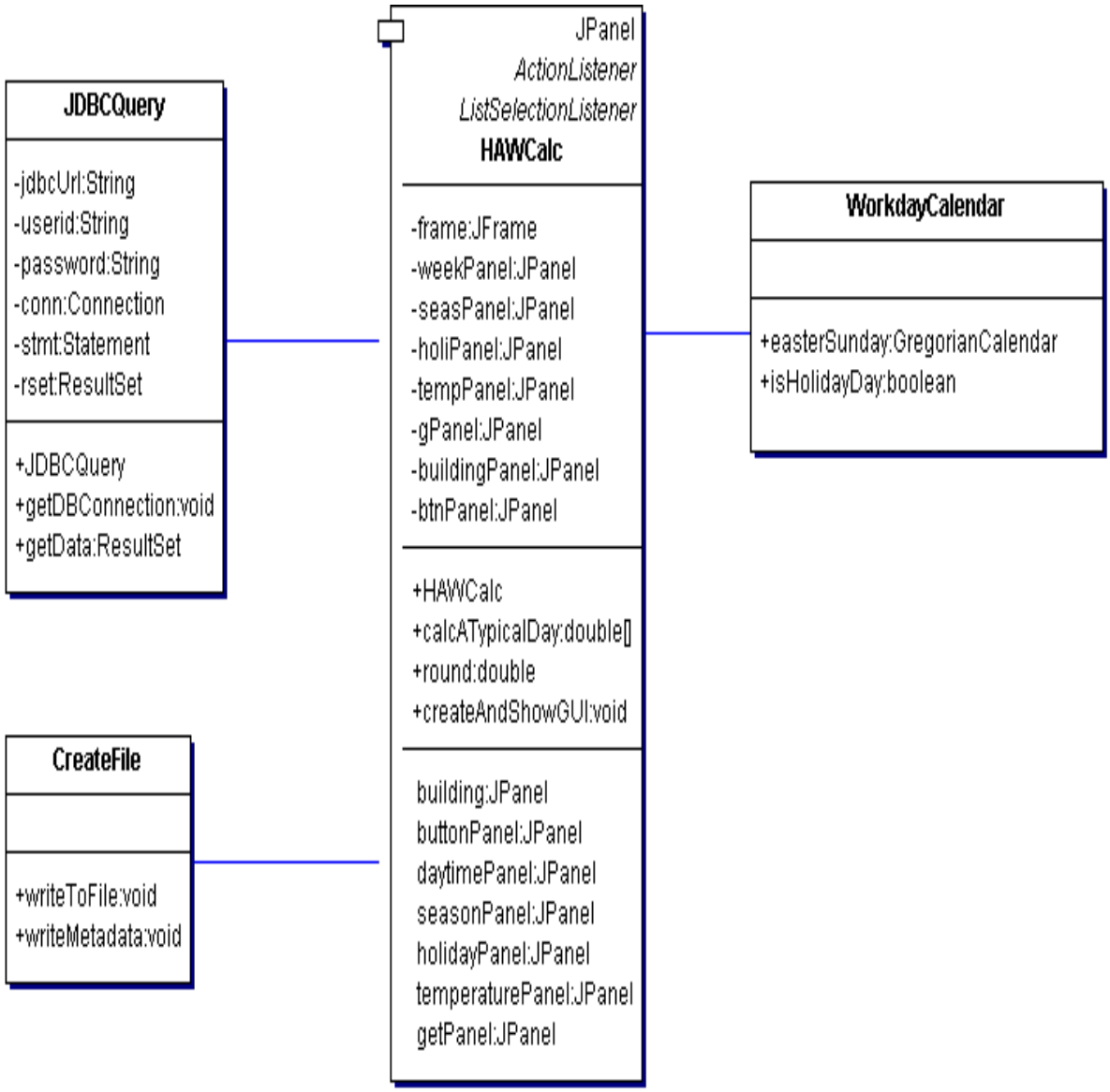


Figure 6.2: Class Diagram of Load Forecasting a typical day

6.3 Interaction Diagram (dynamic behavior view)

The sequence diagram in figure 6.3 shows how the classes shown in figure 6.2 interact with each other. First, a GUI is generated in HAWCalc. Second, the user's selection takes place. Third, the HAWCalc interacts with JDBCQuery to get the historical data from the database and it calculates the typical day where the holiday check is done first by interacting with WorkdayCalendar. At last step, HAWCalc sends the data to CreateFile to generate a CSV file containing the load values.

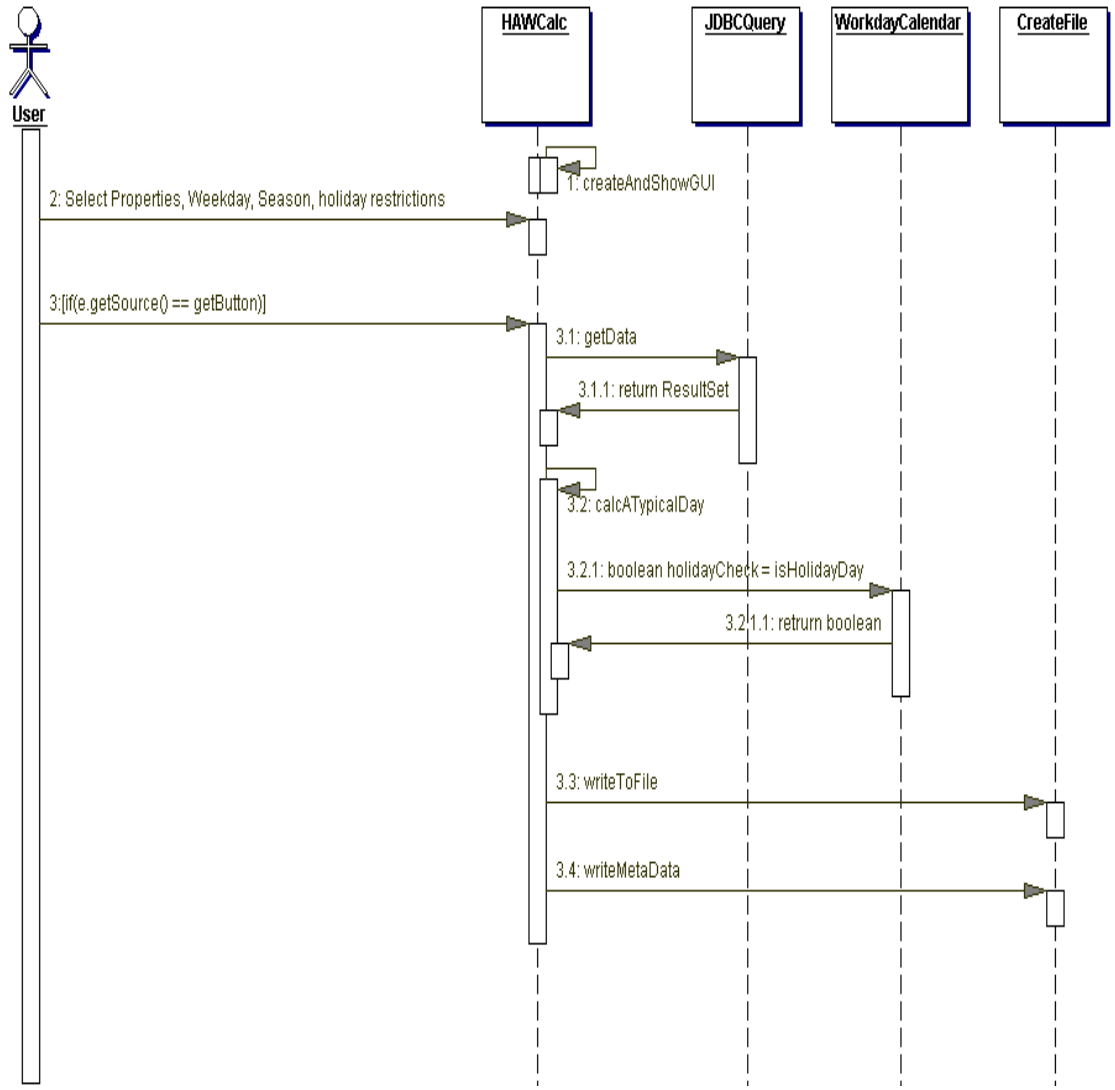


Figure 6.3: Sequence Diagram of Load Forecasting a typical day

Chapter 7

Historical Data

At this point in time, the load data of about 40 public properties exist on the database at the company Envidatec. Each of the public properties has an ID, consumed load value and the timestamp when the consumed load value was read. The database is built up as follows:

Table "**DATA**" contains data samples.

Field name	Type	Description
ID	NUMBER	Data entry ID
TIMESTAMP	DATE	Time this entry corresponds to. For logged data it is the time when the value was received from the sensor.
VALUE	NUMBER	Load Value
DATA_ROW_ID	NUMBER	Identifier of the data row this entry belongs to (link into the DATA_ROWS table)

Table 7.1: Database Table “DATA”

Table **DATA_ROWS** contains the names of the public properties with their ID's. It is linked to the table "DATA"

Field name	**Type**	**Description**
ID	NUMBER	Data row ID
DESCRIPTION	VARCHAR2(256)	Textual descriptor of data row

Table 7.2: Database Table “DATA_ROWS”

In both tables, the field "ID" is the primary key for the respective table. A primary key should have the following properties:

1. It must be constant over the lifetime of an object (row).
2. It must be unique for the lifetime of the application.
3. Its value must not be reused.
4. It must have sufficiently large value domain for the lifetime of the application.
5. Its domain of definition must be completely used.
6. It must be minimal to satisfy the other requirements.
7. It should be constructed by computer.
8. It may be secured by check digit (transparently for the user).

In the database historical data exists from year 2004 till date. From year 2008 onwards, the consumed load values are going to be read at a rate of one sample per minute. Till the end of the year 2007, the historical data was recorded at one sample per fifteen minutes. For the project this data has to be changed into load profiles with a resolution of one sample per minute since the load management system will always try to optimize the load of one polling period which in the UTCE grid on quarter of an hour (15 minutes). First idea was to develop Java program which gets all the load values of each building and then creates random noise data for every minute between two polling periods values such that the average value of 15 random values equals the polling periods value fetched from the database. Figure 7.1 shows graphically that the average value of 15 random values is the measured load value fetched from the database.

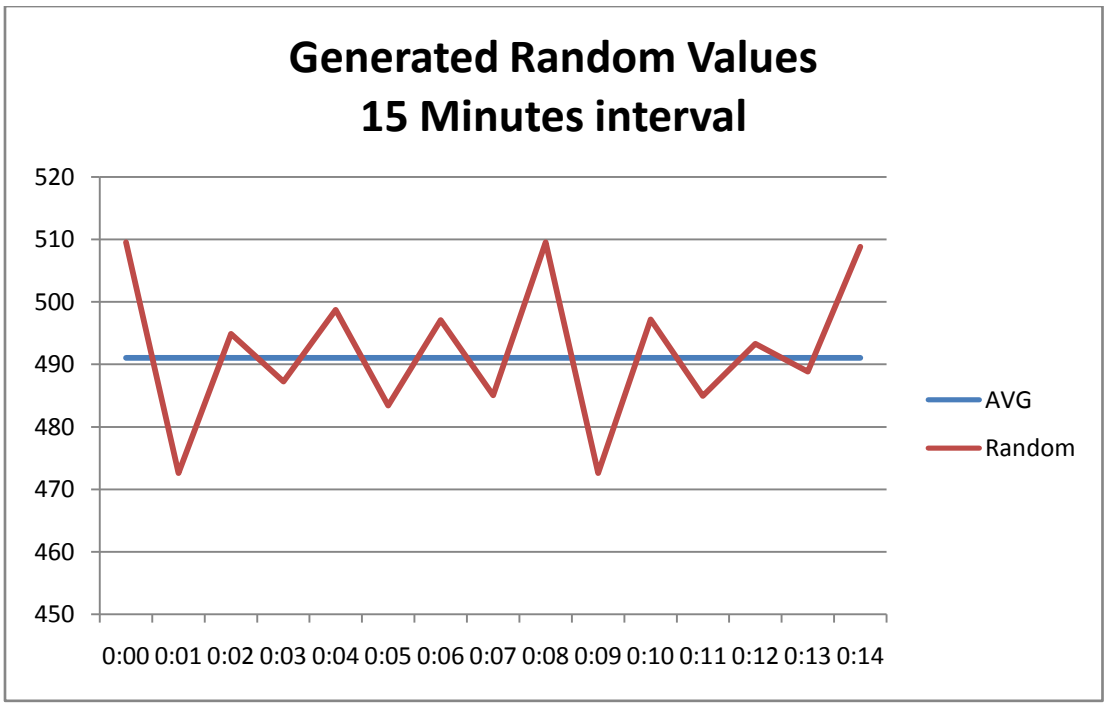


Figure 7.1: Randomly Generated Values

With respect to figure 7.1, figure 7.2 shows the curve after being randomized for an interval of two hours. If the figure is observed very carefully, it may be seen that in each next 15 Minute, there is a jump. Thus, this idea may not be acceptable.

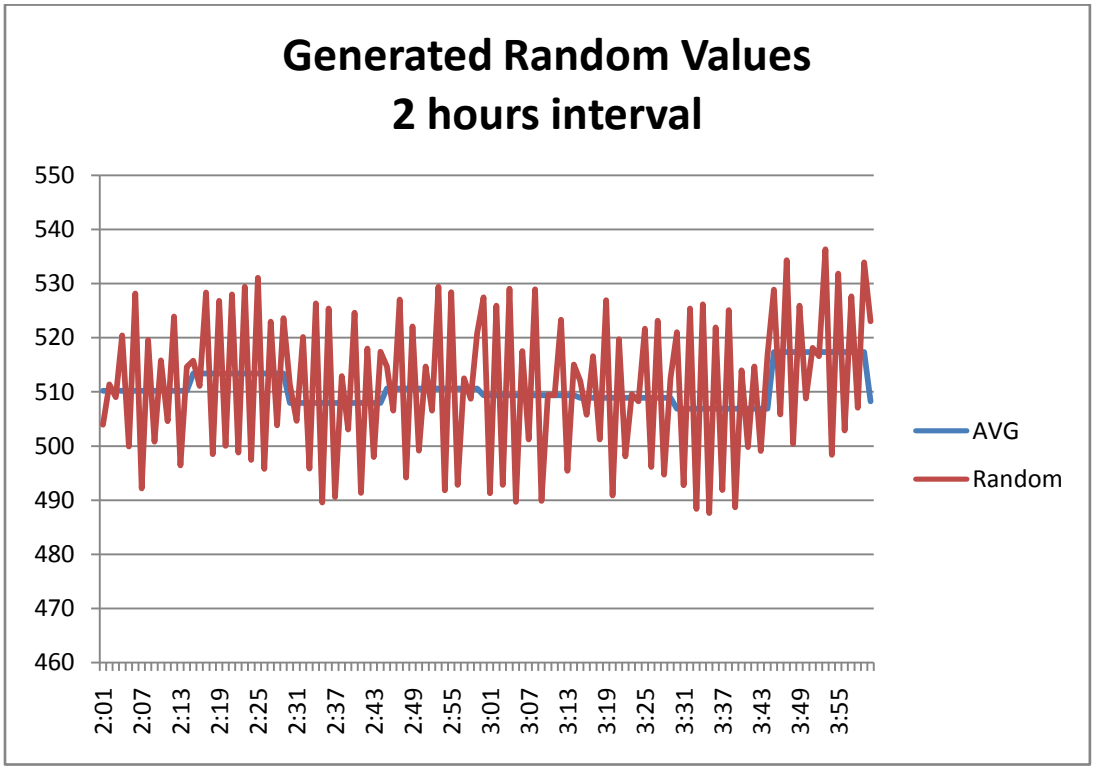


Figure 7.2: Randomly Generated Values (2 hours interval)

To avoid these jumps, an advanced algorithm has been developed using Java. The algorithm works as follows. First, it gets the load values from the database in one sample per fifteen minute which means a day contains 96 load values. Second, the difference between all two fifteen minute load values is calculated. Third, these differences are divided by 15 which will give the difference between the minute values. Using these differences the each fifteen minute value is converted to one minute values, respectively, which mean a day contains 1440 load values now. At the end, these 1440 load values are randomized by multiplying those values with a random factor (like the initial idea) being between plus minus ten percent. Figure 7.3 shows that this algorithm performs very well and there is no huge jump between the values. Finally, the original 15 minutes values are deleted from the database and the database is updated with the newly calculated values, i.e. one sample per minute. Figure 7.4 is a flowchart which shows how this Java program works.

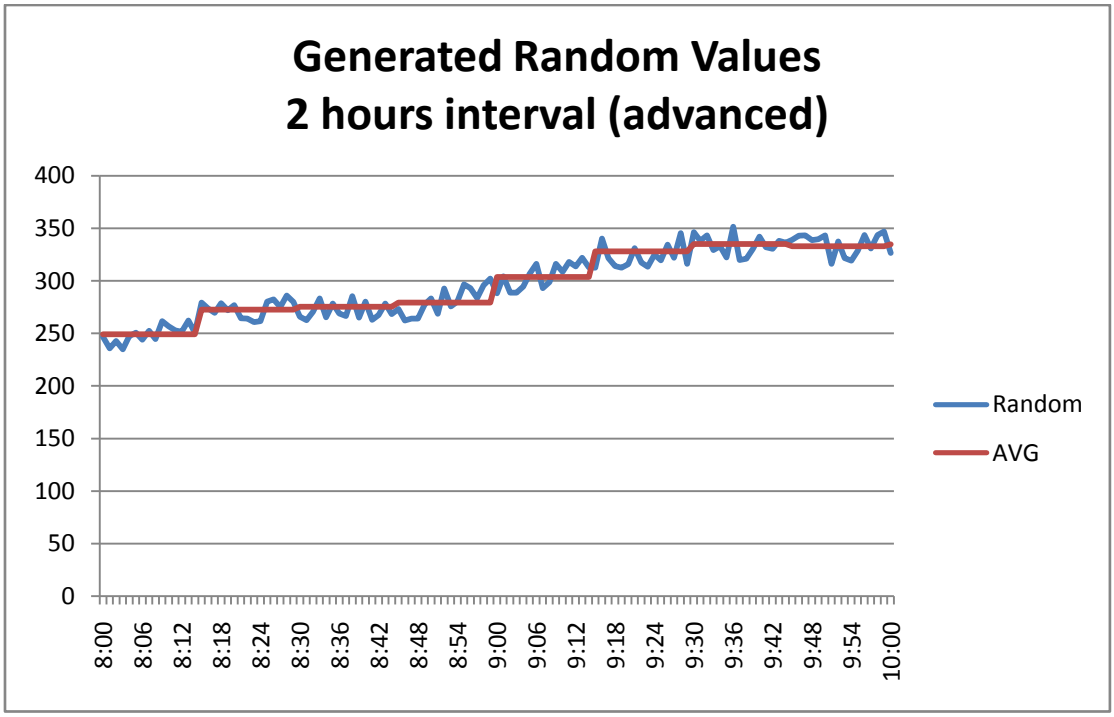


Figure 7.3: Randomly Generated Values (advanced)

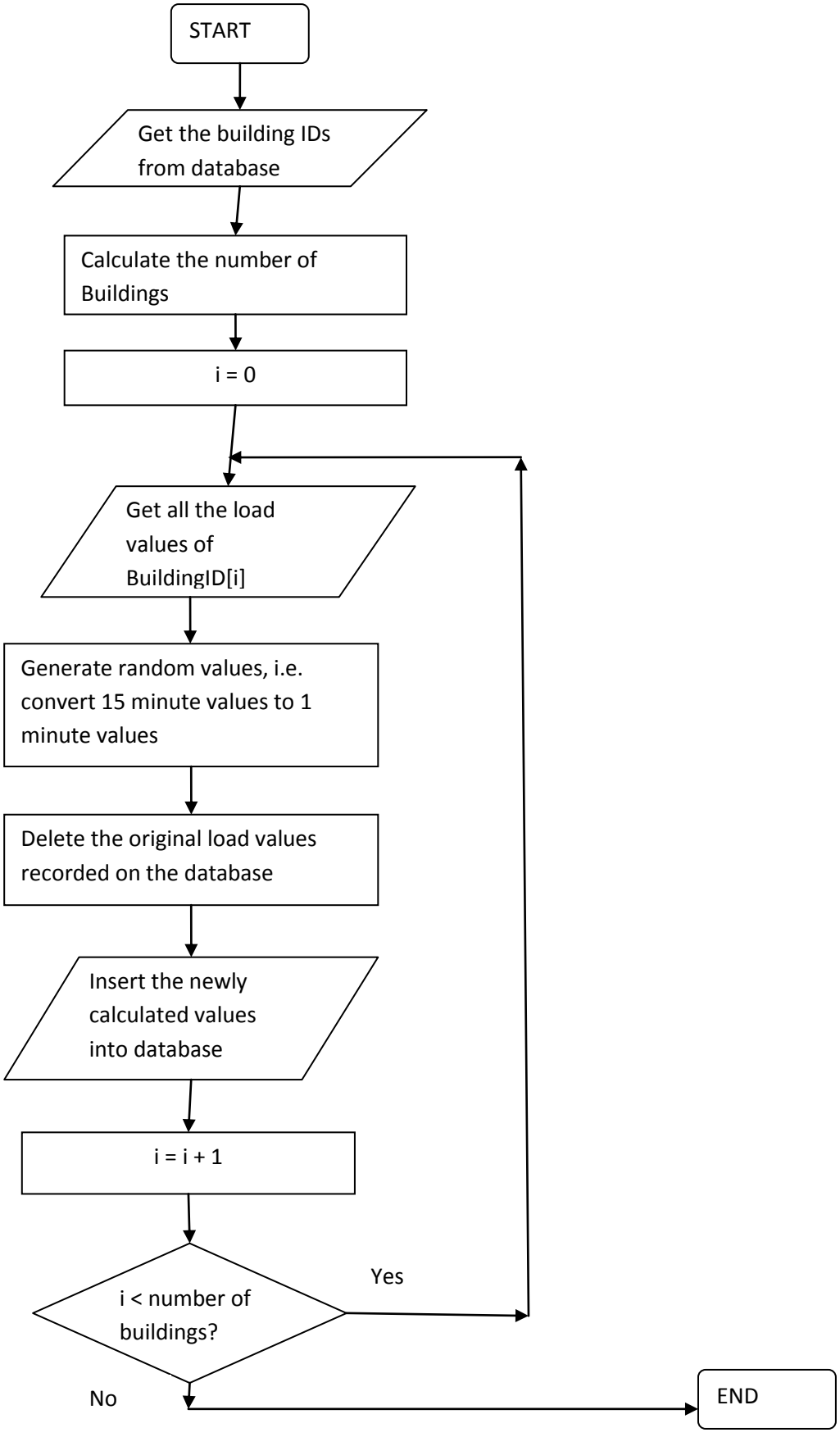


Figure 7.4: Flowchart of recalculation of historical data

Chapter 8

Load Forecasting in Java

This program forecasts a typical day (e.g. typical Wednesday) using the users given specification through the user friendly graphical user interface (GUI). In order to forecast as accurate as possible, it uses the historical data which is placed on the database. After calculating a typical day for the given specification, the program stores the load values for each public property to a CSV file (Comma Separated Value file). In addition to that, it also creates an additional column where the load values of all the properties selected are summed up for each minute (sum load profile).

8.1 Implemented Classes and Methods

In addition to the default libraries in Java which are needed for the graphical user interface, the oracle library "`import oracle.jdbc.*`" has also been imported. For file reading and writing the library "`import java.io.*`" is also needed.

Since in any Java program the file name has to be identical to the class name, it was decided to call the program HAWCalc since the class name is also called HAWCalc (public class HAWCalc extends JPanel implements ActionListener). This class extends "JPanel" because the program creates a graphical user interface. This class also implements ActionListener. ActionListener is the listener interface for receiving action events. The class that is interested in processing an action event implements this interface. The object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

This class is the main class and all the objects, methods and constructors needed are called in this class. In addition to HAWCalc, there are three other classes:

- JDBCQuery
- HolidayCheck
- CreateFile.

8.2 The class HAWCalc

8.2.1 Constructor

The constructor is responsible for creating the graphical user interface. First, it sets the layout of the graphical user interface. Second, it calls the following methods one by one and fits them on the layout:

- `getBuilding();`
  - It gets all the available public properties from the database and puts them in a list. The list is added to the graphical user interface. To establish a connection to the database and get the public properties by their names, an object of JDBCQuery is created.
- `getDaytimePanel();`
  - It creates 7 radio buttons which are initialized with the weekdays (Monday, Tuesday, Wednesday, Thursday, Friday, Saturday and Sunday). As default, "Monday" is selected.
- `getSeasonPanel();`
  - It creates 4 radio buttons which are initialized with the seasons (Spring, Summer, Winter and Autumn). As default, "Spring" is selected.
- `getHolidayPanel();`
  - It creates 3 radio buttons which are initialized with "Include Holidays", "Exclude Holidays" and "Only Holidays". As default, "Exclude Holidays" is selected.
- `getTemperaturePanel();`
  - It creates a text field where the user may input a temperature. It also implements an input verifier which checks whether the input is digit or not.
  - This function will be used in future
- `getGetPanel();`
  - It creates a "Start" button which is clicked after giving the specifications.

8.2.2 The method „public static void main (String args[])”

The “main()” method is needed in Java to run the application. The “main()” is responsible for creating and showing the graphical user interface. Thus, a method called “private static void createAndShowGUI()“ is called in “main()”. The method “createAndShowGUI()” creates the GUI and shows it. For thread safety, this method should be invoked from the event-dispatching thread. This program makes use of threads because the user does selections on a GUI and these selections are handled from the code in parallel. In order to fulfill the safety requirements the method “createAndShowGUI()” is called as follows:

```
public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
```

8.2.3 The method “private static void createAndShowGUI()”

As the name states, this method creates and shows the GUI. First, an object of type “HAWCalc” is created and initialized. Since a graphical interface has to be presented to the user, a “JFrame” Window was selected. An object of type JFrame is initialized and the object of HAWCalc is set as content pane.

After this method is called in “main()” and code is executed as an application, the following output is generated:

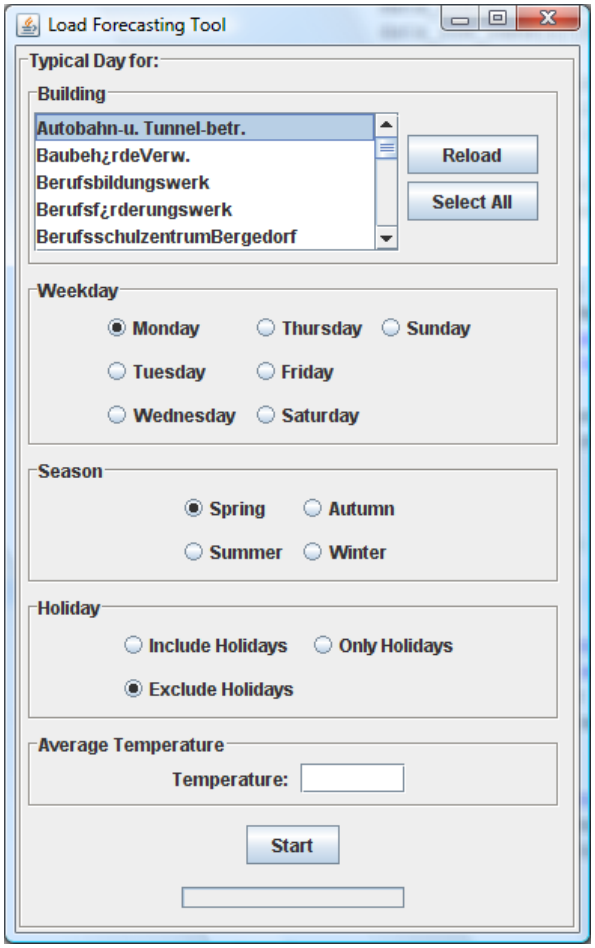


Figure 8.1: Graphical User Interface for forecasting a typical day

8.2.4 The method "public void actionPerformed(ActionEvent e)"

This method defines what should happen if the user clicks on the GUI elements. The user may choose one-to-many buildings, a weekday, a season and holiday restriction. As default weekday, Monday is chosen. On click, this method changes the variable "day" as follows:

Day of week	Constant field value
SUNDAY	1
MONDAY	2
TUESDAY	3
WEDNESDAY	4
THURSDAY	5
FRIDAY	6
SATURDAY	7

Table 8.1: Constant field values for days of week

By switching between the seasons, the following table shows how this method changes the variables "season", "startMonth" and "endMonth". If the user does not click on anything, spring is taken as default season. For each season, the start day is set to 21 and the end day is set to 20, e.g. spring is between 21.03 and 20.06.

Season	Constant field value	startMonth	endMonth
Spring	1	3	6
Summer	2	6	9
Autumn	3	9	12
Winter	4	12	3

Table 8.2: Constant field values for Seasons of year

The variable “holiday” changes as follows, when the user switches between the holiday conditions

Holiday condition	Constant field value
Include Holidays	1
Exclude Holidays	2
Only Holidays	3

Table 8.3: Constant field values for Holiday conditions

The five variables “day”, “season”, “startMonth”, “endMonth” and “holiday” are needed to query the database and get the needed values.

The most important event in this method is the implemented functionality behind the “Start” button. For each selected building, it creates an object of “JDBCQuery” (see chapter 8.3) which does a database query and gets only that data which had a database entry on the user selected season. For that, it needs the variables “startMonth” and “endMonth”.

After the data is fetched from the database, it calculates a typical day for the user selected weekday. For that, it calls the method “private double[] calcATypicalDay(ResultSet rset)“ (see chapter 8.2.5). The object rset is a type of ResultSet on which the fetched data is stored. The method “calcATypicalDay(ResultSet rset)” returns the calculated values back. An object of “CreateFile” (see chapter 8.5) is generated and the calculated load values are stored in a CSV file. In addition to this CSV file, a second CSV file is generated which includes information about the user’s input, i.e. it includes the constant field values for the variables “day”, “season” and “holiday”.

The user gets the following message if the program has calculated a typical day and successfully generated the CSV files:

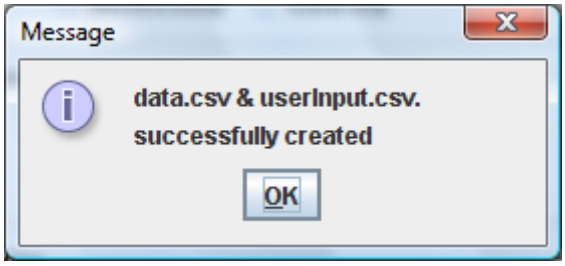


Figure 8.2: User Information Message: CSV files created

8.2.5 The method "private double[] calcATypicalDay(ResultSet rset)"

This method gets as input the result set fetched from the database. From the result set it filters out all the load data which are measured on the weekday given by the user, e.g. if the user selects Monday, then all the Mondays in that specific season will be taken for further calculation. Secondly, it checks the holiday restriction the user has selected. An object of the type "WorkdayCalendar" (see chapter 8.4) is created which checks each given weekday (e.g. Monday) if it is a holiday or a non holiday. According to user's input it filters the load data again.

Now, the method begins to calculate a typical day for the given weekday using the following algorithm (see chapter 5):

$$L_{nor}(i,j) = \frac{L(i,j) - L_{\min}(i)}{L_{\max}(i) - L_{\min}(i)} ,$$

where

$L_{nor}(i,j)$ = normalized load at hour j on day i

$L(i,j)$ = load at hour j on day i

$L_{\min}(i)$ = valley load of day i

$L_{\max}(i)$ = peak load of day i

When the load shape has been predicted, the hourly load forecast can be calculated:

$$\hat{L}(i,j) = \left[\hat{L}_{\max}(i) - \hat{L}_{\min}(i)\right]\hat{L}_{nor}(i,j) + \hat{L}_{\min}(i) ,$$

where the hats indicate that the load values are forecasts.

After the calculation has been finished, it returns an array of calculated load values.

8.3 The class JDBCQuery

This class is responsible for establishing a connection to the database und doing some SQL query. This class is developed such that, it may be used not only to connect to the server placed at the company Envidatec, but to connect to any database if the user ID, the password and the URL of the database is known.

The constructor reads a text file where the user ID, the password and the URL of the database is written. Using this information a connection to the database is established and it waits for the SQL query. The database is queried using the user's input and the fetched result is stored on a result set.

8.4 The class WorkdayCalendar

This class checks if a date is a holiday. The calculation of holidays for all years is done as follows. Since most of the German holidays depend on the calendar position of the Easter holiday, first the Easter Sunday is calculated. There are a lot of algorithms for calculating the Easter holidays. The most popular algorithm is from Carl Friedrich Gauß, Mallen and Oudin. In this project, Oudin's algorithm is used.

```
public static GregorianCalendar easterSunday( int year )
         {
          int i = year % 19;
          int j = year / 100;
          int k = year % 100;

          int l = (19 * i + j - (j / 4) - ((j - ((j + 8) / 25) + 1) / 3) +
15) % 30;
          int m = (32 + 2 * (j % 4) + 2 * (k / 4) - l - (k % 4)) % 7;
          int n = l + m - 7 * ((i + 11 * l + 22 * m) / 451) + 114;

          int month = n / 31;
          int day   = (n % 31) + 1;


          return  new GregorianCalendar( year, month - 1, day );
        }
```

German holidays like New Year, 1. May, Christmas and New Year's Eve are obvious, i.e. the date never changes.

A date is given to the method "public boolean isHolidayDay(int year, int month, int day)" of this class and it makes a holiday check and returns a true for a holiday and false for a non holiday.

8.5 The class CreateFile

This class is a file writer class. It has two methods. Both are responsible for generating a CSV file. One of the method of this class get the newly calculated load values as input and it generates a CSV file and the one gets the user's selection on the GUI as input and writes it to a file.

8.6 Program flow

The program starts by double clicking the JAR file. A JAR file (or Java ARchive) is used for aggregating many files into one. It is generally used to distribute Java classes and associated metadata. JAR files are not just simple archives of java classes' files or resources. They are used as building blocks for applications and extensions. Since this program is a Java application, the Java interpreter executes the "main(String args[])" method at first. Here, the method "createAndShowGUI()" is called to popup the graphical user interface. In order to get the building name and list them on the graphical user interface, at first, the file "Config.txt" is called. This file includes the user ID, the password and the URL of the server. Afterwards, all the other elements (weekdays, seasons and holiday restrictions) of the graphical user interface are generated.

Now, the user is able to generate a typical day by selecting the elements from the graphical user interface. The user may select one, more than one or all buildings. By clicking the "Select all" button all the buildings will be selected. Respectively, if the user clicks the "Reload" button, the buildings are unselected and graphical user interface is brought to its initialized position. It is only possible to select one weekday, season and holiday restriction. After the "Start" button is clicked, the program starts to generate a typical day. For that, it reads the "Config.txt" file again to connect to the server and reads the historical data needed. Using the algorithm given in chapter 5, it forecasts a typical load profile from all the historical data for the selected type of day.

At the end, two CSV files are generated:

- data.csv
    - predicted load values are stored in one sample per minute (1440 Values)
- userInput.csv
    - Input elements of the users are stored

The program is able to give messages to the user such as error messages. If the user tries to query a database and the database has no data for that selection, the following error message pops up.

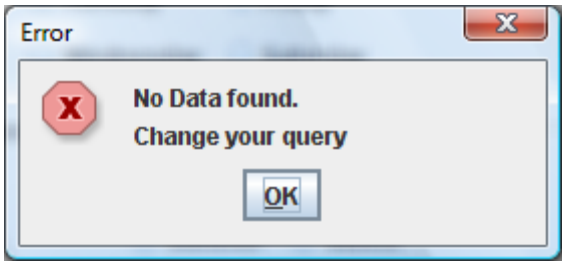


Figure 8.3: User Error Message: No Data found

The file can only be overwritten if no other program is accessing the file, e.g. the data.csv has been opened with MS Excel, but has not been closed yet. In that case, the Java application cannot access that file to overwrite it and the following error message pops up.

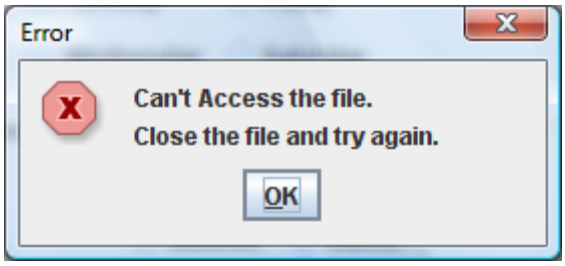


Figure 8.4: User Error Message: Can’t Access the file

This program was developed to run with databases which have the table entry given in chapter 7. As already discussed, the program gets the connection information from the file called “Config.txt”. If the information inside that file is not correct, the following error message will be displayed.

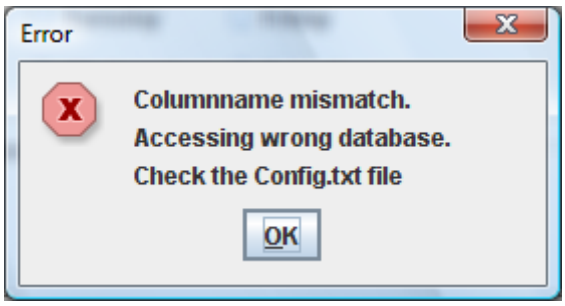


Figure 8.5: User Error Message: Wrong database

If the program gets the data from the database, is able to do the calculation and writes the result to the CSV files, the following information message is displayed.

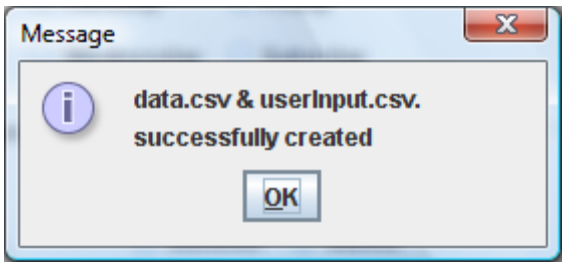
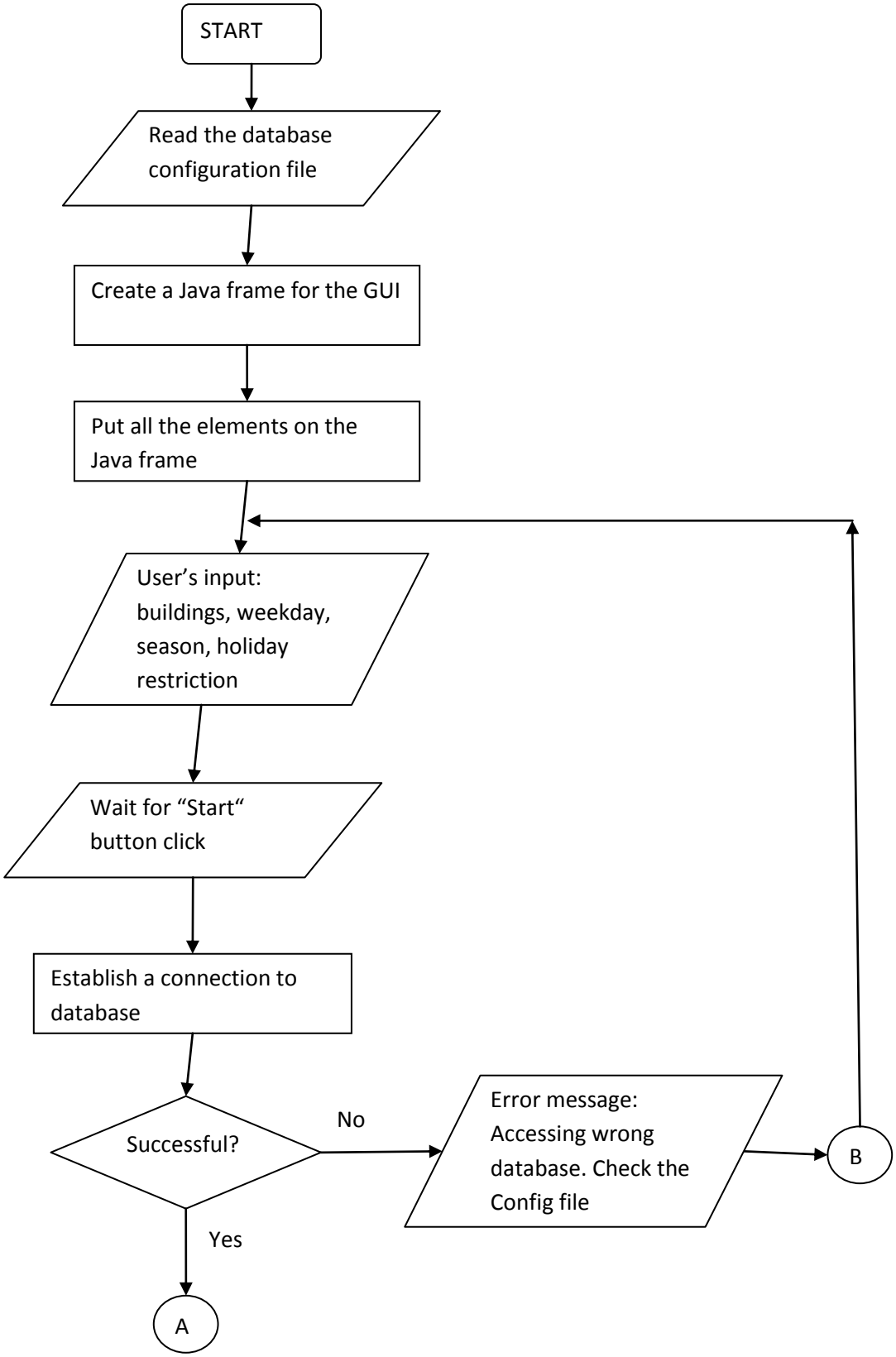
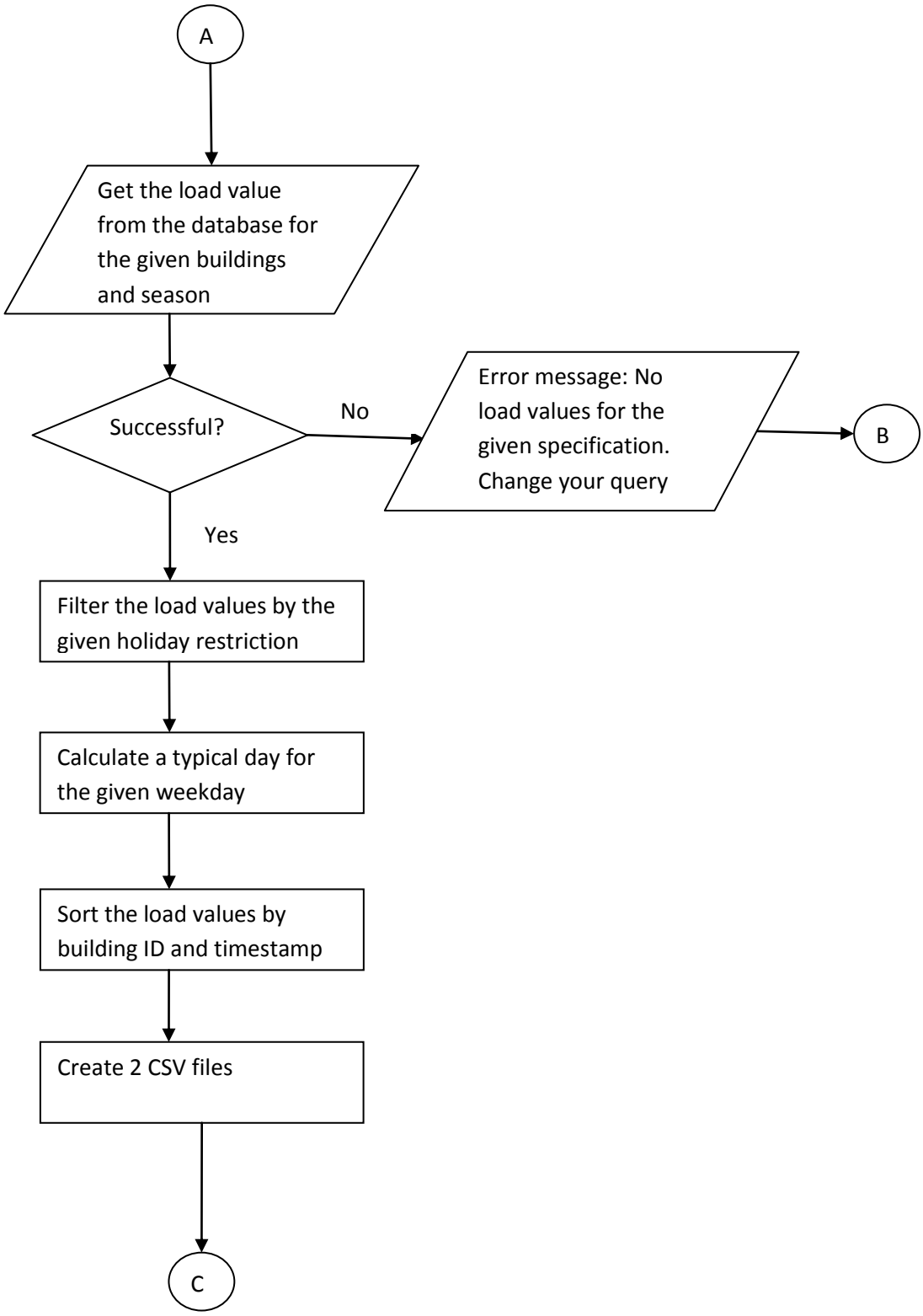


Figure 8.6: User Information Message: CSV file created

The following flow chart graphically shows how the program works.


START
Read the database configuration file
Create a Java frame for the GUI
Put all the elements on the Java frame
User's input: buildings, weekday, season, holiday restriction
Wait for "Start" button click
Establish a connection to database
Successful?
No
Error message: Accessing wrong database. Check the Config file
B
Yes
A


A
Get the load value from the database for the given buildings and season
Successful?
No
Error message: No load values for the given specification. Change your query
B
Yes
Filter the load values by the given holiday restriction
Calculate a typical day for the given weekday
Sort the load values by building ID and timestamp
Create 2 CSV files
C

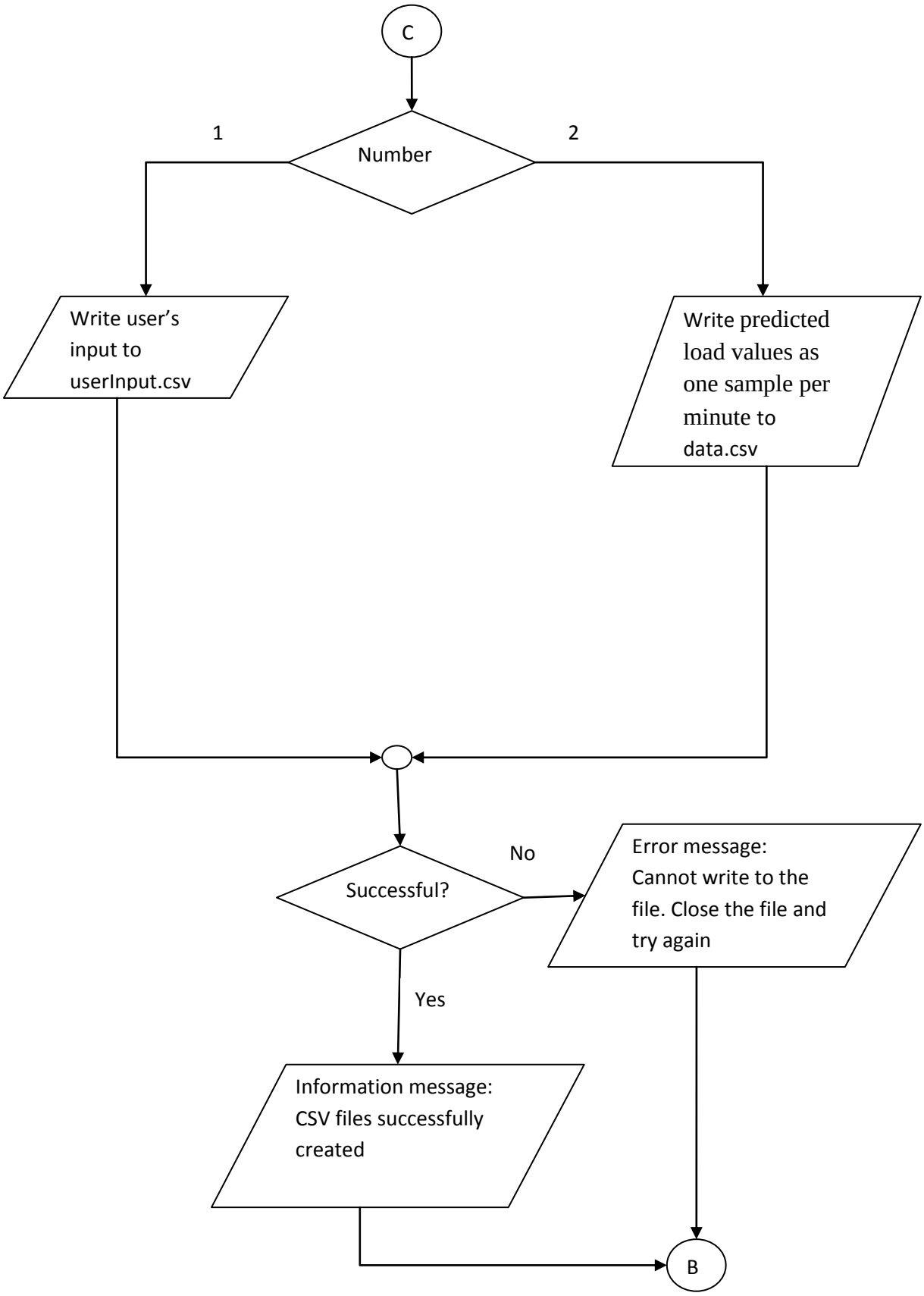


Figure 8.7: Flowchart of Load Forecasting a typical day

Chapter 9

Test Results

In comparing different results, the average percentage forecasting error is used as a measure of the performance. This is defined as:

$$\% \text{ error} = \frac{|\text{java generated value} - \text{measured value}|}{\text{measured value}} \times 100\ \%$$

The reason for using the average percentage error is the fact, that its meaning can be easily understood. It is also the most often used error measure in the load forecasting.

Another possibility as a performance measure would be the root-mean-square (RMS) percent error. This penalizes the square of the error as opposed to the average forecasting error. Therefore, the RMS takes the deviation of the errors into account more effectively. However, when both measures were calculated on some test models with relatively small errors, the orders of preference were in practice the same with both measures. Therefore, the average forecasting error will be used throughout this chapter.

The results of forecasting the daily load profile using the algorithm given in chapter 5 are given in this chapter. To test the performance of the load shape forecasting model, the prediction was carried out with the historical data of forty public properties.

To get a good evaluation of the performance of the model, testing was carried out for two weekdays, Monday and Wednesday, with two cases: including holidays and excluding holidays. One case with including holidays was tested just to show how they disturb the forecasting. Furthermore, two cases were tested: First, forecasting a typical day for only one public property and second, forecasting the sum load for forty public properties.

Figure 9.1 shows the sum load of forty properties for a typical Monday in spring including the holidays. As it may be seen, there is a huge difference between the forecasted curve and the actual curve. The reason is the usage of all historical data which were measured on Mondays, including the holidays. Holidays are more difficult to forecast than non-holidays because of

their relative infrequent occurrence. With respect to figure 9.1, figure 9.2 shows the error rate measured in 15 minute samples. Although the average error rate is 7.73 percent, it shows that during the day, the errors of more than 15 percent occur frequently. Furthermore it can be seen that the error rate is especially low during base load times (night).

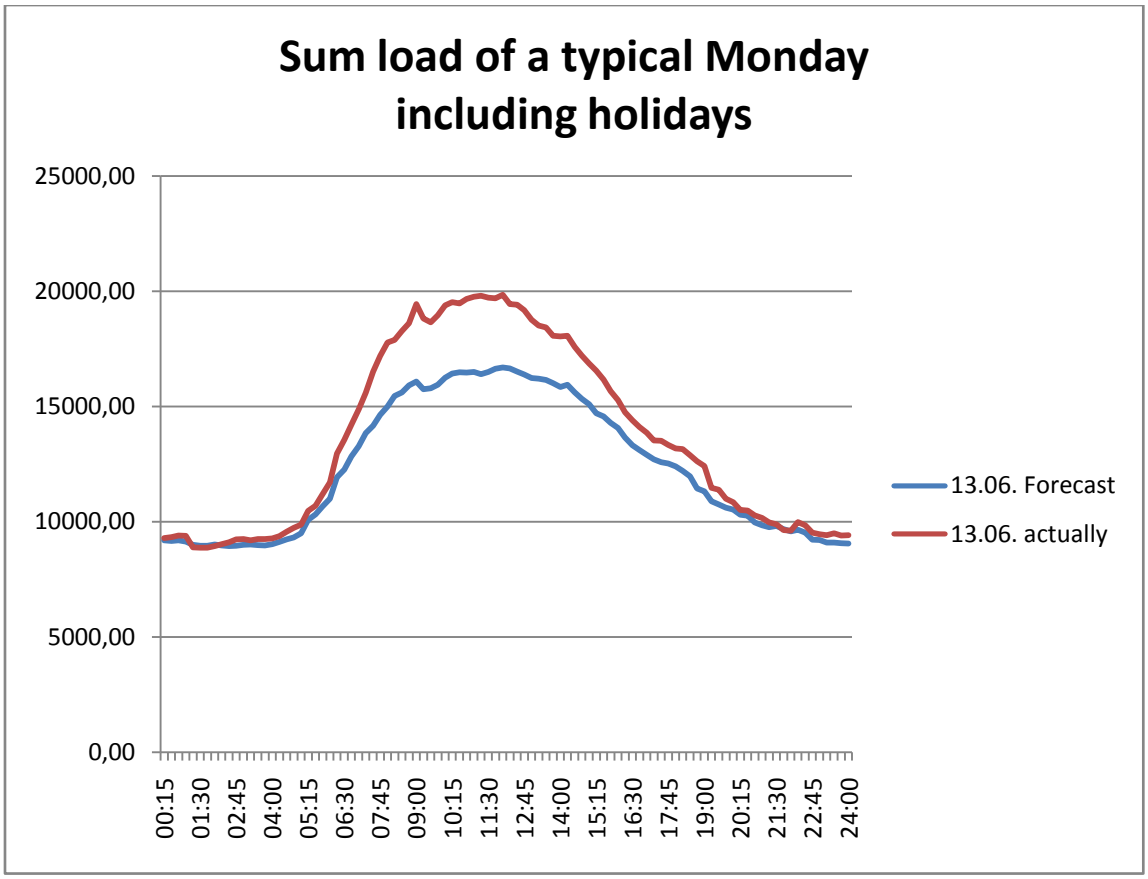


Figure 9.1: Sum load of a typical Monday including holidays

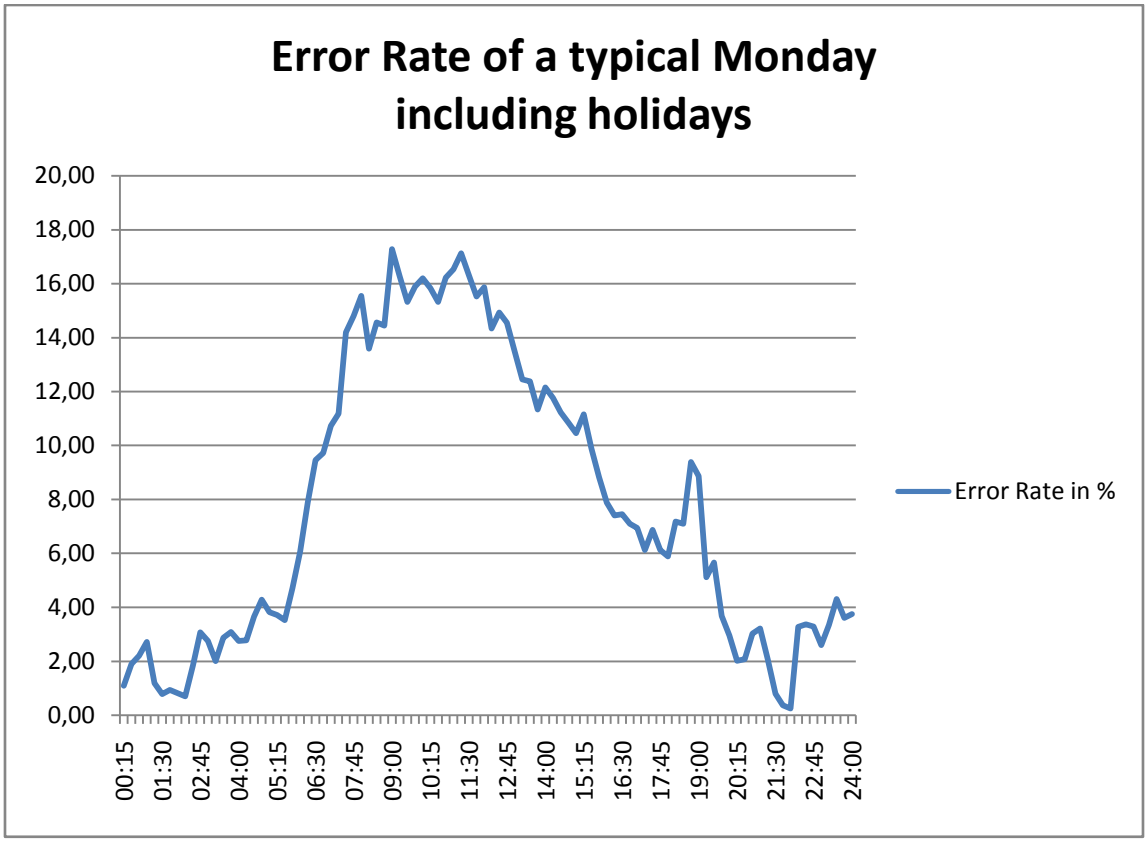


Figure 9.2: Error Rate of a typical Monday including holidays

A better solution would is to exclude the holidays. Again, the sum loads of forty properties for a typical Monday in spring have been calculated, now excluding the holidays. From the historical data only those Mondays have been taken which are non holidays. Figure 9.3 shows a better performance compared to figure 9.1. The average error rate is reduced up to 3.4 percent and is now 4.34 percent. There are still some points in time where the error rate is about 10 percent. The problem is the school holidays because the schools have lower consumption during the spring holidays. Furthermore, the universities also have holidays. Qualitatively the algorithm performs well. It produces all the ups and downs quite well. Most of the time, the error rate is under 5-6 percent which is a good result.

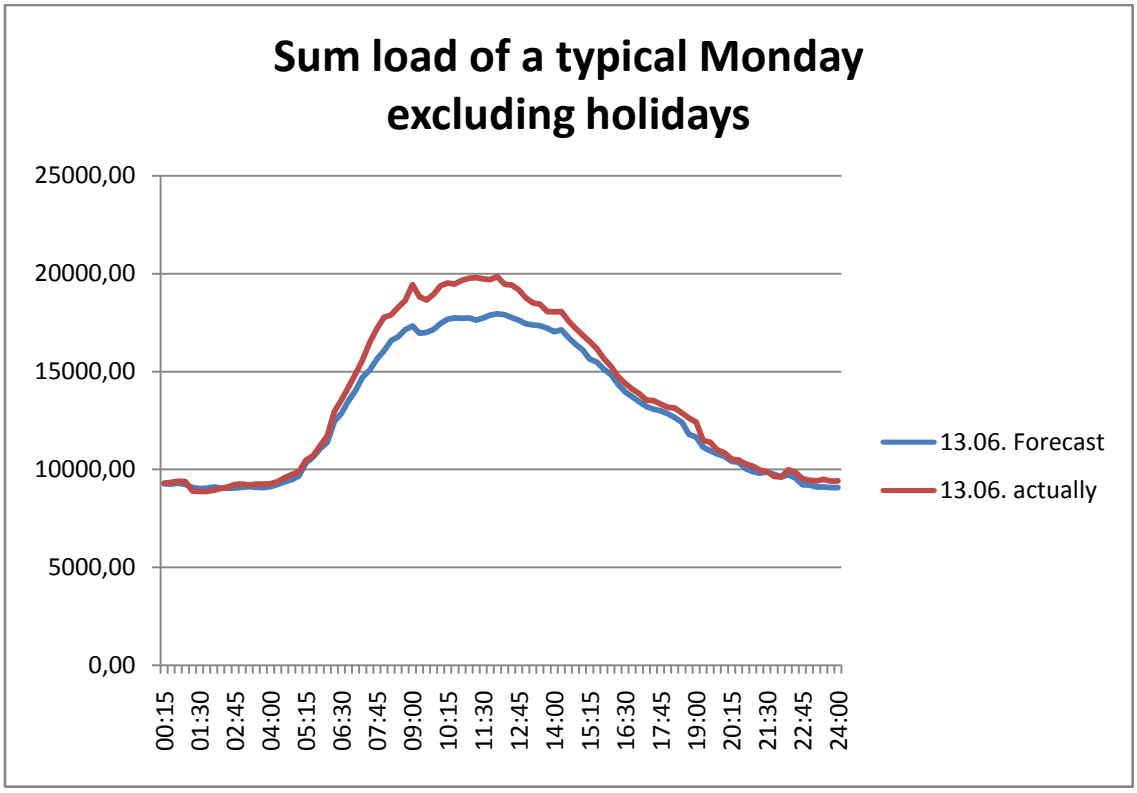


Figure 9.3: Sum load of a typical Monday excluding holidays

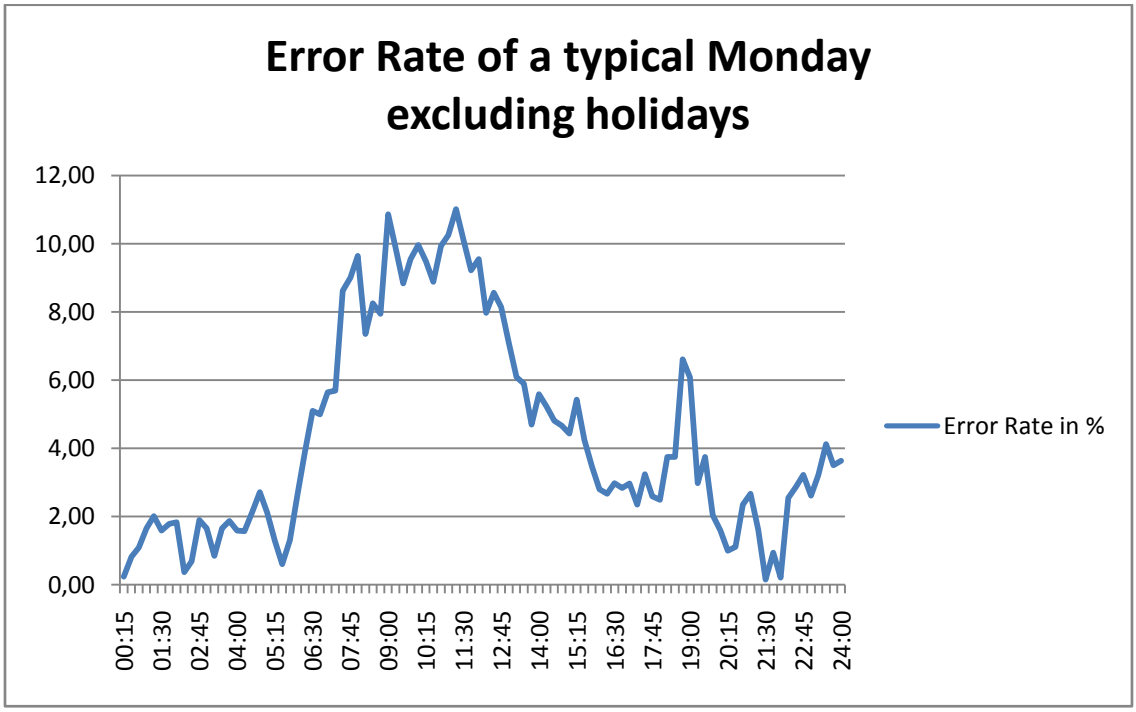


Figure 9.4: Error Rate of a typical Monday excluding holidays

The same test has been done for a Wednesday. As the following figure shows, it behaves differently compared to load curve of a typical Monday. During the night and morning time both curves behave similar, i.e. the error rate is low. In the evening time, the error rate rises. It even reaches 10 percent. The average error rate being about 5 percent is acceptable.

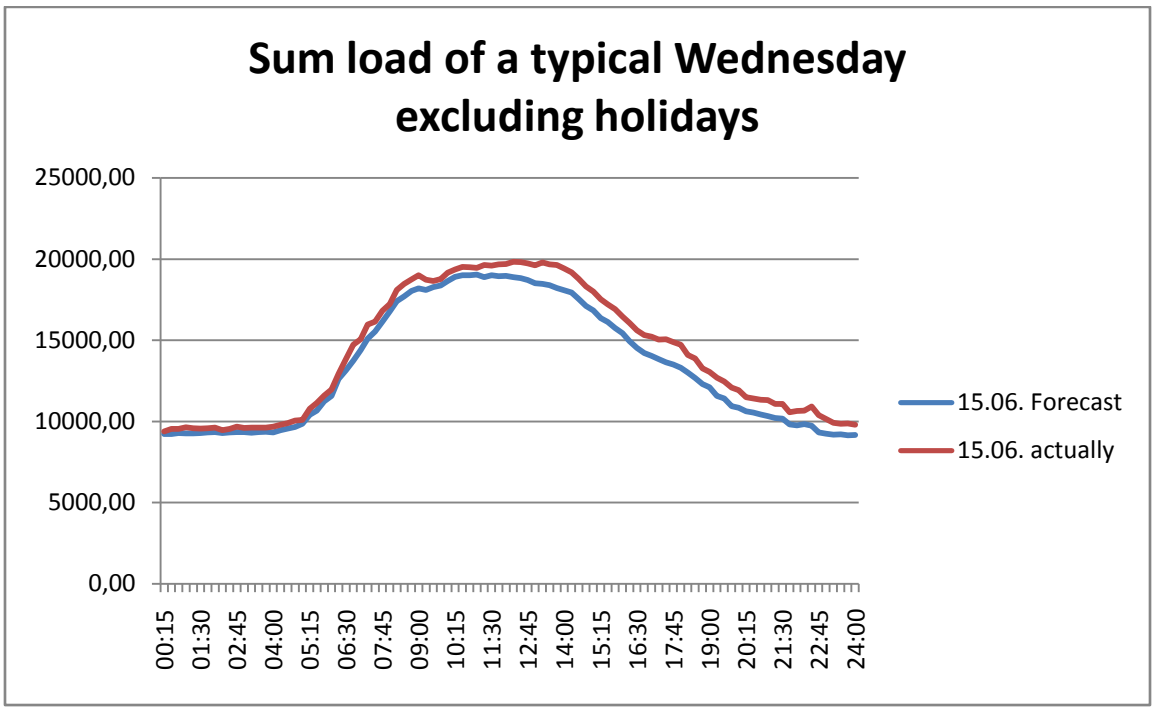


Figure 9.5: Sum load of a typical Wednesday excluding holidays

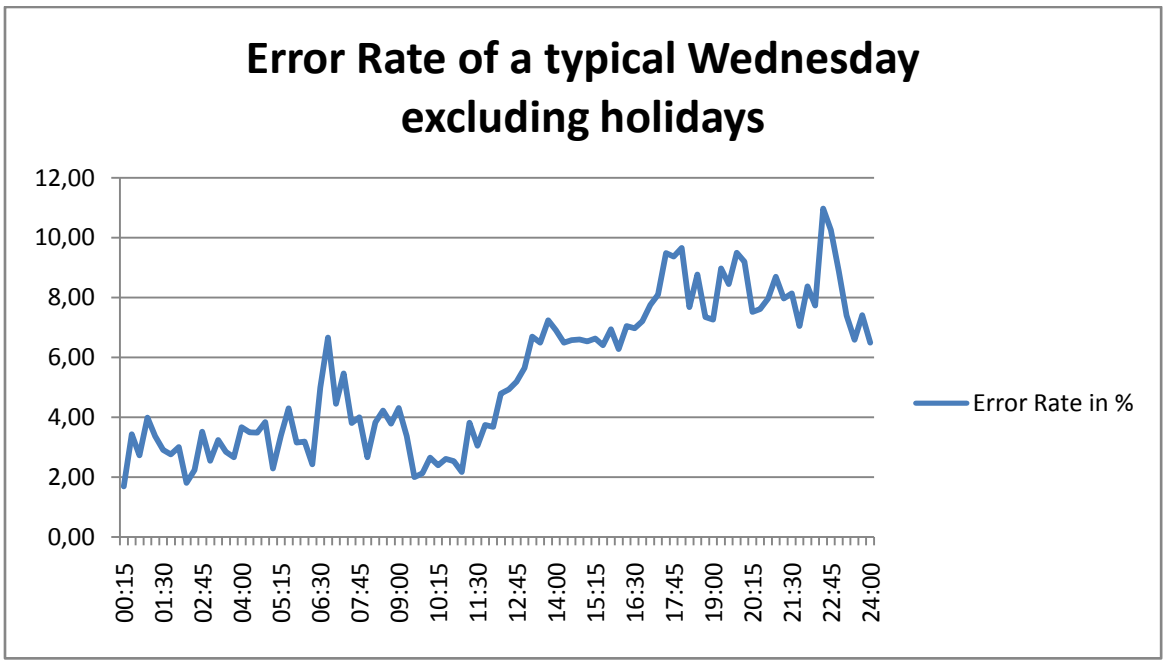


Figure 9.6: Error Rate of a typical Wednesday excluding holidays

Following is the figure of a typical Wednesday for only one property, the HAW Hamburg Berliner Tor. The curves show that the predicted load curve is almost similar to the actual load curve. As the figure shows, the error rate is flattering around the 5 percent rate. Although at 19:45 the error rate goes up to 20 percent, the average error rate is still acceptable being about 5 percent. This is due the fact that only one building is predicted here. In this case, the algorithm performs quantitatively well but qualitatively, it shows a curve that is far too smooth. For example, at night there is this highly characteristic zigzag of the compressor that produces the pressurized air. It is flattened out in the calculation because the ups and downs don't always occur at the same time. This very same effect grants a better prediction for a higher a number of buildings (see figure 9.5). The load curves smooth each other when peaks meet lows at the same point in time.

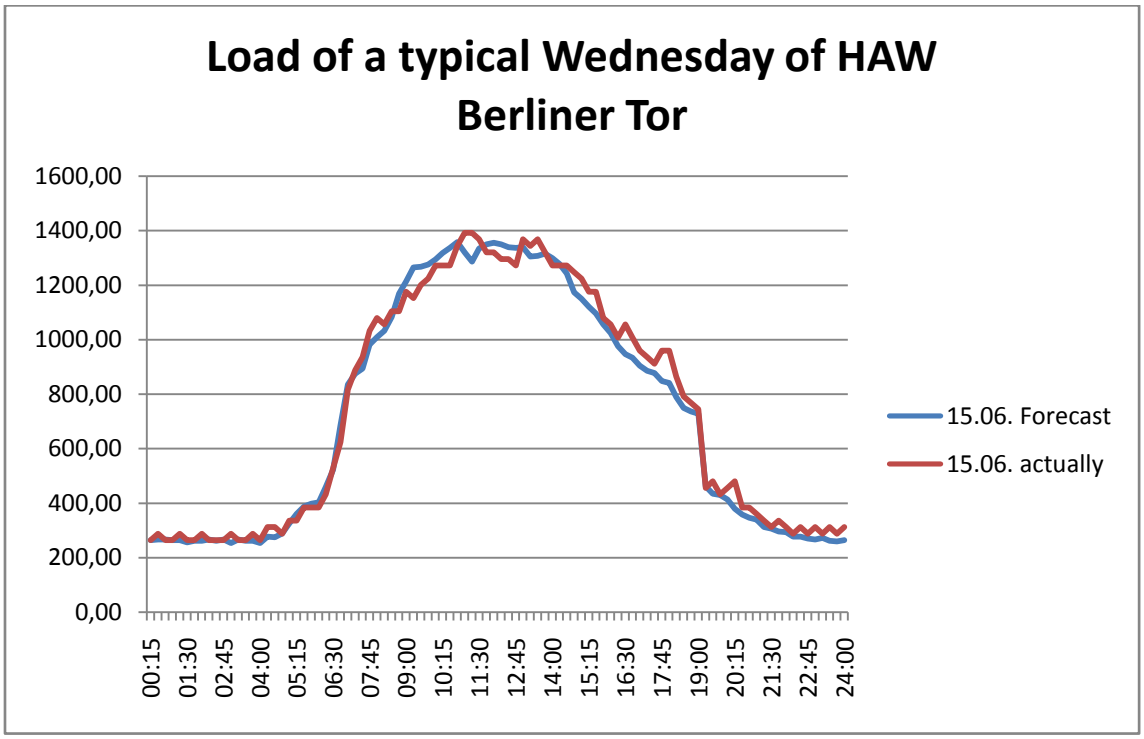


Figure 9.7: Load of a typical Wednesday of HAW Berliner Tor

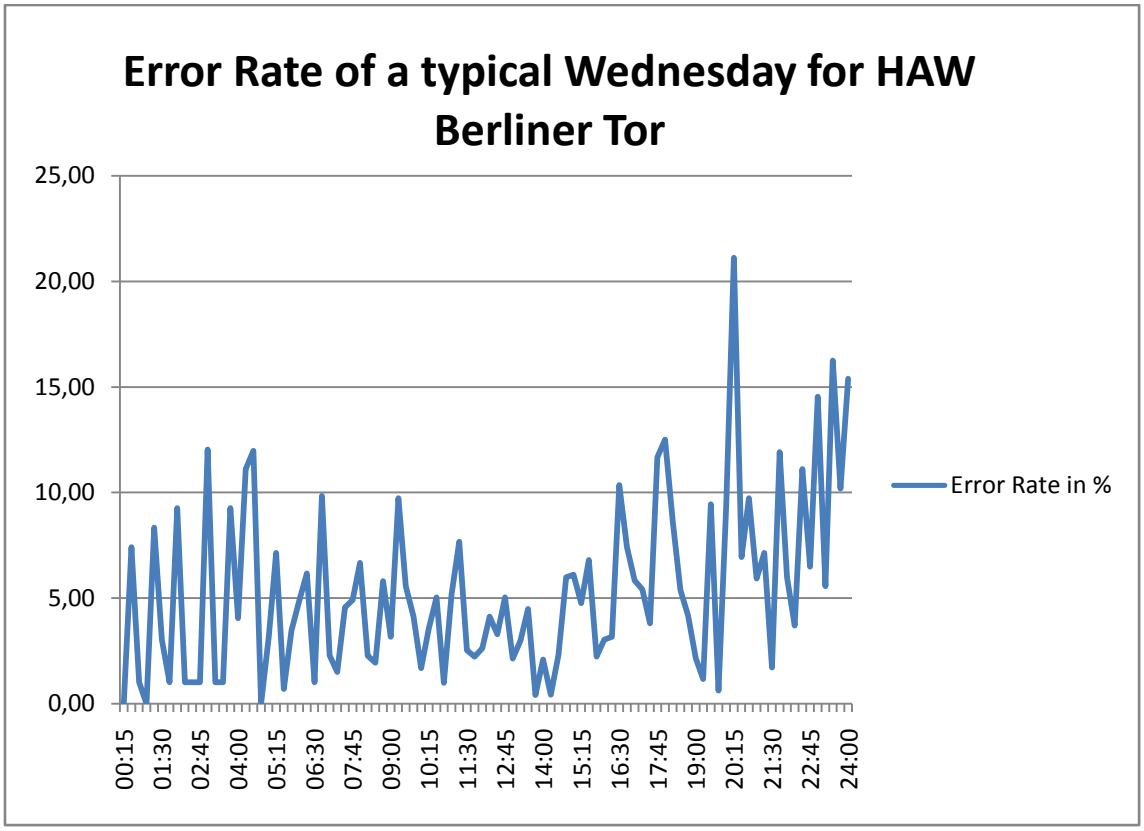


Figure 9.8: Error Rate of a typical Wednesday of HAW Berliner Tor

All the figures discussed up to now, represent the performance of the algorithm without randomized historical load values. The next figure represents a predicted curve which has been generated using the randomized historical load values (see chapter 7). The figure 9.9 shows the load curve of the campus Bergedorf on the 9th May 2005. The blue curve is the real measured load curve on that day and the red curve is the forecasted load curve for that day. As it may be seen, using the load forecasting method Neural Networks, the java generated curve performs very well as compared to the real world measured values.

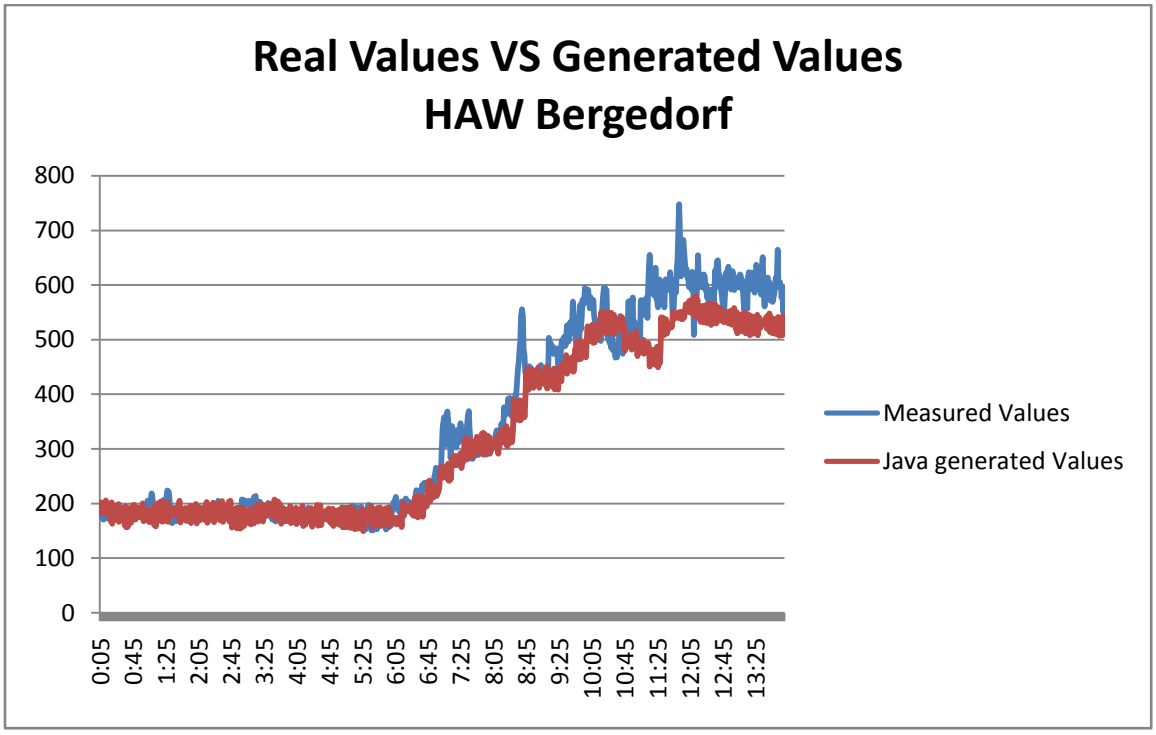


Figure 9.9: Real Values VS Generated Values (HAW Bergedorf)

The following figure shows the measured values and Java generated values of a school. As it may be seen from the graph, the Java generated values is smaller than the measured values, especially, during the day. The reason for that is that by forecasting a typical day all the similar day load values from the database being non holiday is taken into consideration which is required for any public property but not for a school. Because in addition to national holidays, the schools also have other holidays like summer holidays, for example. These days are also included when forecasting a day which will not perform well when forecasting a typical day for a school.

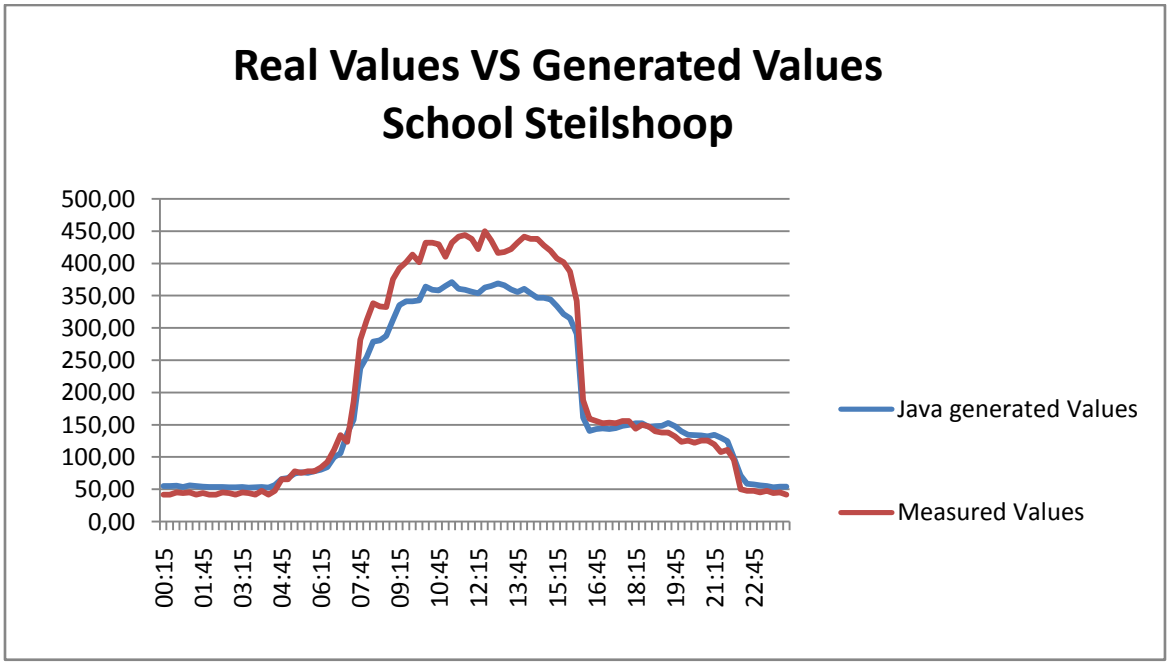


Figure 9.10: Real Values VS Generated Values (School Steilshoop)

Chapter 10

Conclusion

Forecasting the daily load profile was studied using the algorithm given in chapter 5 to forecast the load profile of a typical day. The shape of the load curve was forecasted by using the historical data. The performance of the model was tested on the historical load data of some public properties placed in Hamburg. The best results were obtained with the simplest input structure, which uses the peak value, valley value and the average value of the similar days in historical data as the predictor. The results were better for excluding the holidays as compared to including the holidays.

These error percentages are somewhat larger than many of those reported in the literature. Some other sources have reported average forecasting errors of around 2 % for the total load forecasts ([24] and [25]). The reason for the difference lies most likely in the nature of the test data sets. The existence of bad data in historical load curve affects the precision of load forecasting result. Bad data could be the use of school holidays or the effect of the weather.

A future refinement would be to modify the database by adding one additional column in which a flag should be set if the date on that row is a school holiday. By doing this, the problem of forecasting a typical day for a school will be avoided. As a result, the differences shown in figure 9.9 and 9.10 will be less or even removed.

An additional future refinement would be to make use of the temperatures. It is not a must that the summer is always hot and the winter is always cold. Like for the school holidays, there should be an additional column indicating the temperature of a day.

The developed load forecasting tool performs well with respect to speed. To make the program run even faster, the program might be modified. This program gets first the month interval from the database and inside the code it filters out the selected day and the selected holiday restriction using the specifications given by the user on the GUI. A modification proposal would be to get the only the needed data from the database, i.e. selected month, day and holiday restriction. In this version, it is not done in that way because the database still need to be modified as stated above, i.e. additional columns for holidays and temperature.

Chapter 11

References

[1] http://users.etech.haw-hamburg.de/users/Schubert/forschung.html [06.12.2007]

[2] http://www.nertec.com/standards/ansic1222/JM0107-097-1.doc [09.12.2007]

[3] http://www.allbusiness.com/business-planning/101341-1.html [18.12.2007]

[4] http://www.peci.org/library/PECI_PracticalGuide1_0302.pdf [26.12.2007]

[5] http://www.gocsc.com/uploads/white_papers/FD06E0A2870042769EB8D66129456A95.pdf [29.12.2007]

[6] http://www.sbt.siemens.com/hvp/staefa/press/pr5.asp [27.02.2008]

[7] http://java.sun.com [20.02.2008]

[8] http://ei.cs.vt.edu/book/chap1/java_hist.html [22.02.2008]

[9] http://www.faqs.org/docs/javap/c1/s3.html [21.02.2008]

[10] http://java.sun.com/docs/books/tutorial/jdbc/basics/index.html [21.02.2008]

[11] http://download.oracle.com/docs/cd/B25329_01/doc/appdev.102/b25320/toc.htm [28.02.2008]

[12] R.F. Engle, C. Mustafa, J. Rice, ‘Modeling peak electricity demand’, Journal of Forecasting, 1992, 11, 241 - 251

[13] O. Hyde, P.F. Hodnett, ‘An Adaptable automated procedure for short-term electricity load forecasting’, IEEE Transactions on Power Systems, 1997,12, 84 - 93

[14] S. Ruzic, A. Vuckovic, N. Nikolic, ‘Weather sensitive method for short-term load forecasting in electric power utility of Serbia’, IEEE Transactions on Power Systems, 2003, 18, 1581 - 1586

[15] T. Haida, S. Muto, ‘Regression based peak load forecasting using a transformation technique’, IEEE Transactions on Power Systems, 1994, 9, 1788 - 1794

[16] W. Charytoniuk, M.S. Chen, P.Van Olinda, ‘Nonparametric regression based short-term load forecasting’, IEEE Transactions on Power Systems, 1998, 13, 725 - 730

[17] J.Y. Fan, J.D. McDonald, ‘A real-time implementation of short – term load forecasting for distribution power systems’, IEEE Transactions on Power Systems, 1994, 9, 988 - 994

[18] M.Y. Cho, J.C. Hwang, C.S. Chen, ‘Customer short-term load forecasting by using ARIMA transfer function model’, Proceedings of the International Conference on Energy Management and Power Delivery, EMPD, 1995, 1, 317 - 322

[19] H.T.Yang, C.M. Huang, C.L. Huang, ‘Identification of ARMAX model for short-term load forecasting, An evolutionary programming approach’ IEEE Transactions on Power Systems, 1996, 11, 403 - 408

[20] H.T. Yang, C.M. Huang, ‘A new short-term load forecasting approach using self-organizing fuzzy ARMAX models’, IEEE Transactions on Power Systems, 1998, 13, 217 - 225

[21] http://www.sal.hut.fi/Publications/pdf-files/tmur98.pdf [20.02.2008]

[22] http://en.wikipedia.org/wiki/Unified_Modeling_Language [05.01.2008]

[23] http://ieeexplore.ieee.org/iel5/10834/34152/01627164.pdf [24.02.2008]

[24] Park, D. C., M. A. El-Sharkawi, R. J. Marks II, L. E. Atlas, M. J. Damborg, 1991a, "Electric load forecasting using an artificial neural network", IEEE Transactions on Power Systems, Vol. 6, No. 2, May 1991, pp. 442-449.

[25] Peng, T. M., N. F. Hubele, G. G. Karady, 1992, "Advanvement in the application of neural networks for short-term load forecasting", IEEE Transactions on Power Systems, Vol. 7, No. 1, February 1992, pp. 250-256.

[26] www.ecse.rpi.edu/homepages/chowj/Feinberg.ppt [03.03.2008]

APPENDIX A

Java Source Code

A.1 Config.txt

jdbc:oracle:thin:@localhost

jevi

jevi

//////////////// Configfile ////////////////////////////

//   //

//   //

// row 01 : JDBC URL   //

// row 02 : Username   //

// row 03 : Password   //

//

A2. JDBCQuery.java

```
package loadmanagement;

import java.io.RandomAccessFile;
import java.sql.*;
import oracle.jdbc.pool.OracleDataSource;


/**
 * @author Farid
 *
 */
public class JDBCQuery {
      String jdbcUrl;
      String userid;
      String password;
      Connection conn;
      Statement stmt;
      ResultSet rset;
      String sqlString;
      RandomAccessFile conf;

        /***
            * Constructor; Reads from the file "Config.txt" the jdbcUrl;
            *userid and password
         ***/
        public JDBCQuery(){
            try{
                conf = new RandomAccessFile("Config.txt", "r");
                conf.seek(0);
                jdbcUrl = conf.readLine();
                userid = conf.readLine();
                password = conf.readLine();
            }catch (Exception exception)
            {
                  System.out.println("config-file read error: " +
                  exception.toString());
            }
        }

      /**
       * connects to database using the specification given in Config.txt
       * @throws SQLException
       */
      public void getDBConnection() throws SQLException {
            try {
                  OracleDataSource ds = new OracleDataSource();
                  ds.setURL(jdbcUrl);
                  conn = ds.getConnection(userid, password);
            }catch ( SQLException ex ){
                  System.out.println(ex.toString());
            }
      }


      /**
       * gets data from database using the user's input on the GUI
       * @param query
       * @return result set
       * @throws SQLException
```

```
	 */
	public ResultSet getData(String query) throws SQLException{
		getDBConnection();
		stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,
		ResultSet.CONCUR_READ_ONLY);
		rset = stmt.executeQuery(query);

		return rset;
	}

	/**
	 * updates the database, i.e. delete rows or insert new rows
	 * @param query
	 * @throws SQLException
	 */
	public void updataDB(String query) throws SQLException{
		getDBConnection();
		Statement st = conn.createStatement();
		st.executeUpdate(query);
		st.close();
	}
}
```

A3. Fifteen2oneMin.java

```
/**
 *
 */
package loadmanagement;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Random;
import java.util.Vector;

/**
 * @author Farid
 *
 */
public class Fifteen2oneMin {


	/**
	 * @param args
	 * @throws SQLException
	 */
	public static void main(String[] args) throws SQLException {
		JDBCQuery data = new JDBCQuery();
		Vector <Long> usedIDs = new Vector<Long>();
		ResultSet data_row;
		ResultSet data_Value;
		long[] buildingID;
		//long id = 1191376;
		long id = 0;

		String query = "SELECT DISTINCT DATA.DATA_ROW_ID FROM DATA
		ORDER BY DATA.DATA_ROW_ID";

		data_row = data.getData(query);
		int i=0;
		data_row.last();
		int numOfBuildings = data_row.getRow();
		buildingID = new long[numOfBuildings];
		data_row.first();

		do{
			buildingID[i] = data_row.getLong("DATA_ROW_ID");
			i++;
		}while (data_row.next());
		i=0;
		//System.out.println(buildingID[1]);
		for(int l=0; l<buildingID.length; l++){
			query = "SELECT DISTINCT DATA.ID, DATA.DATA_ROW_ID,
			DATA.VALUE, DATA.TIMESTAMP " +
			"FROM DATA " +
			"WHERE DATA.DATA_ROW_ID =" + buildingID[l] +
			"ORDER BY DATA.DATA_ROW_ID,DATA.TIMESTAMP";
			data_Value = data.getData(query);

			Vector <Double> finalResult = randomize(data_Value);
```

```
            query = "DELETE FROM DATA WHERE DATA.DATA_ROW_ID =" +
            buildingID[l];
             data.updataDB(query);

      query = "SELECT DISTINCT DATA.ID FROM DATA ORDER BY DATA.ID";
      data_Value = data.getData(query);
      usedIDs.clear();

      while(data_Value.next()){
            usedIDs.add(data_Value.getLong("ID"));
      }


            try{
                  int j=0;
                // Open the file that is the first
                // command line parameter
                FileInputStream fstream = new
                FileInputStream("TimestampInOneMinSteps2005.txt");
                // Get the object of DataInputStream
                DataInputStream in = new DataInputStream(fstream);
                BufferedReader br = new BufferedReader(new
                InputStreamReader(in));
                String timestamp;
                //Read File Line By Line
                while ((timestamp = br.readLine()) != null)   {
                  id++;
                  if(usedIDs.contains(id)){
                        id = usedIDs.lastElement()+1;
                  }

                  query = "INSERT INTO DATA
                  (ID,TIMESTAMP,VALUE,DATA_ROW_ID)" +
                  "VALUES(" + id+ ", to_date('"+ timestamp + "',
                  'dd.mm.yyyy hh24:mi:ss')," +
                  finalResult.elementAt(j)+ "," + buildingID[l] +
                  ")";
                  data.updataDB(query);

                  System.out.println (id + ", " + timestamp +", " +
                  finalResult.elementAt(j)+", " +buildingID[l]);
                  j++;
                }
                //Close the input stream
                in.close();
                }catch (Exception e){//Catch exception if any
                  System.err.println("Error: " + e.getMessage());
                }
                id = 0;
      }

}

/**
 * @param data_Value
 * @return randomized Values from 15 Min to 1 Min Samples
 * @throws SQLException
 */
      public static Vector<Double> randomize(ResultSet data_Value)
      throws SQLException{
      Vector <Double> fetchedValues = new Vector<Double>();
      Vector <Double> result = new Vector<Double>();
      Vector <Double> a = new Vector<Double>();
```

```
int i=0;
int counter = 0;
int position = 0;
double[] randomValue = new double[15];
Random generator = new Random();
while(data_Value.next()){
      fetchedValues.add(data_Value.getDouble("VALUE"));
      //usedIDs.add(data_Value.getLong("ID"));
      //System.out.println(fetchedValues.elementAt(i));
      i++;
}

Vector <Double> dkW = new Vector<Double>();
Vector <Double> dkWProMin = new Vector<Double>();
Vector <Double> step = new Vector<Double>();

//Preparing Linear function y= ax+b for randomizing
//1. differnce between two 15 Min Values
//2. divide the difference by 15 to get the scope between
//minutes
//3. mutiply the scope by 7 to get the 1. value
//4. add the scope tp every minute value
//5. randomize


//System.out.println(fetchedValues.size());

  /* Makes out of 15 Minutes sampled Values
 * 1 minute sampled values
 * It generates random Values, so that the averaged
 * 15 Minutes Value is same the original 15 Minutes sampled
 * Value
 */

for(int k=0; k<fetchedValues.size(); k++){
      dkW.add(fetchedValues.elementAt(k+1) -
      fetchedValues.elementAt(k));
      dkWProMin.add(dkW.elementAt(k)/15);
      step.add(dkWProMin.elementAt(k)*7);
}

for(int k=0; ; k+=15){
      a.add(k,fetchedValues.elementAt(position) -
      step.elementAt(position));
      position++;
      if(position >= fetchedValues.size())
            break;
}

position=0;
int new15thMinValue = 1;

for(int k=1; ; k++){
      a.add(k, a.elementAt(k-1) +
      dkWProMin.elementAt(position));
      if((k+1)%15 ==0){
            k=15*new15thMinValue;
            position++;
            new15thMinValue++;
      }
      if(position >= dkWProMin.size())
            break;
}
```

```
            /*for(int k=0; k<a.length; k++){
                  System.out.println(a[k]);
            }*/
            position=0;
            new15thMinValue=0;


            for(int k=0; ; k++){
                        if(counter == 0){
                              for(int l=0; l<randomValue.length; l++){
                                    randomValue[l] = generator.nextDouble()
                                    * 10 + 95;
                                    //System.out.println(randomValue[l]);
                              }
                              /*for(int m=1; m<randomValue.length; m+=2){
                                    randomValue[m] = -randomValue[m-1];
                                    System.out.println(randomValue[m]);
                              }*/
                        }

                  result.add(HAWCalc.round(a.elementAt(position) *
                  randomValue[counter] /100));
                  //System.out.println(result.elementAt(k));
                  counter++;
                  if(counter == 15)
                        counter = 0;
                  if(k!=0)
                        if((k+1)%15 == 0)
                              position++;
                  if(position >= fetchedValues.size())
                        break;
            }
            position=0;
            //System.out.println(fetchedValues.size());
            return result;
      }

}
```

A4. HAWCalc.java

```
package loadmanagement;

import javax.swing.JPanel;

import java.awt.Color;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import java.io.IOException;
import java.math.BigDecimal;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Random;
import java.util.Vector;

import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.DefaultListModel;
import javax.swing.InputVerifier;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JProgressBar;
import javax.swing.JRadioButton;
import javax.swing.JScrollPane;
import javax.swing.JTextField;
import javax.swing.ListSelectionModel;
import javax.swing.SwingUtilities;
import javax.swing.border.TitledBorder;
import javax.swing.JFrame;


/**
 * @author Farid
 *
 */

public class HAWCalc extends JPanel implements ActionListener{

      private static final long serialVersionUID = 1L;
      JComboBox dataComboBox = new JComboBox();
      JLabel seasonLabel = new JLabel();
      JPanel weekPanel =  new JPanel();
      JPanel seasonPanel = new JPanel();
      JPanel holidayPanel = new JPanel();
      JPanel temperaturePanel = new JPanel();
      JPanel getPanel = new JPanel();
      JPanel buildingPanel = new JPanel();
      JPanel buttonPanel = new JPanel();
      JPanel progressBar = new JPanel();
```

```
Color valueIsOKColor = new Color(255,255,255);
Color valueIsNotOKColor = new Color(255,67,54);
static JFrame frame;

private JList list;
private DefaultListModel listModel;
String[] buildingName;
int[] buildingID;
JButton reloadButton;
JButton selectAllButton;
JButton getButton;
JButton okayButton;
JLabel tempLabel;
JTextField tempValue;
JProgressBar bar;

int day = 2;            //default day -> Monday
String season = "1";    //default season -> spring
String holiday = "2";   //default restriction -> exclude Holidays
String temperature = "";
String startMonth = "04";
String endMonth = "05";

static String MondayString = "Monday";
static String TuesdayString = "Tuesday";
static String WednesdayString = "Wednesday";
static String ThursdayString = "Thursday";
static String FridayString = "Friday";
static String SaturdayString = "Saturday";
static String SundayString = "Sunday";
static String SpringString = "Spring";
static String SummerString = "Summer";
static String AutumnString = "Autumn";
static String WinterString = "Winter";
static String incHolidayString = "Include Holidays";
static String exHolidayString = "Exclude Holidays";
static String onlyHolidayString = "Only Holidays";


/**
 * Constructor; sets the layout for the GUI
 *
 */
public HAWCalc(){
      setBorder(new
      TitledBorder(BorderFactory.createEtchedBorder(Color.white,
          new Color(148,
                    145,
                    140)),
                    "Typical Day for:"));
      this.setLayout(new GridBagLayout());
      ((GridBagLayout)this.getLayout()).columnWidths = new int[]
      {160, 0, 0, 0, 0};
      ((GridBagLayout)this.getLayout()).rowHeights = new int[] {0, 0,
      0};
      ((GridBagLayout)this.getLayout()).columnWeights = new double[]
      {0.0, 0.0, 0.0, 0.0, 1.0E-4};
      ((GridBagLayout)this.getLayout()).rowWeights = new double[]
      {0.0, 0.0, 1.0E-4};

      buildingPanel = getBuilding();
      weekPanel = getDaytimePanel();
      seasonPanel = getSeasonPanel();
```

```
        holidayPanel = getHolidayPanel();
        temperaturePanel = getTemperaturePanel();
        getPanel = getGetPanel();
        progressBar = getProgressBar();


        add(buildingPanel, new GridBagConstraints(0, 0, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(weekPanel, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(seasonPanel, new GridBagConstraints(0, 2, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(holidayPanel, new GridBagConstraints(0, 3, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(temperaturePanel, new GridBagConstraints(0, 4, 1, 1, 0.0,
        0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(getPanel, new GridBagConstraints(0, 5, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        add(progressBar, new GridBagConstraints(0, 6, 1, 1, 0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
    }

    /**
     * gets the building from the database and puts
     * on the GUI
     * @return Building Panel
     */
    private JPanel getBuilding(){
        JPanel building = new JPanel();
        building.setBorder(new
        TitledBorder(BorderFactory.createEtchedBorder(Color.white,
            new Color(148,
                      145,
                      140)),
                      "Building"));
        building.setLayout(new GridBagLayout());

        listModel = new DefaultListModel();

        String query = "SELECT DATA_ROWS.ID, DATA_ROWS.DESCRIPTION FROM
        DATA_ROWS ORDER BY DATA_ROWS.ID";
        try {
        JDBCQuery comboBox = new JDBCQuery();
        ResultSet rset = comboBox.getData(query);
        String rowsDescription = "";
        int i=0;
        rset.last();
        int numOfBuildings = rset.getRow();
        buildingName = new String[numOfBuildings];
```

```
        buildingID = new int[numOfBuildings];
        rset.first();

        do{
              rowsDescription = rset.getString("DESCRIPTION");
              listModel.addElement(rowsDescription);
              buildingName[i] = rset.getString("DESCRIPTION");
              buildingID[i] = rset.getInt("ID");
              i++;
        }while (rset.next());

        i=0;
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    //Create the list and put it in a scroll pane.
    list = new JList(listModel);

        list.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELE
        CTION);
        list.setSelectedIndex(0);
        list.setVisibleRowCount(5);

        building.add(new JScrollPane(list), new GridBagConstraints(0,
        0, 1, 1, 0.0, 0.0,
        GridBagConstraints.CENTER, GridBagConstraints.BOTH,
        new Insets(0, 0, 5, 5), 0, 0));

        buttonPanel = getButtonPanel();

        building.add(buttonPanel, new GridBagConstraints(1, 0, 1, 1,
        0.0, 0.0,
        GridBagConstraints.CENTER, GridBagConstraints.BOTH,
        new Insets(0, 0, 5, 5), 0, 0));


        return building;
}

  /**
   * initializes the Reload and the Select all button
   * @return Button
   */
  private JPanel getButtonPanel(){
    JPanel button = new JPanel();

    button.setLayout(new GridBagLayout());

    reloadButton = new JButton("Reload");
    reloadButton.setToolTipText("Reload available Buildings");
    reloadButton.addActionListener(this);

    selectAllButton = new JButton("Select All");
    selectAllButton.setToolTipText("Select All Buildings");
    selectAllButton.addActionListener(this);

    button.add(reloadButton, new GridBagConstraints(0, 0, 1, 1, 0.0,
    0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
```

```
  button.add(selectAllButton, new GridBagConstraints(0, 1, 1, 1, 0.0,
  0.0,
                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                new Insets(0, 0, 5, 5), 0, 0));


  return button;
}

/**
 * creates Radio buttons initialized with weekdays
 * @return Weekdays Panel
 */
private JPanel getDaytimePanel() {
      JPanel weekday = new JPanel();
      weekday.setBorder(new
TitledBorder(BorderFactory.createEtchedBorder(Color.white,
          new Color(148,
                    145,
                    140)),
                    "Weekday"));
  weekday.setLayout(new GridBagLayout());
  JRadioButton mondayButton = new JRadioButton(MondayString);
  mondayButton.setActionCommand(MondayString);
  mondayButton.setSelected(true);

  JRadioButton tuesdayButton = new JRadioButton(TuesdayString);
  tuesdayButton.setActionCommand(TuesdayString);

  JRadioButton wednesdayButton = new JRadioButton(WednesdayString);
  wednesdayButton.setActionCommand(WednesdayString);

  JRadioButton thursdayButton = new JRadioButton(ThursdayString);
  thursdayButton.setActionCommand(ThursdayString);

  JRadioButton fridayButton = new JRadioButton(FridayString);
  fridayButton.setActionCommand(FridayString);

  JRadioButton saturdayButton = new JRadioButton(SaturdayString);
  saturdayButton.setActionCommand(SaturdayString);

  JRadioButton sundayButton = new JRadioButton(SundayString);
  sundayButton.setActionCommand(SundayString);

  //Group the radio buttons.
  ButtonGroup groupDay = new ButtonGroup();
  groupDay.add(mondayButton);
  groupDay.add(tuesdayButton);
  groupDay.add(wednesdayButton);
  groupDay.add(thursdayButton);
  groupDay.add(fridayButton);
  groupDay.add(saturdayButton);
  groupDay.add(sundayButton);

//Register a listener for the radio buttons.
  mondayButton.addActionListener(this);
  tuesdayButton.addActionListener(this);
  wednesdayButton.addActionListener(this);
  thursdayButton.addActionListener(this);
  fridayButton.addActionListener(this);
  saturdayButton.addActionListener(this);
  sundayButton.addActionListener(this);
```

```
        weekday.add(mondayButton, new GridBagConstraints(0, 0, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(tuesdayButton, new GridBagConstraints(0, 1, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(wednesdayButton, new GridBagConstraints(0, 2, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(thursdayButton, new GridBagConstraints(1, 0, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(fridayButton, new GridBagConstraints(1, 1, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(saturdayButton, new GridBagConstraints(1, 2, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));
        weekday.add(sundayButton, new GridBagConstraints(2, 0, 1, 1,
        0.0, 0.0,
                    GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                    new Insets(0, 0, 5, 5), 0, 0));

        return weekday;
    }

    /**
     * creates Radio buttons initialized with Seasons
     * @return Seasons Panel
     */
    private JPanel getSeasonPanel() {
        JPanel season = new JPanel();
        season.setBorder(new
        TitledBorder(BorderFactory.createEtchedBorder(Color.white,
            new Color(148,
                      145,
                      140)),
                      "Season"));
        season.setLayout(new GridBagLayout());

      //Create the radio buttons.
      JRadioButton springButton = new JRadioButton(SpringString);
      springButton.setActionCommand(SpringString);
      springButton.setSelected(true);
      springButton.setToolTipText("April - June");

      JRadioButton summerButton = new JRadioButton(SummerString);
      summerButton.setActionCommand(SummerString);
      summerButton.setToolTipText("July - September");

      JRadioButton autumnButton = new JRadioButton(AutumnString);
      autumnButton.setActionCommand(AutumnString);
      autumnButton.setToolTipText("October - December");

      JRadioButton winterButton = new JRadioButton(WinterString);
      winterButton.setActionCommand(WinterString);
```

```
  winterButton.setToolTipText("January - March");

//Group the radio buttons.
  ButtonGroup groupSeason = new ButtonGroup();
  groupSeason.add(springButton);
  groupSeason.add(summerButton);
  groupSeason.add(autumnButton);
  groupSeason.add(winterButton);

//Register a listener for the radio buttons.
  springButton.addActionListener(this);
  summerButton.addActionListener(this);
  autumnButton.addActionListener(this);
  winterButton.addActionListener(this);

  season.add(springButton, new GridBagConstraints(0, 0, 1, 1, 0.0,
0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));
  season.add(summerButton, new GridBagConstraints(0, 1, 1, 1, 0.0,
0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));
  season.add(autumnButton, new GridBagConstraints(1, 0, 1, 1, 0.0,
0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));
  season.add(winterButton, new GridBagConstraints(1, 1, 1, 1, 0.0,
0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));

      return season;
}

/**
 * creates Radio buttons initialized with Holiday restrictions
 * @return holiday Panel
 */
private JPanel getHolidayPanel(){
      JPanel holiday = new JPanel();
      holiday.setBorder(new
      TitledBorder(BorderFactory.createEtchedBorder(Color.white,
          new Color(148,
                    145,
                    140)),
                    "Holiday"));
      holiday.setLayout(new GridBagLayout());

      //Create the radio buttons.
  JRadioButton incholidayButton = new JRadioButton(incHolidayString);
  incholidayButton.setActionCommand(incHolidayString);
  incholidayButton.setToolTipText("Include Holidays");

  JRadioButton exholidayButton = new JRadioButton(exHolidayString);
  exholidayButton.setActionCommand(exHolidayString);
  exholidayButton.setSelected(true);
  exholidayButton.setToolTipText("Exclude Holidays");

  JRadioButton onlyHolidayButton = new
  JRadioButton(onlyHolidayString);
  onlyHolidayButton.setActionCommand(onlyHolidayString);
  onlyHolidayButton.setToolTipText("Only Holidays");
```

```
//Group the radio buttons.
  ButtonGroup groupHoliday = new ButtonGroup();
  groupHoliday.add(incholidayButton);
  groupHoliday.add(exholidayButton);
  groupHoliday.add(onlyHolidayButton);

//Register a listener for the radio buttons.
  incholidayButton.addActionListener(this);
  exholidayButton.addActionListener(this);
  onlyHolidayButton.addActionListener(this);

   holiday.add(incholidayButton, new GridBagConstraints(0, 0, 1, 1,
0.0, 0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));
  holiday.add(exholidayButton, new GridBagConstraints(0, 1, 1, 1,
0.0, 0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));
  holiday.add(onlyHolidayButton, new GridBagConstraints(1, 0, 1, 1,
0.0, 0.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                  new Insets(0, 0, 5, 5), 0, 0));

  return holiday;
}

/**
 * creates text fields for entering Temperature
 * @return Temperature Panel
 */
private JPanel getTemperaturePanel(){
      JPanel temperature = new JPanel();
      temperature.setBorder(new
      TitledBorder(BorderFactory.createEtchedBorder(Color.white,
          new Color(148,
                    145,
                    140)),
                    "Average Temperature"));
      temperature.setLayout(new GridBagLayout());

      tempLabel = new JLabel("Temperature: ");

  tempValue = new JTextField(6);
  tempValue.setToolTipText("Enter the average temperature");

  /**
   * Verifier that the user can only type float Numbers
   * in the Text Field
   */
 InputVerifier verifier = new InputVerifier() {
      public boolean verify(JComponent comp) {
            boolean returnValue;
            JTextField textField = (JTextField)comp;
            try {
                  Float.valueOf( textField.getText() ).floatValue();
                  //white
                  textField.setBackground(valueIsOKColor);
                  returnValue = true;
            } catch (NumberFormatException e) {
                  // soft red
                  textField.setBackground(valueIsNotOKColor);
```

```
                returnValue = false;
            }
        return returnValue;
        }
 };

  tempValue.setInputVerifier(verifier);
  temperature.add(tempLabel, new GridBagConstraints(0, 1, 1, 1, 0.0,
0.0,
                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                new Insets(0, 0, 5, 5), 0, 0));
  temperature.add(tempValue, new GridBagConstraints(1, 1, 1, 1, 0.0,
0.0,
                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                new Insets(0, 0, 5, 5), 0, 0));

  return temperature;
}

/**
 * Creates a button for getting data from database for given inputs
 * @return Start Button
 */
private JPanel getGetPanel(){
      JPanel getB = new JPanel();
      getButton = new JButton("Start");
      getButton.setToolTipText("Create a File for the given
      Specifications");
      getButton.addActionListener(this);

  getB.add(getButton, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                new Insets(0, 0, 5, 5), 0, 0));
  return getB;

}

private JPanel getProgressBar(){
      JPanel progress = new JPanel();
      bar = new JProgressBar(0,100);
      progress.add(bar, new GridBagConstraints(0, 1, 1, 1, 0.0, 0.0,
                GridBagConstraints.CENTER, GridBagConstraints.BOTH,
                new Insets(0, 0, 5, 5), 0, 0));
      return progress;
}
/**
 * reacts on user's input.
 *
 * @see
 *java.awt.event.ActionListener#actionPerformed(java.awt.event.Action
 *Event)
 */

public void actionPerformed(ActionEvent e) {
      if(e.getActionCommand() == MondayString)
            day = 2;
      if(e.getActionCommand() == TuesdayString)
            day = 3;
      if(e.getActionCommand() == WednesdayString)
            day = 4;
      if(e.getActionCommand() == ThursdayString)
            day = 5;
      if(e.getActionCommand() == FridayString)
```

```
      day = 6;
if(e.getActionCommand() == SaturdayString)
      day = 7;
if(e.getActionCommand() == SundayString)
      day = 1;

if(e.getActionCommand() == SpringString){
      season = "1";
  startMonth = "04";
  endMonth = "05";
}
if(e.getActionCommand() == SummerString){
      season = "2";
  startMonth = "07";
  endMonth = "08";
}
if(e.getActionCommand() == AutumnString){
      season = "3";
  startMonth = "10";
  endMonth = "11";
}
if(e.getActionCommand() == WinterString){
      season = "4";
  startMonth = "01";
  endMonth = "02";
}

if(e.getActionCommand() == incHolidayString)
      holiday = "1";
if(e.getActionCommand() == exHolidayString)
      holiday = "2";
if(e.getActionCommand() == onlyHolidayString)
      holiday = "3";

if(e.getSource() == getButton){
getButton.setEnabled(false);
bar.setIndeterminate(true);
new Thread() {                //Thread for Progress bar
public void run () {    //do the following task

temperature = tempValue.getText();
CreateFile file = new CreateFile();
int[] building = list.getSelectedIndices();
double result[] = new double[1441];
double[] finalResult = new double[1440];
double[] sum = new double[1440];
String[][] data_row = new String[building.length+1][1441];
//+1 for adding up all buildings
double[][] data_value = new double[building.length][1441];
int i=0;


for(i=0; i<building.length+1;i++){
      for(int j=0; j<1441; j++){
            data_row[i][j] = "";          //empty the data rows
      }
}

for(i=0; i<building.length; i++){

String query = "SELECT DISTINCT DATA.DATA_ROW_ID, DATA.VALUE,
DATA.TIMESTAMP " + "FROM DATA " +
"WHERE DATA.DATA_ROW_ID =" + buildingID[building[i]] +
```

```
      "AND to_char(DATA.TIMESTAMP,'MM') >=" + startMonth +
      "AND to_char(DATA.TIMESTAMP,'MM') <=" + endMonth +
      "ORDER BY DATA.DATA_ROW_ID,DATA.TIMESTAMP";

      JDBCQuery data = new JDBCQuery();
      ResultSet rset;
      try {
      //query the DB using above String query
      rset = data.getData(query);
      result = calcATypicalDay(rset);    //calculate a typical day


      for(int k=0; k<finalResult.length; k++){
            finalResult[k] = result[k];
      }
      //right the building IDs at first rows
      data_row[i][0] += buildingID[building[i]];


      for(int j=1; j<finalResult.length+1; j++){
            //for each building, right the results on the following
            //rows
            data_row[i][j] += finalResult[j-1];
            //needed to calculate the sum load
            data_value[i][j] = finalResult[j-1];
      }
} catch (SQLException e1) {
JOptionPane.showMessageDialog(frame, "Columnname mismatch." + "\n" +
"Accessing wrong database." + "\n" + "Check the Config.txt
file","Error",JOptionPane.ERROR_MESSAGE);
e1.printStackTrace();
}
}

//sum ID equals to 0, put it on last column
data_row[i][0] += "0";


for(int m=1; m<finalResult.length+1; m++){
      //calculate the sum load for all
      //selected buildings
      for(int n=0; n<building.length; n++){
            sum[m-1] = sum[m-1] + data_value[n][m];
      }
}
for(int l=1; l<finalResult.length+1; l++){
      //put the sum load at the last column
      data_row[i][l] += round(sum[l-1]);
}


try {
file.writeToFile("data",data_row,finalResult.length,building.length+1
);    //write the result to CSV file
file.writeMetadata(new
Integer(day).toString(),season,temperature,holiday);
//write users input to a file

JOptionPane.showMessageDialog(frame,
"data.csv & userInput.csv." + "\n" + "successfully created",
"Message",JOptionPane.INFORMATION_MESSAGE);
} catch (IOException e1) {
JOptionPane.showMessageDialog(frame,
```

```
      "Can't Access the file." + "\n" + "Close the file and try again.",
      "Error", JOptionPane.ERROR_MESSAGE);
      e1.printStackTrace();
      }

      getButton.setEnabled(true);

      SwingUtilities.invokeLater(new Runnable() {    //task ended
      public void run () {    //disable progress bar
      bar.setIndeterminate(false);
      }
      });
      }
      } .start();
}

if(e.getSource() == selectAllButton){          //select all buidlings
      int begin =0;
      int end = list.getModel().getSize()-1;
      if(end>=0)
      {
            list.setSelectionInterval(begin, end);
      }
}

if(e.getSource() == reloadButton){         //refresh the GUI window
      frame.dispose();
      createAndShowGUI();
}
} //close event handler

      /**
       * Calculates a typical day using user's input and historical data
       * @param rset
       * @return Results of typical day algorithm
       * @throws SQLException
       */
      private double[] calcATypicalDay(ResultSet rset) throws SQLException
      {
      //all Values being 96 has been changed to 1440 -> database has been
      //changed from 15 Min Samples to 1 Min samples

      double result[] = new double[1441];
      double average[] = new double[1441];
      double[] sum = new double[1441];
      double max = 0;         // maximum load value of a day
      double min = 0;         // minimum load value of a day
      Vector<Double> loadValues = new Vector<Double>();    //dynamic array

      int numOfDays = 0;
      int newDay = 1440;      // each hour has 4 samples -> A day contains
                              //24 h * 60 Values
      Calendar c = new GregorianCalendar();
      WorkdayCalendar w;
      boolean holidayCheck;


      while(rset.next()){
            c.setTime(rset.getDate("TIMESTAMP"));
            if(c.get(Calendar.DAY_OF_WEEK) == day){
                  w = new WorkdayCalendar();          //make holiday check
                  holidayCheck = w.isHolidayDay(c.get(Calendar.YEAR),
                  c.get(Calendar.MONTH), c.get(Calendar.DATE));
```

```
            if(!holidayCheck && holiday=="2"){
                  numOfDays++;
                  loadValues.add(rset.getDouble("VALUE"));
            }

            else if(holidayCheck && holiday=="3"){
                  numOfDays++;
                  loadValues.add(rset.getDouble("VALUE"));
            }
            else if(holiday == "1"){
                  numOfDays++;
                  loadValues.add(rset.getDouble("VALUE"));
            }
      }
}
if(numOfDays == 0){
      JOptionPane.showMessageDialog(frame, "No Data found." + "\n" +
      "Change your query", "Error", JOptionPane.ERROR_MESSAGE);
      return result;
}

for(int k=0; k<1440; k++){
      sum[k] = sum[k] + loadValues.elementAt(k);
}

for(int k=0; k<loadValues.size(); k++){

      if(loadValues.size() > 1440){
            sum[k] = sum[k] + loadValues.elementAt(k+newDay);


            if(k%1440 == 0){
                  if(k!=0){
                        k=-1;
                        newDay+=1440;
                  }
            }
            if((k+newDay) == (loadValues.size()-1)){
                  break;
            }
      }
}


for(int j=0; j<sum.length; j++){
      average[j] = sum[j]/(numOfDays/1440);

      if(j==0){
            max = average[j];
            min = average[j];
      }
      else{
            if(average[j] > average[j-1])
                  max = average[j];
            if(average[j] < average[j-1])
                  min = average[j];
      }
}

for(int j=0; j<average.length; j++){
      result[j] = ((max - min)* ((average[j] - min)/(max - min))) +
      min;
}
```

```
      for(int n=0; n<result.length; n++){
            result[n] = round(result[n]);
      }
      return result;
      } //close calcATypicalDay

      /**
       * Rounds double value to 2 decimal fractions
       * @param _input
       * @return rounded double Value
       */
      static double round(double _input)
      {
        BigDecimal bd = new BigDecimal(_input);
        BigDecimal bd_round = bd.setScale( 2, BigDecimal.ROUND_HALF_UP );
        return bd_round.doubleValue();
      }


      /**
       * Create the GUI and show it.  For thread safety,
       * this method should be invoked from the
       * event-dispatching thread.
       */
      private static void createAndShowGUI() {
      //Create and set up the window.
        frame = new JFrame("Load Management");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Create and set up the content pane.
        HAWCalc newContentPane = new HAWCalc();
        newContentPane.setOpaque(true); //content panes must be opaque
        frame.setContentPane(newContentPane);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
      }

    /**
     * @param args
     */
    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```

A.5 WorkdayCalendar.java

```
package loadmanagement;

import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.Vector;


/**
 * @author Farid
 *
 */

      public class WorkdayCalendar {
        /**
         * Calculate the easter Holiday days for given year
         * @param year > 1583
         * @return Easter Sunday.
         */
         public static GregorianCalendar easterSunday( int year )
         {
          int i = year % 19;
          int j = year / 100;
          int k = year % 100;

          int l = (19 * i + j - (j / 4) - ((j - ((j + 8) / 25) + 1) / 3) +
1          5) % 30;
          int m = (32 + 2 * (j % 4) + 2 * (k / 4) - l - (k % 4)) % 7;
          int n = l + m - 7 * ((i + 11 * l + 22 * m) / 451) + 114;

          int month = n / 31;
          int day   = (n % 31) + 1;


          return  new GregorianCalendar( year, month - 1, day );
        }

      /**
       * Checks, if the given date is a Holiday.
       * @param year
       * @param month
       * @param day
       * @return true for Holiday; false for non Holiday
       */
      public boolean isHolidayDay(int year, int month, int day){
               boolean holiday = false;
               Vector <GregorianCalendar> days = new
               Vector<GregorianCalendar>(34);
               GregorianCalendar eastern = easterSunday(year);
               GregorianCalendar tmp;

               days.add(new GregorianCalendar(year,0,1));  //New Year
               tmp = (GregorianCalendar) eastern.clone();  //Karfreitag
               tmp.add(Calendar.DAY_OF_MONTH, -2);
               days.add(new
               GregorianCalendar(tmp.YEAR,tmp.MONTH,tmp.DATE));

               days.add(new
               GregorianCalendar(eastern.YEAR,eastern.MONTH,eastern.DATE));
```

```
            tmp = (GregorianCalendar) eastern.clone();  //Eastern
            tmp.add(Calendar.DAY_OF_MONTH, +1);
            days.add(new
            GregorianCalendar(tmp.YEAR,tmp.MONTH,tmp.DATE));

            days.add(new GregorianCalendar(year,4,1));  //1. May Holiday

            tmp = (GregorianCalendar) eastern.clone();  //Himmelfahrt
            tmp.add(Calendar.DAY_OF_MONTH, +39);
            days.add(new
            GregorianCalendar(tmp.YEAR,tmp.MONTH,tmp.DATE));

            tmp = (GregorianCalendar) eastern.clone();  //Pfingsten
            tmp.add(Calendar.DAY_OF_MONTH, +49);
            days.add(new
            GregorianCalendar(tmp.YEAR,tmp.MONTH,tmp.DATE));
            tmp.add(Calendar.DAY_OF_MONTH, +1);
            days.add(new
            GregorianCalendar(tmp.YEAR,tmp.MONTH,tmp.DATE));

            //Tag der deutschen Einheit
            days.add(new GregorianCalendar(year,9,3));
            days.add(new GregorianCalendar(year,11,24)); //Heiligabend
            days.add(new GregorianCalendar(year,11,25));//Christmas
            days.add(new GregorianCalendar(year,11,26));
            days.add(new GregorianCalendar(year,11,31));//Silvester

            holiday = days.contains(new
            GregorianCalendar(year,month,day));

            return holiday;
    }
}
```

A.6 CreateFile.java

```
package loadmanagement;

import java.io.FileWriter;
import java.io.IOException;

/**
 * @author Farid
 *
 */
public class CreateFile {

	/**
	 * Writes the typical Day load values to a *.csv file
	 * @param temp_data
	 * @param ResultLength
	 * @param buildingLength
	 * @throws IOException
	 */
	public void writeToFile(String fileName,String temp_data[][],int
	ResultLength, int buildingLength) throws IOException{
	FileWriter fw;
	fw = new FileWriter(fileName + ".csv");
	String data_row = "";

	for(int n=0; n<ResultLength+1; n++){
		for(int i=0; i<buildingLength; i++){
			data_row += temp_data[i][n];
			data_row += ",";
			if(i+1 == buildingLength)
				data_row += "\n";
		}
	}
	fw.write(data_row);
	fw.close();
	}

	  /**
	   * Writes user's input to a *.csv file
	   * @param day
	   * @param season
	   * @param temperature
	   * @param holiday
	   * @throws IOException
	   */
	      public void writeMetadata(String day, String season, String
	      temperature , String holiday) throws IOException {
	      String fileName = "userInput.csv";
	      FileWriter fw = new FileWriter(fileName);
	      String data_row = "";
	      fw.write("DAY,SEASON,HOLIDAY");
	      data_row = "\n";
	      data_row += day;
	      data_row += ",\"" + season + "\"";
	      //data_row += ",\"" + temperature + "\"";
	      data_row += ",\"" + holiday + "\"";
	      fw.write(data_row);
	      fw.close();
	  }
}
```

A.7 userInput.csv

```
DAY,SEASON,HOLIDAY
2,"1","1"
```

A.8 data.csv

```
9,0,
501.46,501.46,
480.66,480.66,
501.03,501.03,
481.08,481.08,
503.06,503.06,
479.05,479.05,
502.36,502.36,
479.75,479.75,
501.96,501.96,
480.16,480.16,
503.05,503.05,
479.06,479.06,
501.31,501.31,
480.8,480.8,
501.36,501.36,
699.63,699.63,
676.6,676.6,
698.54,698.54,
677.68,677.68,
700.64,700.64,
675.58,675.58,
697.41,697.41,
678.82,678.82,
695.85,695.85,
680.37,680.37,
695.78,695.78,
680.44,680.44,
697.76,697.76,
678.47,678.47,
699.11,699.11,
699.68,699.68,
678.03,678.03,
702.29,702.29,
675.42,675.42,
697.71,697.71,
….
```

Declaration

I, Farid Naimi, declare within the meaning of section 25(4) of the Examination and Study Regulations of the International Degree Course Information Engineering that: this Master report has been completed by myself independently without outside help and only the defined sources and study aids were used. Sections that reflect the thoughts or works of others are made known through the definition of sources.

Hamburg, 20th March 2008_____